Development, implementation and demonstration of a reference processing pipeline for the future production of official statistics based on multiple Mobile Network Operator data (TSS multi-MNO)

Service Contract Number – 2021.0400

D4.2 – Second code release and documentation

In association with:

Development, implementation and demonstration of a reference processing pipeline for the future production of official statistics based on multiple Mobile Network Operator data (TSS multi-MNO)

Service Contract Number – 2021.0400

Deliverable 4.2: Second code release and documentation

Version number: final

Date: 29 November 2024

The information and views set out in this publication are those of the authors and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

Cover Photo by wd toroMC from Pexels

\ ABSTRACT

The Multi-MNO project aims to **develop, implement and demonstrate a proposal for a reference standard processing pipeline for the future production of official statistics in Europe based on Mobile Network Operator (MNO) data from multiple operators**. The term "processing pipeline" refers to the combination of a methodological framework and a reference open-source software adhering to such a framework. This report presents the list of software artefacts developed within the Multi-MNO project based on the requirements and specifications provided by the methodological framework defined in deliverables D2.2 and D3.1. The software artefacts include: (i) open-source software and testing datasets, (ii) technical documentation (including software requirements, design and tests), (iii) developers guide and (iv) user manual. This report provides detailed information about the software technological stack, requirements and design. Rest of software artefacts (code, testing datasets, user manual, developer guide, etc.) are provided in the project's Github repository publicly available at the following link: https://github.com/eurostat/multimno.

Authors: The report was prepared under the technical coordination of Miguel Picornell (Senior Expert, Nommon). Further contributions, expertise and research support were provided by: (in alphabetical order) Gabriele Ascari, Oliver Bollverk, Roberto Cenciotti, Erika Cerasti, Loredana Di Consiglio, Cristina Faricelli, Ricardo Herranz, Paolo Mattera, Miguel Picornell, Roberta Radini, Giorgia Simeoni, Margus Tiru, Villem Tonnison, Tiziana Tuoto, Luca Valentino and Ivan Vasilyev. Specific contributions for the geolocation and continuous time segmentation modules were provided by: Edwin de Jonge, Jan van der Laan, Matthias Offermans and Martijn Tennekes.

Acknowledgments: The authors express their gratitude for the technical review and coordination ensured by Fabio Ricciato, responsible for steering the project on behalf of Eurostat.

This report was prepared in the context of the service contract ref. 2021.0400 awarded by Eurostat to the consortium led by GOPA (Germany), in collaboration with the industry partners NOMMON (Spain) and POSITIUM (Estonia), and the National Statistical Institutes ISTAT (Italy) and CBS (Netherlands).

On behalf of the contractor, project management is ensured by Florabela Carausu (GOPA).

DOCUMENT VERSION STATUS AND FUTURE UPDATES:

The document is a work-in-progress interim version of the first project deliverable. Therefore, its content may change in future versions. This document and any future updates will be publicly disseminated on the Multi-MNO project webpage: https://cros.ec.europa.eu/multi-mno-project

Readers are invited to submit comments and corrections or share their views via email to multimno-project@gopa.de

Abbreviations

5G	Fifth-generation technology
AWS	Amazon Web Services
EMR	Elastic MapReduce
ESS	European Statistical System
GCP	Google Cloud Platform
GSM	Global System for Mobile Communications
IDE	Integrated Development Environment
ISO	International Organisation for Standardisation
LCL	Lower Control Limit
LTE	Long Term Evolution (mobile networks)
MNO	Mobile Network Operator
n/a	not applicable
PEP 8	Python Enhancement Proposal 8
QW	Quality Warnings
UCL	Upper Control Limit
UMTS	Universal Mobile Telecommunications System
UTC	Universal Time Coordinated

Table of Contents

Index of Figures

Index of Tables

1 INTRODUCTION

1.1 BACKGROUND AND OBJECTIVES

The Multi-MNO project aims to develop, implement and demonstrate a proposal for a reference standard processing pipeline for the future production of official statistics in Europe based on Mobile Network Operator (MNO) data from multiple operators. If successful, the proposal developed by the project may be endorsed as European Statistical System (ESS) standard by the relevant ESS bodies. The term 'processing pipeline' refers to the combination of a methodological framework and a reference open-source software adhering to such a framework. The methodological framework mainly includes the definition of scenarios, use cases, methods, business processes and a quality framework. Detailed information about the methodological framework developed in this project is provided in the following documentation:

- **D2.2- Updated version of technical documentation for scenarios, requirements, use cases and methods, and high-level architecture**
- **D3.1- Interim version of technical documentation for Business Processes and Quality Framework**

Based on the requirements and specifications derived from the methodological framework, an open-source software for the production of official statistics has been developed.

1.2 SCOPE OF THE DOCUMENT

This document presents the complete list of the software artefacts developed within the Multi-MNO project, providing detailed information about the software technological stack, requirements and design. Rest of software documentation (user manual, developer guide, etc.) is provided in the project's Github repository publicly available at the following link: https://github.com/eurostat/multimno.

*[**Remark:** The documentation provided so far covers the scope of the multi-MNO release 0.3. The documentation will be updated as long as new releases are published]*

1.3 DOCUMENT STRUCTURE

In addition to this introductory section, the remainder of this document is organised as follows:

- **Chapter 2 'Overview of software artefacts and mapping with the methodological framework':** presents the list of artefacts developed within the project and provides a mapping between the software components and the methods described in D2.2
- **Chapter 3 'Software requirement specification':** provides the requirements of the software, addressing both general requirements and component-specific requirements.
- **Chapter 4 'Technological Stack':** describes the technology stack established for the software, providing a rationale for the decision taken.
- **Chapter 5 'Design':** provides the design of the software, addressing both general design and component-specific design.
- **Annex I 'Data objects':** presents a detailed description of the data objects generated by the software.
- **Annex II 'Notes for future revision':** annotates a partial list of pending points for improvement in future releases of this deliverable.

2 OVERVIEW OF SOFTWARE ARTEFACTS AND MAPPING TO THE METHODOLOGICAL FRAMEWORK

2.1 SOFTWARE ARTEFACTS AND REPOSITORY STRUCTURE

Table 1 presents the list of software artefacts developed in this project. Main artefacts cover: (i) open-source software and testing datasets, (ii) technical documentation (including software requirements, design and tests), (iii) developers guide, to facilitate maintenance and the future development of the software and (iv) user manual, to guide the deployment and execution of the software. Most of the documentation is publicly available in the project GitHub repository: https://github.com/eurostat/multimno. The GitHub repository includes a README.md file to facilitate the navigation through the software documentation, as well as HTML documentation that enables interactive web-based navigation.

Table 1: List of software artefacts and location of the associated documentation

ARTEFACTS	DESCRIPTION	REPOSITORY
Open-source software	• Open-source code • Synthetic datasets (inputs and components outputs for a set of scenarios)	https://github.com/eurostat/multimno
Technical documentation	• Requirement specifications • Software design	Present document, Chapter 3 (requirements) and Chapter 5 (design)
	• Testing codes and documentation	https://github.com/eurostat/multimno
Developers guide	• Guide for future developers to be able to contribute to the software • Methods and functions documentation as well as code-level comments	https://github.com/eurostat/multimno
User manual	• (how-to guide) on the deployment • (how-to guide) use of the software (data and infrastructure requirements, installation of the software, how to configure and run the software, etc.).	https://github.com/eurostat/multimno

2.2 MAPPING TO THE METHODOLOGICAL FRAMEWORK

Table 2 shows the software components developed for the multi-MNO codes release 0.3 and their correspondence with the methods described in the deliverables from Task 2. The software developed consists of a set of components covering the different use cases defined in Task 2. Software components usually cover one or more functionalities described in Task 2 methods (e.g. the component 'CellConnectionProbabilityEstimation' covers the functionalities described by the methods: 'Cell Connection Probability Estimation Module' and 'Posterior Probability Estimation Module'). Note that it could be the case that a software component is not directly related with any method described in Task 2, nonetheless its implementation is needed for the correct functioning of the solution (e.g. 'InspireGridGenerator' module creates the INSPIRE grid that is used as the reference grid for posterior analyses).

Table 2: Mapping of the software modules in the multi-MNO codes release 0.3 to the methods described in the deliverables from Task 2

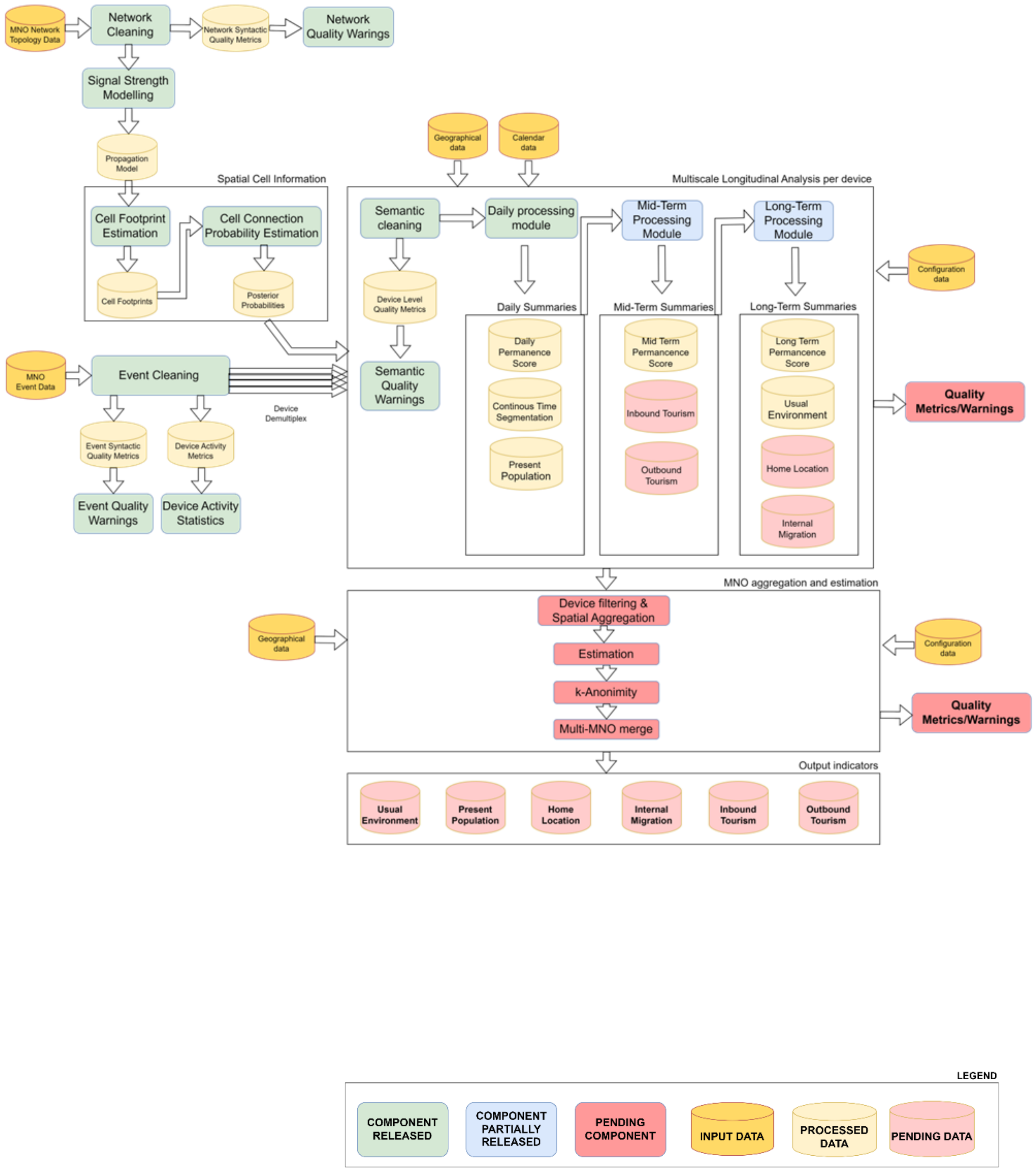
#	SOFTWARE COMPONENTS	METHOD NAME IN TASK 2
1	NetworkCleaning	1.1 Cleaning of MNO Network Topology Data
2	NetworkQualityWarnings	2.1 Generation of MNO Network Topology Data Syntactic Quality Warnings
3	SignalStrengthModeling	3.1 Propagation Estimation Module
4	CellFootprintEstimation	4.1 Cell Footprint Estimation Module
5	CellConnectionProbabilityEstimation	• 5.1 Cell Connection Probability Estimation Module • 6.1 Posterior Probability Estimation Module
6	EventCleaning	• 7.1 Cleaning of MNO Event Data - Syntactic Checks • 9.1 Demultiplexing of MNO Event Data
7	EventQualityWarnings	8.1 Generation of MNO Event Data Syntactic Quality Warnings
8	DeviceActivityStatistics	9.1 Generation of Device Activity Quality Warnings
9	SemanticCleaning	11.1 Cleaning of MNO Event Data at Device Level - Semantic Checks
10	SemanticQualityWarnings	12.1 Generation of MNO Event Data at Device Level Semantic Quality Warnings
11	DailyPermanenceScore	13.2 Estimation of the permanence score for usual environment and home location assignation
12	ContinuousTimeSegmentation	13.3 Estimation of Continuous Time Segmentation
14	InspireGridGenerator	n/a
15	SyntheticDiaries	n/a
16	SyntheticNetwork	n/a
17	PresentPopulation	13.1 Present Population Estimation
18	GridEnrichment	n/a
19	GeoZonesGridMapping	n/a
20	MidTermPermanenceScore	14. Mid-Term Processing Module
21	LongTermPermanenceScore	15. Long-Term Processing Module
22	UsualEnvironmentLabelling	15. Long-Term Processing Module
23	UsualEnvironmentAggregation	15. Long-Term Processing Module

***[Remark -** This document contains the information related to the software release 0.3. that covers part of the methods described in the deliverables from Task 2. This section will be updated as new components of the pipeline are developed.]*

2.3 GENERAL OVERVIEW OF THE SOFTWARE SOLUTION DESIGN AND RELEASE STATUS

In this section, we provide an overview of the software solution design and status of the multi-MNO release 0.3 implementation. The following diagram provides a high-level overview of the software pipeline. The components are presented in boxes of different colours to indicate the current development status: (i) green means that the component is available in release 0.3, (ii) blue means that the component is available in release 0.3 but does not contain yet all the functionalities planned within the project framework, and (iii) red means that the component is pending implementation and will be available in future releases. On the other hand, the data is represented by cylindrical objects in various colors: (i) bright yellow indicates that the data is an input to the process, (ii) light yellow indicates that the data was generated during the process, and (iii) red indicates that the data is not yet available in release 0.3 but will be available once development is completed. It is important to note that the available or partially available components may be improved throughout the project's lifecycle. The enhancements to be incorporated will mainly be driven by the tests conducted in real-world environments with different MNOs.

Figure 1: Design and status of the multi-MNO codes release 0.3 compared to the methodological framework



3 SOFTWARE REQUIREMENTS

This chapter describes all the functional and non-functional requirements that each software module of the pipeline must fulfil. A requirement is a singular documented physical and functional need that a particular design, product or process must be able to perform. In the definition of the software requirements it is important to comply with the following set of rules:

1. **Clear and Unambiguous**: requirements should be expressed in a clear and unambiguous manner, leaving no room for interpretation. Ambiguity can lead to misunderstandings and errors.
2. **Complete**: requirements should cover all necessary aspects of the software's functionality, leaving no critical features or behaviours undocumented.
3. **Consistent**: requirements should not contradict each other, and they should align with the project's goals and constraints. Inconsistencies can lead to confusion and conflicts.
4. **Feasible**: requirements should be technically achievable within the project's constraints, including time, budget, and available resources.
5. **Measurable**: requirements should be quantifiable so that they can be objectively verified during testing or upon delivery. This often involves specifying criteria for success.
6. **Testable**: requirements should be written in a way that allows for effective testing. Test cases should be derived directly from the requirements to ensure thorough testing coverage.
7. **Modular**: requirements should be modular and encapsulate individual pieces of functionalities or features. This modularity simplifies development and maintenance.
8. **Traceable**: requirements should be traceable throughout the software development lifecycle, from the initial concept to the final implementation. Traceability ensures that all requirements are met.
9. **Approved**: requirements should go through an approval process by relevant stakeholders to ensure that they accurately represent their needs and expectations.
10. **Non-Functional Requirements**: these include aspects like performance, security, scalability, usability, and reliability, in addition to functional requirements.
11. **Constraints**: requirements should identify any constraints, such as regulatory, hardware, or budget limitations, that may impact the project.

General requirements (Section 3.1 General requirements) as well as specific module requirements (Section 3.2 Component requirements) are provided. Requirements are provided using a table template (see Table 3) that contains the following information:

- **ID**: requirement identifier with the following naming 'TSS-AAA-NNN', where 'AAA' is the abbreviation of the requirements group (e.g. 'GEN' refers to 'general' requirements) and 'NNN' the number of the requirement within the requirement group (e.g. '001' for the first requirement). Must be unique.
- **Definition**: requirement specification. Must be atomic and not ambiguous.

Table 3: Requirements table template with examples

ID	DEFINITION
TSS-MNO-001	Timestamp data shall be given in the UTC standard.
TSS-MNO-002	The pipeline shall process a single combination of MNO and country data at each instance within the MNO infrastructure

3.1 GENERAL REQUIREMENTS

The general requirements, covering the functional, infrastructure, software, data and performance dimensions are described in this section. Some of the requirements are derived by the fact that Big Data sources are used in the calculation of the indicators[1]. Therefore, the software must be executable within the Apache Big Data ecosystem (Hadoop, Hive, Spark…). Other requirements consider the convenience of using state-of-the-art infrastructure for Big Data analyses (e.g. the use of cloud environments like AWS, GCP or Azure) or the need of local execution (in conventional laptop and desktops) with synthetic datasets for development and/or demonstration scenarios purposes. A complete list of the general requirements considered for the development of the software is presented in Table 4.

Table 4: Software general requirements classified by category

ID	DEFINITION
Functional	
TSS-GEN-001	The software shall record insights of the data transformations performed in the components in their respective quality metrics' data objects.
TSS-GEN-002	The software shall process a single combination of MNO and country data at each instance within the MNO infrastructure.
TSS-GEN-003	Timestamp data shall be given in the UTC standard.
TSS-GEN-004	The software shall be able to generate synthetic data for an end-to-end pipeline execution.
TSS-GEN-005	The software shall generate data quality indicators for a set of pipeline components.
Infrastructure	
TSS-GEN-006	The software shall be executable in cloud environments (AWS, GCP, Azure) of MNO Operators.
TSS-GEN-007	The software shall use the Spark framework for big data processing.
TSS-GEN-008	The software shall be executable in a single computer.
TSS-GEN-009	The software shall be executable in Windows, Linux and Mac operating systems.
Software	
TSS-GEN-010	The software shall execute a pipeline of isolated components which do not share in-memory information between them.[2]

[1] The Multi-MNO project introduces several use cases that involve the processing of MNO data and lists for each use case the statistical indicators that can be produced. In the reports from the project, the terms statistical indicator and indicator are used interchangeably. The use cases and targeted statistical indicators are detailed in the project deliverables D2.

[2] Rationale: by resetting both the Spark Session and the Python cache before the execution of each component, the isolated components paradigm helps maintaining the integrity and predictability of the PySpark application. It ensures that each component can be executed independently, without being influenced by the state or results of other components. This promotes modularity, simplifies debugging, and enhances overall reliability.

ID	DEFINITION
TSS-GEN-011	The software shall use a general configuration file and a specific component configuration file for each of the components in the pipeline.
TSS-GEN-012	Configuration files shall be in INI format.
TSS-GEN-013	If the same configuration value is specified in the component configuration file and the general configuration file, the component configuration value shall be the one to be used by the software.
TSS-GEN-014	Each component execution shall be performed through a *spark-submit* command.
TSS-GEN-015	All software dependencies shall be open source.
TSS-GEN-016	All software dependencies shall be free to use.
TSS-GEN-017	The software shall be able to perform spatial computations in a distributed environment.
TSS-GEN-018	The software shall be able to generate code documentation from docstrings.
TSS-GEN-019	The software shall be implemented with modular components following the object-oriented paradigm.
TSS-GEN-020	The software shall be open source and stored in a public repository.
TSS-GEN-021	The software shall use the European Union public license v. 1.2.
TSS-GEN-029	Each component shall log the configuration used in a log file at the start of its execution.
Data	
TSS-GEN-022	The software shall write intermediate and output data in parquet format file.
TSS-GEN-023	If data to be written contains a geometry column, the software shall write intermediate and output data in geoparquet format file.
TSS-GEN-024	The software shall be able to read and write in local filesystems and distributed filesystems (HDFS, AWS S3, GCP, Azure).
TSS-GEN-025	The software shall be able to ingest reference data in csv, json, txt, shapefile and geojson formats.
TSS-GEN-026	The software shall use a spatial grid following the INSPIRE specification for representing spatial data in intermediate calculations through the pipeline.
TSS-GEN-027	Input data, configuration data and all output data generated by a demo execution of an end-to-end pipeline shall be provided in the code repository.
Performance	
TSS-GEN-028	The software shall be able to execute an end-to-end pipeline for any of the use cases and for the whole national territory of the MNO in less than 24 hours.

3.2 COMPONENT REQUIREMENTS

*[**Remark** - This section contains the requirements for the components available in the release 0.3 of the software.]*

3.2.1 NetworkCleaning

ID	DEFINITION
TSS-NET-001	The software shall read network topology input data from parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-NET-002	The software shall be able to write processed network topology data in parquet format, partitioned by year (YYYY), month (MM), and day (DD).
TSS-NET-003	The software shall be able to write processed network topology top frequent errors data in parquet format, partitioned by year (YYYY), month (MM), and day (DD).
TSS-NET-004	The software shall be able to write processed network topology row error metrics data in parquet format, partitioned by year (YYYY), month (MM), and day (DD).
TSS-NET-005	The software shall check that all the mandatory columns specified in Annex I - I.7 Cell Locations with Physical Properties - Raw exist in the input data.
TSS-NET-006	The software shall be able to read network topology input data with the data type scheme specified in Annex I - I.7 Cell Locations with Physical Properties - Raw.
TSS-NET-007	The software shall write output network topology data following the data type scheme specified in Annex I - I.8 Cell Locations with Physical Properties – Cleaned.
TSS-NET-008	The software shall write output syntactic quality metrics following the data type scheme specified in Annex I - I.9 MNO Network Topology Data Quality Metrics.
TSS-NET-009	The software shall write output top frequent error data following the data type scheme specified in Annex I - I.26 MNO Network Topology Top Frequent Erros.
	The software shall write output row error metrics following the data type scheme specified in Annex I – I.27 MNO Network Topology Row Error Metrics.
TSS-NET-10	The software shall discard records where any of the mandatory fields are null.
TSS-NET-011	The software shall discard records where the cell_id field is not a string of length 14 or 15.
TSS-NET-012	The software shall impute a null value in records where the valid_date_start cannot be parsed as a valid timestamp following the ISO:8601 format YYYY-MM-DDThh:mm.ss.
TSS-NET-013	The software shall impute a null value in records where the valid_date_end, if it is not null, cannot be parsed as a valid timestamp following the ISO:8601 format YYYY-MM-DDThh:mm.ss.
TSS-NET-014	The software shall impute null values where the valid_date_start and valid_date_end fields are both non-null, can be parsed to timestamp, and the valid_date_end is an earlier point in time than the valid_end_start.
TSS-NET-015	The software shall discard records where the latitude field is not within the configuration-specified bounding box.
TSS-NET-016	The software shall discard records where the longitude field is not within the configuration-specified bounding box.
TSS-NET-017	The software shall discard records where the antenna_height is less than or equal to 0.
TSS-NET-018	The software shall discard records where the directionality is not equal to either 0 or 1.
TSS-NET-019	The software shall discard records where the azimuth_angle field is null and the directionality field is equal to 1.
TSS-NET-020	The software shall discard records where the azimuth angle is less than 0 or greater than 360, and the directionality field is equal to 1.
TSS-NET-021	The software shall discard records where the elevation_angle is less than -90 or greater than 90.
TSS-NET-022	The software shall discard records where the horizontal_beam_width is less than 0 or greater than 360.
TSS-NET-023	The software shall discard records where the vertical_beam_width is less than 0 or greater than 360.
TSS-NET-024	The software shall discard records where the power is equal to or less than 0.
TSS-NET-025	The software shall impute a null value in records where the range is equal to or less than 0.
TSS-NET-026	The software shall impute a null value in records where the frequency is equal to or less than 0.
TSS-NET-027	The software shall impute a null value in records where the technology is not equal to one of the allowed configuration-specified values.

ID	DEFINITION
TSS-NET-028	The software shall impute a null value in records where the cell_type is not equal to one of the allowed configuration-specified values.
TSS-NET-029	The software shall record the time when the component was executed and save it as the result_timestamp field of the output quality metrics data.
TSS-NET-030	The software shall count the number of records that the input network topology dataset had before performing any transformation or check.
TSS-NET-031	The software shall record a quality metric with the number of registers in the original input network topology dataset, with a field_name of null, and a type_code equal to the "total rows at the start of the method" corresponding error code.
TSS-NET-032	The software shall count the number of records that the output network topology dataset has after all transformations and checks are performed.
TSS-NET-033	The software shall count the number of records that are deleted after the transformations and checks are performed.
TSS-NET-034	The software shall count the number of records that had any erroneous or missing value in any of its fields.
TSS-NET-035	The software shall record a quality metric with the number of registers in the original input network topology dataset, with a field_name of null, and a type_code equal to the "total rows at the end of the method" corresponding error code.
TSS-NET-036	The software shall count, for each of the fields of the input data object I.7 Cell Locations with Physical Properties - Raw (see Annex I), the number of records that had a correct value for that field.
TSS-NET-037	The software shall record a quality metric with the number of correct values in a given field, with a field_name equal to that field's value, and a type_code equal to the "no error" corresponding error code.
TSS-NET-038	The software shall count, for each of the fields of the input data object I.7 Cell Locations with Physical Properties - Raw (see Annex I), the number of records that had a non-admitted null value for that field.
TSS-NET-039	The software shall record a quality metric with the number of null values in a given field, with a field_name equal to that field's value, and a type_code equal to the "null error" corresponding error code.
TSS-NET-040	The software shall count, for each applicable field of the input data I.7 Cell Locations with Physical Properties - Raw (see Annex I), the number of records that had a value that could not be parsed.
TSS-NET-041	The software shall record a quality metric with the number of non-null values that could not be parsed in a given field, with a field_name equal to that field's value, and a type_code equal to the "could not parse" corresponding error code.
TSS-NET-042	The software shall count, for each applicable field of the input data object I.7 Cell Locations with Physical Properties - Raw (see Annex I), the number of records that had a value outside of the accepted value range.
TSS-NET-043	The software shall record a quality metric with the number of non-null values that could not be parsed in a given field, with a field_name equal to that field's value, and a type_code equal to the "out of range" corresponding error code.
TSS-NET-044	The software shall record a quality metric with the number of registers with non-null valid_date_start and valid_date_end fields such that valid_date_end was an earlier point in time than valid_date_start, with a field_name of null, and a type_code equal to the "out of range" corresponding error code.
TSS-NET-045	The software shall be able to count the number of invalid entries found for each of the fields of the input data, as well as the frequency of each particular invalid value.
TSS-NET-046	The software shall be able to record the top *k* most frequent invalid values found in the input data, where *k* is an integer representing the number of most frequent values to record specified via configuration, whenever an absolute number of the most frequent invalid values is indicated via configuration.
TSS-NET-047	The software shall be able to record the most frequent invalid values found in the input data that represent a *k* percentage of all total invalid values ordered by absolute frequency, where *k* is a number larger than 0 and equal or less than 100, whenever a percentage number of the most frequent invalid values is indicated via configuration.

ID	DEFINITION
TSS-NET-048	The software shall read via configuration the parameter *k* and the parameter frequent_error_criterion, indicating whether the top *k* most frequent invalid values or the most frequent invalid values covering *k* percentage of all invalid values must be recorded.
TSS-NET-049	The software shall read via configuration the float parameter latitude_min that will define the bounding box used to check for out-of-range values.
TSS-NET-050	The software shall read via configuration the float parameter latitude_max that will define the bounding box used to check for out-of-range values.
TSS-NET-051	The software shall read via configuration the float parameter longitude_min that will define the bounding box used to check for out-of-range values.
TSS-NET-052	The software shall read via configuration the float parameter longitude_max that will define the bounding box used to check for out-of-range values.
TSS-NET-053	The software shall read via configuration the comma-separated parameter cell_type_options that will define accepted values of the cell_type field.
TSS-NET-054	The software shall read via configuration the comma-separated parameter technology_options that will define accepted values of the technology field.
TSS-NET-055	The software shall read via configuration the date parameter data_period_start, the starting date (included) for which data is to be processed.
TSS-NET-056	The software shall read via configuration the date parameter data_period_end, the ending date (included) for which data is to be processed.

3.2.2 NetworkQualityWarnings

ID	DEFINITION
TSS-NQW-001	The software shall read network topology syntactic quality metrics input data from parquet files stored partitioned by year (YYYY), month (MM) and day (DD).
TSS-NQW-002	The software shall be able to write quality warnings log table in parquet format, partitioned by year (YYYY), month (MM), and day (DD).
TSS-NQW-003	The software shall be able to write line plot data in parquet format, partitioned by variable, year (YYYY), month (MM), day (DD), and execution timestamp.
TSS-NQW-004	The software shall be able to write pie plot data in parquet format, partitioned by variable, year (YYYY), month (MM), day (DD), and execution timestamp.
TSS-NQW-005	The software shall be able to read network topology syntactic quality metrics data following the data type scheme specified in Annex I - I.9 MNO Network Topology Data Quality Metrics.
TSS-NQW-006	The software shall be able to write the output quality warnings log table following the data type scheme specified in Annex I - I.10 MNO Network Topology Data Quality Warnings – log table.
TSS-NQW-007	The software shall be able to write the output line plot data following the data type scheme specified in Annex I - I.23 MNO Network Syntactic Quality Warnings Line Plot Data.
TSS-NQW-008	The software shall be able to write the output pie plot data following the data type scheme specified in Annex I - I.24 MNO Network Syntactic Quality Warnings Pie Plot Data.
TSS-NQW-009	The software shall check that all the metrics for the current date, as well as for the previous period used for comparison, exist in the input data, and stop the execution if they do not.
TSS-NQW-010	The software shall be able to compute the average value of every quality metric for the previous period used for comparison.
TSS-NQW-011	The software shall be able to compute the sample standard deviation of every quality metric for the previous period used for comparison.
TSS-NQW-012	The software shall record, for each quality warning, the date of execution of the quality warnings component.
TSS-NQW-013	The software shall record, for each quality warning, the study date of the metric that raised the warning.
TSS-NQW-014	The software shall record, for each quality warning, the value of the metric that raised the warning.
TSS-NQW-015	The software shall record, for each quality warning, the value of the threshold crossed by the metric that raised the warning.
TSS-NQW-016	The software shall record, for each quality warning, the condition that had to be checked in order to raise the warning
TSS-NQW-017	The software shall record, for each quality warning, a warning text giving context to the raised warning.
TSS-NQW-018	The software shall be able to create a warning when the number of rows before the syntactic checks in the study date, N, verifies some of the following: a) 100 * (N - AVG) / AVG > T1, b) 100 * (N - AVG) / AVG < T2, c) N > AVG + T3 * S, d) N < AVG - T3 * S, e) N > T4, f) N < T5; where AVG and S are the average and standard deviation of the number of rows before the syntactic checks over the previous period respectively, T1 and T2 are percentage thresholds, T3 is a number-of-standard-deviations threshold, and T4 and T5 are absolute thresholds.
TSS-NQW-019	The software shall be able to create a warning when the number of rows after the syntactic checks in the study date, N, verifies some of the following: a) 100 * (N - AVG) / AVG > T1, b) 100 * (N - AVG) / AVG < T2, c) N > AVG + T3 * S, d) N < AVG - T3 * S, e) N > T4, f) N < T5; where AVG and S are, respectively, the average and standard deviation of the number of rows before the syntactic checks over the previous period, T1 and T2 are percentage thresholds, T3 is a number-of-standard-deviations threshold, and T4 and T5 are absolute thresholds.
TSS-NQW-020	The software shall be able to create a warning when the error rate in the study date, E, verifies some of the following: a) 100 * (E - AVG)/AVG > T1, b) E > AVG + T2 * S, c) E > T3; where AVG and S are, respectively, the average and standard deviation of the error rate over the previous period, T1 is a percentage threshold, T2 is a number-of-standard-deviations threshold, and T3 is an absolute threshold.

ID	DEFINITION
TSS-NQW-021	The software shall be able to create a warning when the rate of missing values for any applicable field in the study date, M, verifies some of the following: a) 100 * (M - AVG)/AVG > T1, b) M > AVG + T2 * S, c) M > T3; where AVG and S are, respectively, the average and standard deviation of the rate of missing values for a given field over the previous period, T1 is a percentage threshold, T2 is a number-of-standard-deviations threshold, and T3 is an absolute threshold.
TSS-NWQ-022	The software shall be able to create a warning when the rate of out-of-range values for any applicable field in the study date, R, verifies some of the following: a) 100 * (R - AVG)/AVG > T1, b) R > AVG + T2 * S, c) R > T3; where AVG and S are, respectively, the average and standard deviation of the rate of out-of-range values for a given field over the previous period, T1 is a percentage threshold, T2 is a number-of-standard-deviations threshold, and T3 is an absolute threshold.
TSS-NQW-023	The software shall be able to create a warning when the rate of parsing errors for any applicable field in the study date, P, verifies some of the following: a) 100 * (P - AVG)/AVG > T1, b) P > AVG + T2 * S, c) P > T3; where AVG and S are, respectively, the average and standard deviation of the rate of parsing errors for a given field over the previous period, T1 is a percentage threshold, T2 is a number-of-standard-deviations threshold, and T3 is an absolute threshold.
TSS-NQW-024	The software shall be able to write into a parquet file the necessary data to create a line plot showing the time evolution of the number of rows before the syntactic checks over the previous period and the study date, along with the average, upper control limit, and lower control limit over the previous period.
TSS-NQW-025	The software shall be able to write into a parquet file the necessary data to create a line plot showing the time evolution of the number of rows after the syntactic checks over the previous period and the study date, along with the average, upper control limit, and lower control limit over the previous period.
TSS-NQW-026	The software shall be able to write into a parquet file the necessary data to create a line plot showing the time evolution of the error rate over the previous period and the study date, along with the average and upper control limit over the previous period.
TSS-NQW-027	The software shall be able to write in a parquet file the necessary data to create, for each field of the network topology data, a pie plot showing the percentage distribution of errors for that field in the current date.
TSS-NQW-028	The software shall be able to read the extent of the lookback period from a configuration file with the following options: “week” (7 days), “month” (30 days), and ”quarter” (90 days).
TSS-NQW-029	The software shall be able to read from a configuration file the percentage threshold over the average for all quality metrics, one per metric.
TSS-NQW-030	The software shall be able to read from a configuration file the percentage threshold under the average for the number of rows before the syntactic checks and the average number of rows after the syntactic checks, one for each metric.
TSS-NQW-031	The software shall be able to read from a configuration file the number of standard deviations threshold over the average for all quality metrics, one per metric.
TSS-NQW-032	The software shall be able to read from a configuration file the number of standard deviations threshold under the average for the number of rows before the syntactic checks and the average number of rows after the syntactic checks, one for each metric.
TSS-NQW-033	The software shall be able to read from a configuration file the absolute threshold over the average for all quality metrics, one per metric.
TSS-NQW-034	The software shall be able to read from a configuration file the absolute threshold under the average for all quality metrics, for the number of rows before the syntactic checks and the average number of rows after the syntactic checks, one for each metric.
TSS-NQW-035	The software shall contain default values for every threshold to be used in case they were not specified via configuration file.

3.2.3 SIGNALSTRENGTHMODELING

ID	DEFINITION
TSS-SSE-001	The software shall read input data objects from parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-SSE-002	The software shall read INSPIRE grid data from geoparquet files.
TSS-SSE-003	The input shall be I.8 Cell Locations with Physical Properties – Cleaned and I.11 Reference Grid Data Objects (in Annex I).
TSS-SSE-004	The output shall be I.12 Cells Signal Strengths Data Object (in Annex I).
TSS-SSE-005	The software shall read input data for a date range based on the configuration parameter.
TSS-SSE-006	The software shall perform all processing steps for each date in the given date range independently.
TSS-SSE-007	The software shall write output data into parquet format partitioned by year, month, and day.
TSS-SSE-008	The software shall verify the presence of all required attributes of cells for signal strength propagation modeling. The required attributes are: power, antenna_height. If directionality is equal 1, then elevation_angle, vertical_beam_width, horizontal_beam_width shall be present as well.
TSS-SSE-009	The software shall impute missing attributes with default values for different cell types. Default values are provided for 2 cell types: normal cells and micro cells. Default values for normal cells are: power = 10, antenna_height = 30, elevation_angle = 5, vertical_beam_width = 9, horizontal_beam_width = 65 Default values for micro cells are: power = 5, antenna_height = 8, elevation_angle = 5, vertical_beam_width = 9, horizontal_beam_width = 65
TSS-SSE-010	The software shall add additional attributes which are not part of I.8 Cell Locations with Physical Properties – Cleaned, but are required for signal strength modeling. These attributes are: range, path_loss_exponent, azimuth_signal_strength_back_loss, elevation_signal_strength_back_loss
TSS-SSE-011	The software shall add additional attributes with default values for different cell types. Default values are provided for 2 cell types: normal cells and micro cells. Default values for normal cells are: range = 10000, path_loss_exponent = 3.75, azimuth_signal_strength_back_loss = -30, elevation_signal_strength_back_loss = -30 Default values for micro cells are: range = 1000, path_loss_exponent = 6.0, azimuth_signal_strength_back_loss = -30, elevation_signal_strength_back_loss = -30
TSS-SSE-012	If the cell type is missing the software shall impute missing attributes and add necessary additional attributes using a single set of default values. The default values are: power = 5, antenna_height = 8, elevation_angle = 5, vertical_beam_width = 9, horizontal_beam_width = 65, range = 5000, path_loss_exponent = 3.75, azimuth_signal_strength_back_loss = -30, elevation_signal_strength_back_loss = -30
TSS-SSE-013	All default properties for a set of different cell types shall be provided in the configuration file.
TSS-SSE-014	The software shall convert cell antenna power parameters from watts to decibel milliwatts using formula: P(dBm) = 10 * log10(P(W)) + 30
TSS-SSE-013	The software shall create 3D point geometry of cells using latitude, longitude, and elevation plus the height of the antenna.
TSS-SSE-014	The software shall create 3D point geometry of grid centroids by adding elevation.
TSS-SSE-015	If elevation or height of the antenna is missing in the input data, the software shall set these attributes to 0.
TSS-SSE-016	The software shall perform a spatial join of cells with grid centroids based on cell range.
TSS-SSE-017	For signal strength propagation modeling, the software shall calculate planar and 3D cartesian distances between each cell and grid centroids within the cell radius. If both cells and grid centroids' elevations are 0, only planar distance is calculated.

ID	DEFINITION
TSS-SSE-018	The software shall calculate signal strength per grid tile based on the path loss exponent and power attributes of a cell and 3D distance from cell to grid tile. Formula for it is: Sg,a = S0 - Sdist(Rg,a), where S0 is P(dBm), Sdist is path_loss_exponent * 10 * log10(distance_to_cell_3D).
TSS-SSE-019	For directional cells, the software shall support the option for adjusting calculated previously signal strength values using formula Sg,a = S0 - Sdist(Rg,a) - Sazi(δg,a), where S0 - Sdist(Rg,a) is previously calculated signal strength values, Sazi(δg,a) is relation between signal loss and the offset azimuth angles between main direction of a cell and a grid tile. Sazi(δg,a) is calculated using linear transformation of the Gaussian formula: f(φ) = c - c * exp(-(φ^2) / (2 * σ^2)) where c and $\sigma 2$ are constants, whose value is determined by the cell's direction, horizontal beam width and the difference in signal strength between back and front of the cell (azimuth_signal_strength_back_loss). Whether to perform the adjustment or not shall be a configuration parameter.
TSS-SSE-020	For directional cells, the software shall support the option for adjusting signal strength values using formula Sg,a = S0 - Sdist(Rg,a) - Selev(εg,a), where S0 - Sdist(Rg,a) is previously calculated signal strength values, Selev(εg,a) is relation between signal loss and the offset elevation angles between tilt of a cell and a grid tile. Selev(εg,a) is calculated using linear transformation of the Gaussian formula: f(φ) = c - c * exp(-(φ^2) / (2 * σ^2)) where c and $\sigma 2$ are constants, whose value is determined by the elevation angle (tilt), vertical beam width and the difference in signal strength between back and front of the cell (elevation_signal_strength_back_loss). Whether to perform the adjustment step or not should be a configurable parameter.

3.2.4 CELLFOOTPRINTESTIMATION

ID	DEFINITION
TSS-CFE-001	The software shall read input data objects from parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-CFE-002	The input shall be I.12 Cells Signal Strengths Data Object (in Annex I).
TSS-CFE-003	The output shall be I.13 Cell Footprints Data Object (in Annex I).
TSS-CFE-004	The optional output shall be I.14 Cell Intersection Groups Data Object (in Annex I).
TSS-CFE-005	The software shall write output data objects into parquet format partitioned by year, month, day.
TSS-CFE-006	The software shall read input data for a date range based on the configuration parameter.
TSS-CFE-007	The software shall perform all processing steps for each date in the given date range independently.
TSS-CFE-008	The software shall produce a 'footprint' attribute of type float in the domain [0, 1] from 'signal strength' input.
TSS-CFE-009	Transformation of 'signal strength' to 'signal dominance' shall be performed using logistic equation: $(g, a) = 1 / (1 + \exp(-(S(g, a) - Smid)))$.
TSS-CFE-010	Parameters for signal strength transformation equation - Ssteep and Smid shall be defined in configuration file. Default values are Ssteep = 0.2, $Smid$ = -92.5.
TSS-CFE-011	The software shall have functionality to prune records by selecting cells for which the share to the total signal dominance of a grid tile is higher than the given threshold.
TSS-SFE-012	The software shall have functionality to prune records by selecting top X cell footprints per grid tile where X is a configurable parameter.
TSS-SFE-013	The software shall have functionality to prune records by selecting signal dominance values higher than given threshold.
TSS-SFE-014	Pruning steps are optional. Whether the step is performed or not shall be based on config parameters.
TSS-CFE-015	Parameters for defining top X cells per grid tile and threshold signal dominance values shall be defined in configuration file.
TSS-CFE-016	The software shall have optional step for calculating I.14 Cell Intersection Groups Data Object (in Annex I).
TSS-CFE-017	Cell Intersection Groups Calculation shall be performed with the following workflow: 1) Grouping per grid tile to aggregate all overlapping cells into sorted lists of cell groups; 2) Drop duplicated cell groups; 3) Generate all missing intersections. For example, for groups of 3 cells create all possible combinations of groups of 2 cells, for groups of 4 cells generate all possible combinations of groups of 2 cells and 3 cells and so on; 4) Drop duplicated cell groups again

3.2.5 CELLCONNECTIONPROBABILITYESTIMATION

ID	DEFINITION
TSS-CCPPPE-001	The software shall read input data objects from parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-CCPPPE-002	The software shall have one or two inputs: cell footprint values (mandatory) and land use prior probabilities (optional).
TSS-CCPPPE-003	The input schema for the cell footprint values input dataset shall be I.13 Cell Footprints. This input is mandatory.
TSS-CCPPPE-004	The input schema for the land use prior probabilities input dataset shall be I.11 Reference Grid. This input is optional.
TSS-CCPPPE-005	The software shall write output data objects to parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-CCPPPE-006	The software shall have one output: cell connection probability values.
TSS-CCPPPE-007	The output schema for cell connection probability values shall be I.15 Cell Connection and Posterior Probabilities.
TSS-CCPPPE-008	The software shall receive a configurable value for the validity period of input data (start and end dates).
TSS-CCPPPE-009	The software shall receive a configurable boolean value for deciding whether to use the land use prior probabilities input data in calculating the posterior_probability column.
TSS-CCPPPE-010	For each date in the validity period, for each grid tile, the software shall calculate the sum of cell signal dominances on that grid tile using the corresponding data from the cell footprint values input dataset.
TSS-CCPPPE-011	For each date, for each grid tile, for each cell, the software shall calculate normalized signal dominance. Normalized signal dominance is calculated by dividing the cell signal dominance value by the sum of signal dominance values of the same grid tile on the same date.
TSS-CCPPPE-012	For each date, for each grid tile, the normalized signal dominance values shall add up to a sum of 1.
TSS-CCPPPE-013	If using land use prior probabilities is enabled, for each date, for each grid tile, for each cell, the software shall calculate the posterior probability. Posterior probability is calculated by multiplying the normalized signal dominance value by the prior probability value of the same grid tile in the land use prior probabilities input dataset.
TSS-CCPPPE-014	If using land use prior probabilities is not enabled, the posterior probability value is equal to the normalized signal dominance value.
TSS-CCPPPE-015	For each date, for each grid tile, for each cell, the software shall calculate the cell connection probability. The cell connection probability is calculated by dividing the posterior probability value by the sum of posterior probability values of the same cell on the same date.
TSS-CCPPPE-016	For each date, for each cell, the normalized cell connection probability values shall add up to a sum of 1.
TSS-CCPPPE-017	For each date, for each grid tile, for each cell, the output shall contain both the normalized signal dominance and the normalized cell connection probability.

3.2.6 EVENTCLEANING

ID	DEFINITION
TSS-EVN-001	The software shall read event input data from parquet files stored under a folder structure with the format year=YYYY/month=MM/day=DD/user_id_modulo=x where YYYY represents the year, MM the month, DD the day for the event, x the user id modulo value and the '/' symbol denotes different folders.
TSS-EVN-002	The software shall be able to write the syntactically cleaned event output data in parquet format, partitioned by year (YYYY), month (MM) and day (DD), and user_id_modulo in the schema defined I.2 MNO Event Data – Syntactically Cleaned.
TSS-EVN-003	The software shall be able to write the quality metrics frequency distribution in parquet format, partitioned by date, in the schema defined in I.4 MNO Event Data Syntactic Quality Metrics – frequency distribution.
TSS-EVN-004	The software shall be able to write the quality metrics by column in parquet format, partitioned by date, in the schema defined in I.3 MNO Event Data Syntactic Quality Metrics – by column.
TSS-EVN-005	The software shall check that all the mandatory columns specified in I.1 MNO Event Data – Raw data object exist in the input data.
TSS-EVN-006	If the input data is missing an optional column, the software shall create the optional column with all its values set to the null value.
TSS-EVN-007	The software shall create a year column of the pyspark datatype ShortType from the timestamp data.
TSS-EVN-008	The software shall create a month column of the pyspark datatype ByteType from the timestamp data.
TSS-EVN-009	The software shall create a day column of the pyspark datatype ByteType from the timestamp data.
TSS-EVN-010	The software shall sort the output data to be written by the user_id and timestamp column.
TSS-EVN-011	The software shall write output data in parquet format partitioned by year, month, day and user_id_modulo.
TSS-EVN-012	The software shall infer the domain of the data following the logic: • If the plmn value is not null, the domain is outbound. • If the mcc value is equal to local_mcc (defined in the configuration), the domain is domestic. • Otherwise, the domain is inbound.
TSS-EVN-013	The software shall discard domestic and inbound records which do not meet at least one of the following conditions: • the cell_id value is valid; • latitude and longitude values are valid.
TSS-EVN-014	The software shall discard records in which any mandatory field doesn't comply with the field requirements specified in the I.1 MNO Event Data – Raw data object.
TSS-EVN-015	The software shall be able to receive by configuration a *bounding_box* value composed of four decimal numbers that define a square within WGS84 bounds.
TSS-EVN-016	The software shall discard records in which the user_id is not a binary data type of 32 bytes.
TSS-EVN-017	The software shall discard domestic and inbound records which do not meet both of the following conditions: • the mcc value is a 3-digit code; • the mnc is a 2- or 3-digit code (can also begin with 0).
TSS-EVN-018	The software shall discard outbound records where the plmn value is not a 5 or 6 digit number.
TSS-EVN-019	The software shall discard domestic and inbound records if no latitude and longitude values are given and the cell_id does not follow CGI and eCGI standards.
TSS-EVN-020	The software shall discard domestic and inbound records if no cell_id value is given and the latitude or longitude values are not within WGS84 bounds and the *bounding_box* if given by configuration.
TSS-EVN-021	The software shall receive timestamp data in UTC format.
TSS-EVN-022	The software shall extract the first 12 characters of the user id hash, convert it into integer of base 10, and apply the modulo function on that result to calculate the user_id_modulo.
TSS-EVN-023	The software shall discard records with identical timestamps and identical location information for every user, i.e. it shall remove same location duplicates from the data. Two rows have an identical location information when user_id, timestamp, cell_id, longitude and latitude and plmn columns are identical.

3.2.7 EVENTQUALITYWARNINGS

ID	DEFINITION
TSS-EVN-QW-001	The software shall be able to perform Quality Warnings checks after MNO Event Cleaning - Syntactic Checks.
TSS-EVN-QW-002	The software shall be able to read and process configs of Event Cleaning Quality Warnings.
TSS-EVN-QW-003	The software shall read Quality Metrics of MNO Event Cleaning in parquet format stored under a folder structure with the format date=YYYY-mm-dd. The Quality Metrics include Frequency Distribution and By Column Data Objects, with schema specified in I.4 MNO Event Data Syntactic Quality Metrics – frequency distribution and I.3 MNO Event Data Syntactic Quality Metrics – by column.
TSS-EVN-QW-004	The output of the component shall be a Log Table and ForPlots Data Objects following the data type scheme specified in I.5 MNO Event Data Quality Warnings – log table and I.22 MNO Event Data Quality Warnings – for plots.
TSS-EVN-QW-005	The software shall perform Quality Warnings based on Quality Metrics data. Given that data_period_startand data_period_end define the time boundaries of Event Quality Warnings, the period of available data of Quality Metrics should span over [data_period_start - lookback_period, data_period_end], since the intermediate results for Quality Warnings are calculated based on previous data.
TSS-EVN-QW-006	The software shall write Quality Warnings Log Table and ForPlots data into parquet format partitioned by date column, the storing period of Log Table should be [data_period_start , data_period_end], ForPlots - [data_period_start - lookback_period, data_period_end].
TSS-EVN-QW-007	The software shall compute Quality Warnings and store results in Log Table Data Object regarding data size (initial and final frequency), which includes checking if a size within a range of two absolute numbers (upper and lower limit) and between [mean+X*std, mean-X*std] boundaries (calculated based on lookback data). Applicable only for Event Cleaning Quality Warnings.
TSS-EVN-QW-008	The software shall compute Quality Warnings and store results in Log Table Data Object regarding error rate (formula = (Total initial frequency - Total final frequency) / Total initial frequency*100))on four granularity levels: by date, by date and cell_id, by date and user_id, by date and cell_id and user_id. The error rate is checked on three warnings: should not be higher than some absolute number; should not be higher than average of previous error rates by some X%, should not be higher than mean + X*std (average and std are calculated on lookback data). Applicable only for Event Cleaning Quality Warnings.
TSS-EVN-QW-009	The software shall compute Quality Warnings and store results in Log Table Data Object regarding error type rate for each specified error rate&field name combination (formula = number of errors of error type&field name combination / Total initial frequency *100). The checks on error type rate contains three warnings: it must not exceed a specific absolute number; it should not surpass the average of prior error type rates by a certain percentage X; and it cannot be greater than the mean plus X times the standard deviation (where both average and standard deviation are determined using historical data). Applicable for Event Cleaning Quality Warnings.
TSS-EVN-QW-010	The software shall store data in ForPlots Data Object to plot three variables' distribution - initial frequency, final frequency, and error rate by date along with their corresponding mean, mean+X*std (UCL - upper control limit), mean-X*std values (LCL - lower control limit), computed on lookback period. Applicable only for Event Cleaning Quality Warnings.
TSS-EVN-QW-011	The software's orchestration shall provide the option to flexibility define what group of Quality Warnings to compute and what values for different thresholds to choose.

3.2.8 EVENTDEDUPLICATION

ID	DEFINITION
TSS-EVNDED-001	The software shall perform removal of same and different location duplicates and calculate corresponding quality metrics.
TSS-EVNDED-002	The software shall read and process configs of EventDeduplication module.
TSS-EVNDED-003	The input event data schema shall be I.2 MNO Event Data – Syntactically Cleaned.
TSS-EVNDED-004	The software shall calculate and write three quality metrics: (i) *same location deduplication discarded row count*, (ii) *different location deduplication discarded row count*, (iii) *record frequency distribution.*
TSS-EVNDED-005	The output schema for deduplicated event data shall be I.6 MNO Event Data – Deduplicated.
TSS-EVNDED-006	The software shall distinguish between two types of duplication errors: same location duplicates and different location duplicates. In case of same location duplicates, only one row is kept. In case of different location duplicates, all rows are dropped. All rows that remain are considered deduplicated event data.
TSS-EVNDED-007	The software shall apply same location deduplication before different location deduplication.
TSS-EVNDED-008	The software shall identify as a group of same location duplicates any two or more event records that meet each of the following conditions: (i) identical user id, (ii) identical timestamp, (iii) identical location (either (iiia) identical cell id or (iiib) identical longitude-latitude pair).
TSS-EVNDED-009	The software shall for each group of same location duplicates keep one record and discard all other records.
TSS-EVNDED-010	The software shall calculate the *same location deduplication discarded row count quality metric* as the total number of records discarded by same location duplicate removal for each date.
TSS-EVNDED-011	The output schema for the *same location deduplication discarded row count quality metric* shall be I.3 MNO Event Data Syntactic Quality Metrics – by column, using the error code reserved for same location duplicate errors.
TSS-EVNDED-012	The software shall identify as a group of different location duplicates any two or more event records that meet each of the following conditions: (i) identical user id, (ii) identical timestamp, (iii) non-identical location (either (iiia) non-identical cell id or (iiib) non-identical longitude-latitude pair).
TSS-EVNDED-013	The software shall for each group of different location duplicates discard all the records.
TSS-EVNDED-014	The software shall calculate the *different location deduplication discarded row count quality metric* as the total number of records discarded by different location duplicate removal for each date.
TSS-EVNDED-015	The output schema for the *different location deduplication discarded row count quality metric* shall be I.3 MNO Event Data Syntactic Quality Metrics – by column, using the error code reserved for different location duplicate errors.
TSS-EVNDED-016	The software shall calculate the *record frequency distribution quality metric* as the number of records per user per cell before deduplication and after deduplication for each date.
TSS-EVNDED-017	The output schema for the *record frequency distribution quality metric* shall be I.4 MNO Event Data Syntactic Quality Metrics – frequency distribution.

3.2.9 SEMANTICCLEANING

ID	DEFINITION
TSS-ESC-001	The software shall be able to read deduplicated event input data from parquet files stored partitioned by year (YYYY), month (MM), day (DD), and user_id_modulo.
TSS-ESC-002	The software shall be able to read clean network topology input data from parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-ESC-003	The software shall be able to read syntactically clean event input data with the data type scheme specified in I.6 MNO Event Data – Deduplicated Cleaned.
TSS-ESC-004	The software shall be able to read clean network topology input data with the data type scheme specified in I.8 Cell Locations with Physical Properties – Cleaned.
TSS-ESC-005	The software shall be able to write the semantically clean event output data in parquet format, partitioned by year (YYYY), month (MM), day (DD), and user_id_modulo.
TSS-ESC-006	The software shall be able to write the semantic quality metrics output data in parquet format, partitioned by year (YYYY), month (MM), and day (DD).
TSS-ESC-007	The software shall be able to write the output semantically cleaned event data at device level following the data scheme specified in I.16 MNO Event Data – Semantically Cleaned.
TSS-ESC-008	The software shall be able to write the output device semantic quality metrics following the data scheme specified in I.17 MNO Device Semantic Quality Metrics.
TSS-ESC-009	The software shall be able to flag with the corresponding error code those event registers which make a reference to a cell ID that does not exist in the input network topology data for the date in which that event was registered.
TSS-ESC-010	The software shall be able to flag with the corresponding error code those event registers which make a reference to a cell ID that does in the input network topology data for the date in which that event was registered, but is not operational at the moment the event was registered.
TSS-ESC-011	The software shall be able to flag with the error code corresponding to certain incorrect location an event register where the estimated distance and speed between this event and both the previous and following event are above some distance and speed thresholds specified via configuration.
TSS-ESC-012	The software shall be able to flag with the error code corresponding to suspicious incorrect location an event register where the estimated distance and speed between this event and either the previous or the following event, but not both at the same time, are above some distance and speed thresholds specified via configuration.
TSS-ESC-013	The software shall be able to flag with the error code corresponding to suspicious incorrect location the first event of a user if the estimated distance and speed with the second event register are above some distance and speed thresholds specified via configuration.
TSS-ESC-014	The software shall be able to flag with the error code corresponding to different location duplicate.
TSS-ESC-015	The software shall not delete any event registers when they are flagged.
TSS-ESC-016	The software shall be able to count, for a given day, the number of events flagged with each error code, as well as non-flagged events.
TSS-ESC-017	The software shall be able to record each quality metric together with the timestamp of the moment when the component was executed.
TSS-ESC-018	The software shall be able to record each quality metric together with the date to which it refers.
TSS-ESC-019	The software shall be able to read from a configuration file the minimum distance threshold, in meters, above which an event might be flagged with a location related error code.
TSS-ESC-020	The software shall be able to flag with the corresponding error code those event registers which are different location duplicates. These are rows which have identical timestamp values for a given user, but non-identical values in any other columns.

3.2.10 SEMANTICQUALITYWARNINGS

ID	DEFINITION
TSS-ESW-001	The software shall be able to read semantic quality metrics input data from parquet files stored partitioned by year (YYYY), month (MM) and day (DD).
TSS-ESW-002	The software shall be able to read semantic quality metrics input data with the data type scheme specified in I.17 MNO Device Semantic Quality Metrics.
TSS-ESW-003	The software shall be able to write bar plot data in parquet format, partitioned by variable, year (YYYY), month (MM), day (DD), and execution timestamp.
TSS-ESW-004	The software shall be able to write the output quality warnings log table following the data type scheme specified in I.18 MNO Event Data at Device Level Semantic Quality Warnings – log table.
TSS-ESW-005	The software shall be able to write the output bar plot data following the data type scheme specified in I.25 Event Data at Device Level Semantic Quality Warnings Bar Plot Data.
TSS-ESW-006	The software shall be able to calculate the percentage with which each error type occurs, including the “no error” type, defined as the fraction of the number of said error type over the total number of events for a given date.
TSS-ESW-007	The software shall be able to calculate, for each error type, the sample standard deviation of the percentage of each error type over a lookback period specified via configuration file for each particular error type.
TSS-ESW-008	The software shall be able to raise a warning, in the case that all lookback period dates are present, when the percentage of a given error type is greater than the average percentage over its lookback period by more than a given number of standard deviations
TSS-ESW-009	The software shall be able to raise a warning, in the case that all lookback period dates are present but are strictly lower than 3, when the percentage of a given error type is greater than a given absolute threshold.
TSS-ESW-010	The software shall not raise a warning for a given error type when any of the dates in its lookback period is missing.
TSS-ESW-011	The software shall record the percentage of each error type (excluding the “no error type”).
TSS-ESW-012	The software shall record the threshold computed for a given error type from the average and sample standard deviations of its lookback period whenever all dates in its lookback period are present.
TSS-ESW-013	The software shall be able to write into a parquet file the necessary data to create a bar plot with the dates in its horizontal axis, ranging from the furthest lookback period in the past to the study date, and the absolute count of each error type, including the ”no error” type, in the vertical axis, for the dates in which this data exists.
TSS-ESW-014	The software shall be able to write into a parquet file the necessary data to create a bar plot with the dates in its horizontal axis, ranging from the furthest lookback period in the past to the study date, and the percentage of each error type, including the ”no error” type, in the vertical axis, for the date in which this data exists.
TSS-ESW-015	The software shall be able to read from a configuration file the lookback period to be considered, measured in days, one per metric.
TSS-ESW-016	The software shall be able to read from a configuration file the number of standard deviations to be considered to compute the thresholds, one per metric.
TSS-ESW-017	The software shall be able to read from a configuration file the percentage threshold to use as threshold when the standard deviation cannot be computed, one per metric.
TSS-ESW-018	The software shall contain default values for every threshold and lookback period, to be used in case they are not specified via configuration file.

3.2.11 DAILYPERMANENCESCORE

ID	DEFINITION
TSS-DPS-001	The software shall read semantically cleaned event input data from parquet files stored partitioned by year (YYYY), month (MM), day (DD) and user_id_modulo.
TSS-DPS-002	The software shall be able to read semantically cleaned event input data with the data type scheme specified in I.16 MNO Event Data – Semantically Cleaned.
TSS-DPS-003	The software shall read cell footprint input data from parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-DPS-004	The software shall be able to read cell footprint input data with the data type scheme specified in I.13 Cell Footprints.
TSS-DPS-005	The software shall be able to write daily permanence score data to parquet files stored partitioned by year (YYYY), month (MM), day (DD), user_id_modulo and id_type.
TSS-DPS-006	The software shall write output daily permanence score data following the data type scheme specified in I.21 Daily Permanence Score.
TSS-DPS-007	The software shall load parameter values from a configuration file.
TSS-DPS-008	The software shall load all events of the user within the analysed date, also including the last event preceding this date and the first event following it.
TSS-DPS-009	The software shall find, for each user and analysed date, each group of 3 events.
TSS-DPS-010	The software shall calculate the minimum Euclidean distance between the cell footprints of the event sequence. E.g.: d(A,B), d(B,C), d(A,C).
TSS-DPS-011	The software shall calculate the sum of the distance from the first cell of the 3-cell event sequence (A) to the second cell (B) plus the distance from this "intermediate" cell (B) to the last cell of the event sequence (C). E.g.: d(A,B,C) = d(A,B) + d(B,C)
TSS-DPS-012	The software shall select the maximum distance from d(A,C) and d(A,B,C). E.g.: d = max(d(A,C),d(A,B,C))
TSS-DPS-013	The software shall calculate the time difference between the first and the last event of each event sequence. E.g.: Δt = t(C) - t(A)
TSS-DPS-014	The software shall calculate the speed resulting from dividing the maximum distance by the time difference between the first and last events of the sequence. s = d / Δt
TSS-DPS-015	The software shall tag all the intermediate events between the first and last events of the sequence as "move" events if the maximum speed (s) is higher than a specified threshold (e.g. 50 km/h).
TSS-DPS-016	The software shall generate a preliminary initial timestamp and a preliminary final timestamp for each permanence-associated event (events which have not been tagged as "move"). The preliminary initial timestamp shall be equal to the average time point between the previous event of the user and this event, and the preliminary final timestamp shall be equal to the average time point between this event and the following. If no previous event or no posterior events are available, a standard predefined displacement (max_time_thresh/2) shall be applied to the event time in order to obtain the initial/final timestamp (e.g. 15 min).
TSS-DPS-017	The software shall modify the preliminary initial timestamp when the previous event happens in the same cell as the current event if the time between the previous event and the current event is higher than a specified threshold. This threshold shall be configurable for day (max_time_thresh_day for 9:00-22:59) and night (max_time_thresh_night for 23:00-8:59) periods (e.g. 2h from 9:00h to 22:59h and 8h from 23:00 to 8:59h). In case the time difference is higher than the threshold, a standard predefined displacement (max_time_thresh/2) shall be applied to the current event time in order to obtain its initial timestamp (e.g. 15 min).
TSS-DPS-018	The software shall modify the preliminary final timestamp when the posterior event happens in the same cell as the current event if the time between the posterior event and the current event is higher than a specified threshold. This threshold shall be configurable for day (max_time_thresh_day for 9:00-22:59) and night (max_time_thresh_night for 23:00-8:59) periods (e.g. 2h from 9:00h to 22:59h and 8h from

ID	DEFINITION
	23:00 to 8:59h). In case the time difference is higher than the theshold, a standard predefined displacement (max_time_thresh/2) shall be applied to the current event time in order to obtain its final timestamp (e.g. 15 min).
TSS-DPS-019	The software shall modify the preliminary initial timestamp when the previous event happens in a different cell from the current event cell if the time between the previous event and the current event is higher than a specified threshold (max_time_thresh). In case the time difference is higher than the threshold, a standard predefined displacement (max_time_thresh/2) shall be applied to the current event time in order to obtain its initial timestamp (e.g. 15 min).
TSS-DPS-020	The software shall modify the preliminary final timestamp when the posterior event happens in a different cell from the current event cell if the time between the posterior event and the current event is higher than a specified threshold (max_time_thresh). In case the time difference is higher than the threshold, a standard predefined displacement (max_time_thresh/2) shall be applied to the current event time in order to obtain its final timestamp (e.g. 15 min).
TSS-DPS-021	The software shall split the day into N time slots of equal length, where N is the "time_slot_number" parameter loaded from the configuration file.
TSS-DPS-022	The software shall intersect each of the N time slots with the permanence events of the user in the analysed date, and thus obtain, for each time slot, the cells in which the user presents permanence and for how long within the time slot.
TSS-DPS-023	The software shall compute for each user and time slot the total time in seconds the user has not presented permanence in any cell.
TSS-DPS-024	The software shall convert from cell to tile by using the cell footprint data, obtaining the information of how long the user presents permanence in each tile in each time slot.
TSS-DPS-025	The software shall map the total time in seconds in a time slot that a user has not presented permanence in any cell to a virtual grid_id under the name ‘unknown’.
TSS-DPS-026	The software shall record for each location in the ‘id_type’ field a value of ‘grid’ whenever the ‘grid_id’ field contains the ID of a grid tile, or ‘unknown’ when the ‘grid_id’ field contains the ‘unknown’ value.
TSS-DPS-027	The software shall generate a daily permanence score (DPS) for each time slot and tile combination according to the time that the user presents permanence in that tile during the time slot *t*, and the duration of that timeslot, *T*. The DPS shall be an integer between 0 and 1, following this rule: • If the user presents permanence in a tile for 0 < t < T/2 during the time slot, then DPS = 0 for that user, tile and time slot. • If the user presents permanence in a tile for T/2 <= t <= T during the time slot, then DPS = 1 for that user, tile and time slot.

3.2.12 CONTINUOUSTIMESEGMENTATION

ID	DEFINITION
TSS-CTS-001	The software shall read input data from parquet files stored under a folder structure with the format year=YYYY/month=MM/day=DD.
TSS-CTS-002	The input data shall be I.16 MNO Event Data – Semantically Cleaned and I.14 Cell Intersection Groups Data Objects.
TSS-CTS-003	The optional input data is previously calculated I.20 Daily Continuous Time Segments for the date before current processing date.
TSS-CTS-004	The output data shall be I.20 Daily Continuous Time Segments Data Object.
TSS-CTS-005	The software shall write output data in parquet format partitioned by year, month, day.
TSS-CTS-006	The software shall read input data for a date range based on the configuration parameter.
TSS-CTS-007	The software shall validate that events data for a given date range is available.
TSS-CTS-008	The software shall perform all processing steps for each date in the given date range independently.
TSS-CTS-009	The software shall be able to do Time Segments aggregation either from scratch or starting from the last created Time Segments. This shall be defined by configuration parameter.
TSS-CTS-010	For each processing date, the software shall get MNO Events and Cell Intersection Groups data for the current date and next date (D, D+1).
TSS-CTS-011	If available, the software shall additionally get the last Time Segment for each user for the previous date (D-1).
TSS-CTS-012	For each user’s daily events data, the software shall aggregate events into Continuous Time Segments based on time and location difference between consecutive events. This operation shall be performed by iteration over events and comparisons of previously created Time Segments timestamps and cells with current event timestamp and cell.
TSS-CTS-013	The first Time Segment of a day shall start at 00:00:00, the last time segment of a day shall end at 23:59:59.
TSS-CTS-014	The software shall be able to create Time Segments for every device and every day since the device has appeared in data.
TSS-CTS-015	The software shall be able to create the following Time Segments types: ‘unknown', ‘undetermined’, ‘stay’, 'move’.
TSS-CTS-016	The 'unknown' Time Segment shall be created if one of following conditions are met: 1) there are no events for the device for this date is available; 2) cells of the previous Time Segment and cell of the current event are belong to the same cell intersection group and the difference between the previous Time Segment end timestamp and current event timestamp is more than 12 hours (configuration parameter); 3) cells of the previous Time Segment and cell of the current event do not belong to the same cell intersection group and the difference between the previous Time Segment end timestamp and current event timestamp is more than 2 hours (configuration parameter).
TSS-CTS-017	The ‘undetermined' Time Segment shall be created either if the previous Time Segment is of type ‘unknown’ or if the previous Time Segment is of type ‘move’ and the difference between its end timestamp and current event timestamp is less than 15 minutes (config parameter).
TSS-CTS-018	The ‘stay' Time Segment shall be created if the previous Time Segment is 'undetermined’, the difference between its end timestamp and current event timestamp is more than 15 minutes (config parameter) and its cells and cell of the current event belong to the same cell intersection group.
TSS-CTS-019	The ‘stay’ Time Segment shall be extended if the difference between its end timestamp and current event timestamp is less than 12 hours (config parameter) and its cells and cell of the current event belong to the same cell intersection group.
TSS-CTS-020	The ‘move' Time Segment shall be created if the previous Time Segment is ‘undetermined’, ’stay' or 'move', the difference between its end timestamp and current event timestamp is less than 2 hours (config parameter) and its cells and cell of the current event do not belong to the same cell intersection group.

3.2.13 INSPIREGRIDGENERATOR

ID	DEFINITION
TSS-GG-001	The software shall be able to generate Spatial Reference Grid following INSPIRE Specification.
TSS-GG-002	The software shall have possibility to generate Spatial Reference Grid for extent given in WGS84 coordinate system.
TSS-GG-003	The software shall have possibility to generate Spatial Reference Grid for a country polygon given in WGS84 coordinate system.
TSS-GG-004	The software shall be able to read countries dataset input data with the data type and schema specified in I.29 Countries.
TSS-GG-005	Spatial context option (extent or country) for grid generation shall be defined in configuration file.
TSS-GG-006	If spatial context option is extent, extent has to be provided as a parameter. If spatial context option is country, country iso2 code has to be set in config.
TSS-GG-007	The software shall be able to convert spatial zones dataset coordinate system into internal coordinate system (EPSG: 3035) before grid generation.
TSS-GG-008	The software shall be able to extend country polygon to fixed buffer distance before grid generation. Buffer distance shall be provided in config file.
TSS-GG-009	The software shall write grid data object following schema specified in I.28 INSPIRE Grid.
TSS-GG-010	The software shall be able to partition output grid data object by quadkey.
TSS-GG-011	The software shall be able to assign quadkey to each grid tile.
TSS-GG-012	Quadkey level for partitioning shall be provided in configuration file.

3.2.14 SYNTHETICDIARIES

ID	DEFINITION
TSS-SYN-DI-001	The software shall load all necessary parameters from a configuration file.
TSS-SYN-DI-002	The output of the component shall be I.30 Synthetic Diaries parquet files partitioned by year, month and date.
TSS-SYN-DI-003	The software shall generate N activity trip diaries per specified date, where N is a parameter provided through the configuration file.
TSS-SYN-DI-004	The software shall generate, for each user, an activity trip diary that contains some of the stays specified in the ‘stay_sequence_superset’ by probabilistically generating (or not) each of the stays according to the specified ‘stay_sequence_probabilities’.
TSS-SYN-DI-005	The software shall generate, for each user and date, a home location that is within the bounding box provided through the parameters ‘longitude_min’, ‘longitude_max’, ‘latitude_min’ and ‘latitude_max’.
TSS-SYN-DI-006	The software shall generate, for each user and date, a work location that is within the bounding box provided through the parameters ‘longitude_min’, ‘longitude_max’, ‘latitude_min’ and ‘latitude_max’, and which is at a distance between ‘home_work_distance_min’ and ‘home_work_distance_max’ of the home location of the user.
TSS-SYN-DI-007	The software shall locate every user stay that is not of type ‘home’ or ‘work’ to a location which is at a distance of between ‘other_distance_min’ and ‘other_distance_max’ from the location of the previous stay, and which is within the bounding box.
TSS-SYN-DI-008	The software shall assign a duration for each trip (interval between stays), which is equal to the distance between the locations of the stays divided by the specified standard ‘displacement distance’.
TSS-SYN-DI-009	The software shall assign a duration for each stay, which shall be compatible with the ‘duration min’ and ‘duration max’ parameters corresponding to the stay type.
TSS-SYN-DI-010	The stays and trips of each diary shall cover the whole day, from 00:00:00 to 23:59:59.

3.2.15 SYNTHETICNETWORK

ID	DEFINITION
TSS-SN-001	The software shall be able to write synthetic generated network topology data to parquet files stored partitioned by year (YYYY), month (MM, and day (DD).
TSS-SN-002	The software shall be able to write synthetic generated network topology data following the schema specified in I.7 Cell Locations with Physical Properties - Raw.
TSS-SN-003	The software shall be able to read from a configuration file the start and end date of the range of dates for which data will be generated.
TSS-SN-004	The software shall be able to read from a configuration file the number of cells to be generated.
TSS-SN-005	The software shall be able to read from a configuration file a seed value that will be applied to all the random processes within the component.
TSS-SN-006	The software shall be able to read from a configuration file the latitudes and longitudes defining a bounding box in which cell coordinates will be generated.
TSS-SN-007	The software shall be able to read from a configuration file the minimum and maximum value for the values of the altitude, power, range, and frequency fields.
TSS-SN-008	The software shall be able to read from a configuration file the maximum, positive value that the antenna height field can take.
TSS-SN-009	The software shall be able to read from a configuration file the value that will be set in the valid date start and valid date end fields of all cells.
TSS-SN-010	The software shall be able to read from a configuration file the list of values that the field cell_type can take.
TSS-SN-011	The software shall be able to read from a configuration file the probabilities of not generating any optional fields, of setting null values in mandatory fields, of generating values outside of the allowed ranges, and of creating erroneous values in the cell_id, valid_date_start and valid_date_end fields.
TSS-SN-012	The software shall be able to create physical network topology data for the number of cells and the range of dates specified via configuration.
TSS-SN-013	The software shall be able to create records where all the non-mandatory columns of I.7 Cell Locations with Physical Properties - Raw have a non-null value with a probability specified via configuration.
TSS-SN-014	The software shall be able to create records where mandatory fields have a null value with a probability specified via configuration.
TSS-SN-015	The software shall be able to create records where fields take values outside of the allowed ranges with a probability specified via configuration.
TSS-SN-016	The software shall be able to create records with erroneous values in the cell_id, valid_date_start and valid_date_end fields with a probability specified via configuration.

3.2.16 SYNTHETICEVENTS

ID	DEFINITION
TSS-EVN-QW-001	The software shall be able to read and process the data objects of I.7 Cell Locations with Physical Properties - Raw and I.30 Synthetic Diaries.
TSS-EVN-QW-002	The output of the component shall be I.1 MNO Event Data – Raw partitioned by year, month and date.
TSS-EVN-QW-003	The software shall generate timestamp, latitude and longitude values for moves (move events without cell_ids) based on Synthetic Diaries, taking into account the following: 1) The total amount of events generated on the line between the current stay point and next stay point (as provided in synthetic diaries), shall be equal to the configuration parameterevent_freq_moves. 2) The time differences between event timestamps for a given move event and user shall be randomly distributed. 3) The generated points shall be randomly distributed on the line from the current stay point and next stay point.
TSS-EVN-QW-004	The software shall perform all spatial calculation operations using the coordinate reference system in the configuration parameter cartesian_crs.
TSS-EVN-QW-005	The software shall generate timestamp, latitude and longitude values for stays (stay events without cell_ids) based on Synthetic Diaries, taking into account the following: 1) The total amount of events generated for the stay location (a single point) between the period of initial and final timestamp of a stay, shall be equal to event_freq_moves. 2) The time differences between event timestamps for a given stay event and user shall be randomly distributed.
TSS-EVN-QW-006	The software shall generate timestamp, latitude and longitude values for stays (stay events without cell_ids) based on Synthetic Diaries, taking into account the following: 1) The total amount of events generated for the stay location (a single point) between the period of initial and final timestamp of a stay, shall be equal to event_freq_moves. 2) The time differences between event timestamps for a given stay event and user shall be randomly distributed.
TSS-EVN-QW-007	From the generated stays and move events, the software shall randomly select, using as seed the configuration parameter seed, the ratio of rows equal to error_location_probability for generating location errors. It shall modify the longitude and latitude values of these rows, taking into account the following: 1) The minimum distance from the existing point to the newly generated erroneous point shall not be below the configuration parameter error_location_distance_min and the maximum distance shall not exceed the configuration parameter error_location_distance_max. 2) The software shall support generating error values in all directions: north, east, south and west of a point. 3) The software shall store the distance from the newly generated point and existing point in the column loc_error.
TSS-EVN-QW-008	From the generated stays and move events, the software shall randomly select, using as seed the configuration parameter seed, the ratio of rows equal to error_cell_id_probability for the generating errors in the cell_id column. It shall create a cell_id column for the selection of these rows, considering the following: 1) The generated cell_id values shall syntactically follow the format of the cell_id column. 2) The generated cell_id shall be such that no cell_id value in the Synthetic Network data matches it.
TSS-EVN-QW-009	The software shall calculate the closest cell_id for each generated event, based on the latitude and longitude values, taking into account the following: 1) For every event, only cells within the distance defined by configuration parameter closest_cell_distance_max are considered. 2) For every event, the cell_id is selected randomly but only as many cells are considered, as defined by the configuration parameter max_n_of_cells.

ID	DEFINITION
TSS-EVN-QW-010	The software shall set the value of mcc and mnc as a single value for all users as defined by configuration parameters mcc and mnc respectively.
TSS-EVN-QW-011	The software shall set the value of plmn as null for all users.

3.2.17 PRESENTPOPULATIONESTIMATION

ID	DEFINITION
TSS-PPE-001	The software shall be able to read semantically cleaned event input data from parquet files stored partitioned by year (YYYY), month (MM), day (DD), and user_id_modulo.
TSS-PPE-002	The software shall be able to read cell connection probabilities input data from parquet files stored partitioned by year (YYYY), month (MM), and day (DD).
TSS-PPE-003	The software shall be able to read clean semantically cleaned event input data with the data type scheme specified in I.16 MNO Event Data – Semantically Cleaned.
TSS-PPE-004	The software shall be able to read cell connection probabilities input data with the data type scheme specified in I.15 Cell Connection and Posterior Probabilities.
TSS-PPE-005	The software shall be able to perform computations based on the 100m x 100m INSPIRE grid with the data type scheme specified in I.28 INSPIRE Grid.
TSS-PPE-006	The software shall be able to read, if so specified via configuration file, the mapping of grid tiles to zones data with the data type scheme specified in I.36 Zones – Grid Map.
TSS-PPE-007	The software shall be able to write the estimated present population output data in parquet format, partitioned by year (YYYY), month (MM), day (DD).
TSS-PPE-008	The software shall be able to write the output estimated present population data following the data scheme specified in I.42 Present Population (at grid level) if so specified via configuration file.
TSS-PPE-009	The software shall be able to write the output estimated present population data following the data scheme specified I.42 Present Population (at the level of the zoning system) if so specified via configuration file.
TSS-PPE-010	The software shall be able to read from a configuration file whether the present population estimation will be written at the grid system level or at the zoning level; and in the latter case, be able to read a dataset ID and, if applicable, a hierarchical level (starting from 0, 1, 2, …).
TSS-PPE-011	The software shall be able to read from a configuration file the timestamp for which the present population is to be estimated.
TSS-PPE-012	The software shall be able to read from a configuration file the time gap, in seconds, such that all devices with a register with a time difference to the timestamp no larger than this gap will be included in the estimation of the present population.
TSS-PPE-013	The software shall be able to read from a configuration file the tolerance threshold for the sum of absolute differences between the spatial distribution of devices of the current iteration and the previous iteration over all grid tiles, to stop the iterative procedure of this component.
TSS-PPE-014	The software shall be able to read from a configuration file the maximum number of iterations that the estimation procedure will be allowed to run for.
TSS-PPE-015	The software shall be able to read all registers that are within a distance of the timestamp no larger than the time gap specified via configuration, including registers of the previous or following day if the time gap crosses over midnight.
TSS-PPE-016	The software shall filter out all registers that have a time difference larger than the time gap with respect to the timestamp.
TSS-PPE-017	The software shall consider the remaining devices together with the cell they connected to according to their register closest to the timestamp. In the case that two events are equally close to the timestamp, the earlier event is selected.
TSS-PPE-018	The software shall count, for each cell, the number of devices that have connected to that cell in their register closest to the timestamp.
TSS-PPE-019	The software shall estimate the spatial distribution of devices over the grid tiles that cover the country and its buffer through an iterative Bayesian procedure.
TSS-PPE-020	The software shall use a uniform prior distribution of the devices over the grid tiles equal to the total number of devices divided by the total number of grid tiles of the distribution as the initial value of the spatial distribution of devices.
TSS-PPE-021	The software shall, in each iteration, compute the posterior probability of a device being in grid tile *j* when connected to cell *i* according to Bayes' theorem, multiplying the value of the spatial distribution for tile *j* found in the previous iteration (the prior) by the known cell connection probability of connecting to cell *i* when being in tile *j*, and then normalising over all grid tiles.

ID	DEFINITION
TSS-PPE-022	The software shall, in each iteration, compute the new spatial density of devices for a given grid tile j as the sum, over all cells i, of the product of devices connected to cell i and the posterior probability of being in grid tile j when connected to cell i.
TSS-PPE-023	The software shall stop the iterative estimation if the number of iterations exceeds the maximum number of iterations specified via the configuration file.
TSS-PPE-024	The software shall stop the iterative estimation if the sum of absolute differences between the spatial distribution of devices of the current iteration and the previous over all grid tiles is less than the tolerance threshold specified via configuration file.
TSS-PPE-025	The software shall take the spatial distribution of devices over the grid tiles obtained in the iterative procedure as the estimation of the present population.
TSS-PPE-026	The software shall be able to add up the estimated present population from the grid level to the zoning system level when so specified, by using the mapping between grid and zones given as input, based on the dataset ID and hierarchical level (if it applies) specified via configuration.

3.2.18 GRIDENRICHMENT

ID	DEFINITION
TSS-GE-001	The software shall be able to read INSPIRE Grid data object partitioned by quadkey with schema specified in I.28 INSPIRE Grid.
TSS-GE-002	The software shall be able to read transportation data in a format and schema specified in I.33 Transportation.
TSS-GE-003	The software shall be able to read landuse data in a format and schema specified in I.32 Landuse.
TSS-GE-004	The software shall ensure that all processing steps are compliant with spatial data processing standards and ensure accurate spatial data manipulation to prevent data integrity issues.
TSS-GE-005	The software shall include error handling mechanisms to manage missing or incomplete data inputs.
TSS-GE-006	The software shall be able to calculate landuse prior probabilities and path loss exponent coefficient for each input grid tile using landuse and transportation data. This shall be optional based on configuration parameter.
TSS-GE-007	If landuse and transportation data are not available or prior probabilities and path loss exponent not used in pipeline application the software shall skip corresponding processing steps and assign value 1 to both variables.
TSS-GE-008	The software shall write grid data object following schema specified in I.31 Enriched Grid partitioned by quadkey.
TSS-GE-009	The software shall be able to convert transportation lines to polygons using predefined buffer widths specific to each transportation class (primary, secondary, tertiary, pedestrian, railroad) as specified in the configuration.
TSS-GE-010	The software shall be able to perform spatial operations to intersect and merge landuse polygons with transportation polygons based on their geographical boundaries in a way that transportation polygons do not overlap with landuse polygons.
TSS-GE-011	The software shall create grid tiles polygons from grid tiles IDs
TSS-GE-012	The software shall intersect grid tiles to calculate the proportion of each landuse category within the boundaries of each grid tile. This includes calculating the area of each landuse category within a tile as a percentage of the total tile area.
TSS-GE-013	If any landuse category is missing in any given grid tile the software shall assign 0.0 as area share value.
TSS-GE-014	If there are no landuse categories in any given grid tile the software shall assign 1.0 as open_area category area share value.
TSS-GE-015	The software shall apply the predefined weights to the proportions of landuse categories within each grid tile to calculate the landuse prior probabilities. The software shall normalize these probabilities so that the sum across the whole grid equals one.
TSS-GE-016	The software shall apply the predefined weights to the proportions of landuse categories within each grid tile to calculate the path loss exponent coefficient for each tile.
TSS-GE-017	The software shall provide functionality to configure the weights for the calculation of both landuse prior probabilities and path loss exponent coefficients. This configuration should allow adjustment of weights for each landuse category.

3.2.19 GEOZONESGRIDMAPPING

ID	DEFINITION
TSS-ZGM-001	The software shall be able to read INSPIRE Grid data object partitioned by quadkey with schema specified in I.28 INSPIRE Grid.
TSS-ZGM-002	The software shall be able to perform mapping of either administrative units data or other geographic zones data to grid. Which zoning type to use shall be defined in config file
TSS-ZGM-003	If ‘administrative units’ is the selected zoning type, the software shall be able to read administrative zoning data in a format and schema specified in I.34 Administrative Units.
TSS-ZGM-004	If ‘other geographic zones’ is the selected zoning type, the software shall be able to read other geographic zoning data in a format and schema specified in I.35 Geographic Zones.
TSS-ZGM-005	The software shall be able to perform mapping with any datasets of a given zoning type based on configuration parameter.
TSS-ZGM-006	The software shall perform spatial intersection operation using grid tiles centroids and zone polygons.
TSS-ZGM-007	The software shall extract number of hierarchical levels from zoning dataset. If number of levels is more than 1, so the dataset is hierarchical, mapping shall be performed on the lowest level of hierarchy
TSS-ZGM-008	For each grid tile the software shall extract zone IDs of all levels of hierarchy and combine them into hierarchical id using “|” as a separator between levels.
TSS-ZGM-009	If a grid tile centroid doesn’t spatially intersect with any of the given zones’ polygons, zone_id and hierarchica_id shall be set to ‘undefined’
TSS-ZGM-010	The software shall write grid data object following schema specified in I.36 Zones – Grid Map partitioned by dataset_id.

3.2.20 MIDTERMPERMANENCEESTIMATION

ID	DEFINITION
TSS-MPE-001	The software shall be able to read daily permanence score input data from parquet files stored partitioned by year (YYYY), month (MM), day (DD), id_type, and user_id_modulo.
TSS-MPE-002	The software shall be able to read holiday dates input data from parquet files.
TSS-MPE-003	The software shall be able to read the I.21 Daily Permanence Score and I.40 Holiday Dates Calendar Data Objects.
TSS-MPE-004	The software shall be able to write mid-term permanence score metrics to parquet files stored partitioned by year (YYYY), month (MM), day_type, time_interval, id_type, and user_id_modulo.
TSS-MPE-005	The software shall write results data following schema specified in I.38 Mid-Term Permanence Metrics for all full months provided in the configured processing interval.
TSS-MPE-006	The software shall be able to read from a configuration file the start and end month for interval to process in the format YYYY-MM.
TSS-MPE-007	The software shall be able to read from a configuration file the country of study for which to consider holiday dates.
TSS-MPE-008	The software shall be able to read from a configuration file the number of days (as a positive integer) for which to include data from the previous month. This parameter will determine the number of calendar days, for which the values of the daily permanence score data object are read, prior to the month currently being processed. For instance, if the currently processable month is defined as 05-2024, and the parameter value is 15, rows from the daily permanence data object corresponding to dates between 16-04-2024 and 30-04-2024 are also included in the processing for mean and standard deviation calculation of regularity indices, but not for frequency calculation.
TSS-MPE-009	The software shall be able to read from a configuration file the number of days (as a positive integer) for which to include data from the next month. This parameter will determine the number of calendar days, for which the values of the daily permanence score data object are read, after the month currently being processed. For instance, if the currently processable month is defined as 05-2024, and the parameter value is 15, rows from the daily permanence data object corresponding to dates between 01-06-2024 and 15-06-2024 are also included in the processing for mean and standard deviation calculation of regularity indices, but not for frequency calculation.
TSS-MPE-010	The software shall be able to read from a configuration file the definition of a day based on its start hour. For example, one might want to consider that a day D starts at 4 AM. The definition shall be limited to full hours, and the parameter must be an integer between 0 and 23. All sub-monthly intervals shall be defined using these borders. For example, Monday interval shall be defined from Monday 4 AM until Tuesday 4 AM.
TSS-MPE-011	The software shall be able to read from a configuration file the definition of 'night_time' hours as a list of different start and ending hours, in HH:MM format, for example 18:45 to 08:15. Allowed values of the minutes (MM) are 00, 15, 30, and 45. Following the definition of a day based on its start hour, specified via configuration, the 'night_time' will belong to the date that contains its start hour. It is allowed that the 'night_time' hours cross the limit between two dates. It is not allowed, however, that when the 'night_time' end hour is different from 00:00 and the 'night_time' start hour is earlier than the day start hour, the 'night_time' end hour is earlier than the 'night_time' start hour. Example of non-allowed configuration: day start hour equal to 4, 'night_time' start hour equal to 03:30, 'night_time' end hour equal to 01:00 (01:00 < 03:30 < 04:00).
TSS-MPE-012	The software shall be able to read from a configuration file the definition of 'working_hours' as a list of different start and ending hours, in HH:MM format, for example 08:00 to 17:00. Allowed values of the minutes (MM) are 00, 15, 30, and 45. Following the definition of a day based on its start hour, specified via configuration, the 'working_hours' will belong to the date that contains its start hour. It is not allowed that the 'working_hours' cross the limit between two dates. It is also not allowed that when the 'working_hours' end hour is different from 00:00 and the 'working_hours' start hour is earlier than the day start hour, the 'working_hours' end hour is earlier than the 'working_hours' start hour. Example of non-allowed configuration: day start equal to 4, 'working_hours' start hour equal to 03:30, 'working_hours' end hour equal to 01:00 (01:00 < 03:30 < 04:00).

ID	DEFINITION
TSS-MPE-013	The software shall be able to read from a configuration file the definition of ‘evening_hours’ as a list of different start and ending hours, in HH:MM format, for example 08:00 to 17:00. Allowed values of the minutes (MM) are 00, 15, 30, and 45. Following the definition of a day based on its start hour, specified via configuration, the ‘evening_hours’ will belong to the date that contains its start hour. It is not allowed that the ‘evening_hours’ cross the limit between two dates. It is also not allowed that when the ‘evening_hours’ end hour is different from 00:00 and the ‘evening_hours’ start hour is earlier than the day start hour, the ‘evening_hours’ end hour is earlier than the ‘evening_hours’ start hour. Example of non-allowed configuration: day start equal to 4, ‘evening_hours’ start hour equal to 03:30, ‘evening_hours’ end hour equal to 01:00 (01:00 < 03:30 < 04:00).
TSS-MPE-014	The software shall be able to check that the time slot duration and limits of the daily permanence score input data is compatible with the ‘night_time’, ‘evening_hours’, and ‘working_hours’ time intervals defined in the configuration, and stop the execution and warn the user when one of the required dates has an incompatible time slot duration.
TSS-MPE-015	The software shall be able to read from a configuration file the definition of weekend start and end days by specifying values between 1 and 7, starting from Monday as 1, Tuesday as 2, up to Sunday as 7. For example, if the start day is 6 and the end day is 7, and the start hour of the day was specified as 4 AM, then the weekend starts at 4 AM of the Saturday and ends at 4 AM of the Monday.
TSS-MPE-016	The allowed sub-daily periods are: ‘all’, ‘night_time’, ‘evening_time’, ‘working_hours’, and the sub-monthly periods are: ‘all’, ‘workdays’, ‘holidays’, ‘weekends’, ‘mondays’, ‘tuesdays’, ‘wednesdays’, ‘thursdays’, ‘fridays’, ‘saturdays’, ‘sundays’.
TSS-MPE-017	The software shall be able to calculate metrics for the sub-monthly period ‘all’, defined as all dates within each month being studied.
TSS-MPE-018	The software shall be able to calculate metrics for the sub-monthly period ‘workdays’, defined as those days of the week that do not belong to the weekend and that are not marked as holidays in the country of study within each month being studied.
TSS-MPE-019	The software shall be able to calculate metrics for the sub-monthly period ‘holidays’, defined as those days marked as holidays in the country of study within each month being studied.
TSS-MPE-020	The software shall be able to calculate metrics for the sub-monthly period ‘mondays’, ‘tuesdays’, ‘wednesdays’, ‘thursdays’, ‘fridays’, ‘saturdays’, and ‘sundays’, defined by all of the corresponding days of the week within each month being studied.
TSS-MPE-021	The software shall be able to calculate metrics for the sub-daily period ‘all’, defined by all time slots contained in a date.
TSS-MPE-022	The software shall be able to calculate metrics for the sub-daily period ‘night_time’, defined by all time slots contained between the start and ending ‘night_time’ hours specified via configuration.
TSS-MPE-023	The software shall be able to calculate metrics for the sub-daily period ‘working_hours’, defined by all time slots contained between the start and ending ‘night_time’ hours specified via configuration.
TSS-MPE-024	The software shall be able to calculate metrics for the sub-daily period ‘evening_time’, defined by all time slots contained between the start and ending ‘night_time’ hours specified via configuration.
TSS-MPE-025	The combinations of sub-daily and sub-monthly periods for mid-term metrics calculation shall be provided in configuration file. The input structure has the shape of a dictionary, where the keys are the allowed and non-repeated values of sub-monthly periods, and the values are a list of allowed and non-repeated values of sub-daily periods surrounded by quotes. Example: {'all': ['all', ‘night_time', ‘evening’, ‘working_hours’], ‘workdays’: [‘night_time', 'working_hours']}.
TSS-MPE-026	The software shall read in the daily permanence score data object for the configured month and days in the previous and next month as defined by configuration parameters.
TSS-MPE-027	The software shall be able to calculate mid-term permanence score, mid-term frequency count and mid-term regularity indices per device and grid tile, as well as for the ‘unknown’ location, using permanence score values from Daily Permanence Score Data Object for the combinations of sub-daily and sub-monthly periods over each full month as specified via configuration.
TSS-MPE-028	The software shall be able to calculate the ‘device observation’ mid-term permanence score per device from the daily permanence score data, equal to the number of time slots of the sub-monthly and sub-daily periods over one full month that have a value of the daily permanence score equal to 1 in at least

ID	DEFINITION
	one grid tile; as well as the mid-term frequency count, equal to the number of dates in which at least one time slot of a grid tile the sub-monthly and sub-daily period over one full month has a daily permanence score equal to 1. These shall be done using the permanence score values from the Daily Permanence Score Data Object for the combinations of sub-daily and sub-monthly periods over each full month as specified via configuration.
TSS-MPE-029	The mid-term permanence score of a device in a grid tile or 'unknown' location shall be calculated as the summation of the daily permanence score of the device in that location over all time slots belonging to the corresponding sub-monthly and sub-daily periods and month being considered.
TSS-MPE-030	The mid-term frequency count of a device in a grid tile or 'unknown' location shall be calculated as number of days of the sub-monthly period of the month being studied in which that location has non-zero permanence score, i.e. permanence score values equal to 1, for any of the time slots in the sub-daily period being considered.
TSS-MPE-031	The mid-term regularity indices shall be calculated as the mean and the standard deviation of the temporal distance in number of days between consecutive dates of the sub-monthly period and month being studied with daily permanence score equal to 1 (i.e., greater than zero) in any of the time slots in the sub-daily period being considered. The start date shall be taken as the latest date of the sub-monthly period, daily permanence score equal to one in any time slot of the sub-daily period, and that belongs to the dates considered from the previous month for the calculation of these indices. In the case that none of these dates satisfy these conditions, the start date shall be taken as the earliest date considered. Analogously, the end date shall be taken as the first date among the dates of the following month that satisfies these conditions, and if it does not exist, the latest date is considered.
TSS-MPE-032	The mid-term metrics that refer to a specific grid tile shall have in their 'id_type' field a value equal to 'grid', together with the ID of that grid tile in the 'grid_id' field. If the metrics refer to an unknown location or to the device observation, they shall have the value 'unknown' or 'device_observation' respectively under both 'grid_id' and 'id_type' fields.

3.2.21 LONGTERMPERMANENCEESTIMATION

ID	DEFINITION
TSS-LPE-001	The software shall be able to read the I.38 Mid-Term Permanence Metrics Data Object stored as parquet partitioned by year (YYYY), month (MM), day_type, time_interval, id_type, and user_id_modulo.
TSS-LPE-002	The software shall be able to read from a configuration file the start and end month for interval to process Mid-term Permanence Metrics in the format YYYY-MM.
TSS-LPE-003	The software shall be able to read from a configuration file the definition of a sub-yearly intervals. Potential implementation as a dictionary {'winter':[12,1,2]. 'summer':[5,6,7,8]}, where keys are names of sub-yearly interval and values are lists of months that constitutes this interval.
TSS-LPE-004	The software shall read in the monthly permanence metrics for the configured months.
TSS-LPE-005	The software shall be able to calculate long-term permanence score, long-term frequency count, long-term frequency mean, long-term frequency standard deviation and long-term regularity indices per device, grid tile and 'unknown' location (id_type = 'unknown') using monthly permanence metrics values from Monthly Permanence Score Data Object for all combinations of sub-daily, sub-monthly and sub-yearly periods set in the configuration file over all months in the given time interval.
TSS-LPE-006	The combinations of sub-daily, sub-monthly and sub-yearly periods for long-term metrics calculation shall be provided in configuration file. Potential implementation as a dictionary of dictionaries: {'all':{'all': ['all', 'night_time', 'evening', 'working_hours'], 'working_days': ['night_time', 'working_hours']}, 'summer':{ 'weekends': ['all']}}, where keys are sub-yearly periods and values are dictionaries of sub-monthly periods as keys and and lists of sub-daily periods as values.
TSS-LPE-007	The software shall perform validation that all configured for processing sub-daily and sub-monthly periods are present in Mid-term Permanence Metrics Data Object and notify the user about missing combinations and stop processing.
TSS-LPE-008	Long-term permanence score shall be calculated as the sum of the tile monthly permanence scores ('mps') from Mid-term Permanence Metrics Data Object.
TSS-LPE-009	Long-term frequency count shall be calculated by sum of the monthly frequency counts from Mid-term Permanence Metrics Data Object. Long-term frequency mean shall be calculated as the mean of the monthly frequency count values of months belonging to the period of reference. The Long-term frequency std shall be calculated as the std of the monthly frequency count values of months belonging to the period of reference.
TSS-LPE-010	Long-term regularity indices per tile are calculated by taking the mid-term monthly mean distances between consecutive permanencies in the given tile and by computing the mean and the standard deviation of the mean distances.
TSS-LPE-011	The software shall be able to calculate the long-term 'device observation' metrics: • long-term device observation permanence score per device by summing up 'mps' column values of id_type = 'device_observation' from Mid-term Permanence Metrics Data Object • long-term device observation frequency by summing up 'frequency' column values of id_type = 'device_observation' from Mid-term Permanence Metrics Data Object Metrics shall be calculated for all combinations of sub-daily, sub-monthly and sub-yearly periods set in the configuration file over all months in the given time interval.
TSS-LPE-012	The mid-term metrics that refer to a specific grid tile shall have in their 'id_type' field a value equal to 'grid', together with the ID of that grid tile in the 'grid_id' field. If the metrics refer to an unknown location or to the device observation, they shall have the value 'unknown' or 'device_observation' respectively under both 'grid_id' and 'id_type' fields.
TSS-LPE-013	The software shall write results data following schema specified in I.39 Long-Term Permanence Metrics Object for the whole period provided in the configured processing interval.

3.2.22 USUALENVIRONMENTLABELING

ID	DEFINITION
TSS-UEL-001	The software shall be able to read the I.39 Long-Term Permanence Metrics Data Object stored as parquet partitioned by start_date, end_date and user_id_modulo.
TSS-UEL-002	The software shall be able to read following threshold parameters for labeling from the configuration file: • gap_ps_threshold (integer): the threshold of the difference in long permanence score values between consecutive tiles ordered by long term permanence score in descending order. Used to filter out tiles with long permanence score difference above this value and all the tiles following in descending long term permanence score values order. Default: 1 if only tiles with highest score are to be kept • total_ps_threshold (float): the total device permanence score assigned in reference period below which the user is not assigned a usual environment label, and is flagged as 'rarely observed'. Default: 300 (average PS=5 per day in 60 days, when the default value for the period of reference length is 6 months) • freq_days_treshold (float): the percentage out of total number of days when device has permanence in the reference period below which the user is not assigned a usual environment label and is flagged as 'rarely observed'. Default: 30 (unit: percentage) • ue_gap_ps_threshold (float): same as gap_ps_threshold, but used for filtering in UE labeling. Default: 20 (20 % of the highest permanence score value) • ue_ps_threshold (float): the percentage of permanence scores in top tiles out of the sum of daily device observation values in reference period. Tiles above this threshold are labeled as Usual Environment tiles. Default value: 70 (unit: percentage) • ue_ndays_threshold (float): the percentage of the sum of the number of days with non-zero permanence in top tiles out of total number of non-zero permanence days in reference period. Tiles above this threshold are labeled as Usual Environment tiles. Default value: 70 (unit: percentage) • home_ps_threshold (float): the percentage of permanence scores in top tiles out of the sum of daily device observation values in reference period. If the device has at least this value in top tiles these tiles are labeled as Home Location. Default value: 80 (unit: percentage) • home_ndays_threshold (float): the percentage of the sum of the number of days with non-zero permanence in top tiles out of total number of non-zero permanence days in reference period. If the device was in the first tile or group of tiles at least this value these tiles are labeled as Home Location. Default value: 80 (unit: percentage). • work_ps_threshold (float): minimum percentage of sum of permanence score during working days and daytime in top tiles out of total permanence score during working days and daytime, for a tile to be labelled as a work location tile. Default value: 80 (unit: percentage). • work_ndays_threshold (float): minimum percentage of non-zero permanence score days during working days and daytime in top tiles out of total number of non-zero permanence score days during working days and daytime, for a tile to be labelled as a work location tile. Default value: 80 (unit: percentage).
TSS-UEL-003	The software shall be able to read from a configuration file the start and end month for interval to process Long-term Permanence Metrics in the format YYYY-MM.
TSS-UEL-004	The software shall perform Usual Environment labeling and Home and Work locations labeling of tiles for each device.
TSS-UEL-005	The software shall perform validation that all combinations of periods required for labeling are present in Long-term Permanence Metrics Data Object, notify the user about missing combinations and stop further execution. Currently, the required periods are: • UE labeling - all days: all intervals • Home labeling - all days: all intervals, all days: night-time • Work labeling - work days: working hours
TSS-UEL-006	The software shall filter rarely observed devices using all days: all intervals combination based on the following rules:

ID	DEFINITION
	1. Filter devices for which 'lps' value in id_type = ‘device_observation' row < total_ps_threshold parameter into separate table. Mark such devices as filtered by rule ‘device_filter_1’ (rarely observed). 2. Filter devices for which 'total_frequency' value in id_type = ‘device_observation' row < freq_days_threshold parameter into same as above separate table. Mark such devices as filtered by rule ‘device_filter_2’ (discontinuously observed). Devices filtered during this step shall not be used for any labeling.
TSS-UEL-007	The software shall label tiles as Usual Environment tiles based on following algorithm: 1. For all days : all intervals combination: a. Get the highest value of a long permanence score (‘lps’) of a device over all grid tiles (PS max). b. Calculate the difference in ‘lps’ values between consecutive tiles ordered by ‘lps’ values in descending order. c. Find tiles that have a difference in ‘lps’ value > ue_gap_threshold (default is 20% of PS max), filter out these tiles and all the tiles with ‘lps’ values below. 2. For each tile in the remaining tiles group check if its ‘lps’ value is at least ue_ps_threshold (default is 70% of 'lps' value in id_type = ‘device_observation' row of all days: all intervals combination). Tiles for which this condition is fulfilled are labeled as UE tiles. Labeling rule code: ‘ue_1’. 3. For tiles which have not got UE label in previous step perform the same check for all other combinations of day types and periods. If condition is fulfilled for any of the combinations, label tiles as UE tiles. Labeling rule code: ‘ue_2’. 4. If no UE label being assigned after all of the above steps, add ue labeling rule ‘ue_na’ - label not assigned.
TSS-UEL-008	The software shall save the rule based on which tiles were labeled as UE tiles using predefined rule codes.
TSS-UEL-009	The software shall label tiles as Home location tiles based on following algorithm: 1. For all days: all intervals combination: a. Calculate the difference in long permanence scores (‘lps’) between consecutive tiles ordered by ‘lps’ values in descending order. b. Find first tile that have a difference in ‘lps’ value > gap_ps_threshold (default is 1), filter out this tile and all the tiles with ‘lps’ values below from all period combinations used for home labeling. 2. For each tile in the remaining tiles group check if its ‘lps’ value is at least home_ps_threshold. (default is 80% of 'lps' value in id_type = ‘device_observation' row of all : all periods combination). Such tiles are labeled as Home tiles. Labeling rule code: ‘h_1’. 3. If no home label being assigned, repeat this condition check for all days: night_time combination. Tiles that fulfilled this condition are labeled as Home tiles. Labeling rule code: ‘h_2’. 4. If no home label being assigned, check if the device was in the tiles at least home_ndays_threshold (default is 80% of 'total_frequency' value in id_type = ‘device_observation' row of all days: all intervals combination). Tiles that fulfilled this condition are labeled as Home tiles. Labeling rule code: ‘h_3’. 5. If no Home label being assigned after all of the above steps, add labeling rule ‘loc_na’ - label not assigned.
TSS-UEL-010	The software shall label tiles as Work location tiles based on following algorithm: 1. For working days : working time combination: a. Calculate the difference in long permanence scores (‘lps’) between consecutive tiles ordered by ‘lps’ values in descending order. b. Find first tile that have a difference in ‘lps’ value > gap_ps_threshold (default is 1), filter out this tile and all the tiles with ‘lps’ values below from all period combinations used for work labeling. 2. For each tile in the remaining tiles group check for workdays: working_hours combination if its ‘lps’ value is at least work_ps_threshold. (default is 70% of 'lps' value in id_type =

ID	DEFINITION
	'device_observation' row of all days: all intervals combination). Tiles for which this condition is fulfilled are labeled as Work tiles. Labeling rule code: 'w_1'. 3. If no work label being assigned, for each tile in the remaining tiles group check for workdays : working_hours periods combination if its 'total_frequency' value is at least work_ndays_threshold (default value is 70% of 'total_frequency' value in id_type = 'device_observation' row of all days: all intervals combination). Tiles for which this condition is fulfilled are labeled as Work tiles. Labeling rule code: 'w_2'. 4. If no Work label being assigned after all of the above steps, add labeling rule 'loc_na' - label not assigned.
TSS-UEL-011	The software shall save the rule based on which tiles were labeled as Home or Work tiles using predefined rule codes.
TSS-UEL-012	The software shall perform labeling for each location type individually, so same tile can be labeled multiple times.
TSS-UEL-013	The software shall write results data following schema specified in I.37 UE Labels Data Object for all full months provided in the configured processing interval. Partitioned by year, month, day, user_id_mod.
TSS-UEL-014	The software shall produce following quality metrics: a. Number of tiles in each labeling rule. b. Number of home location tiles which are not labeled as UE. c. Number of work location tiles which are not labeled as UE. d. Number of devices which are filtered out as rarely observed and discontinuously observed.
TSS-UEL-015	The software shall write quality metrics following schema and format specified in I.43 Labeling Quality Metrics.

3.2.23 USUALENVIRONMENTAGGREGATION

ID	DEFINITION
TSS-UEA-001	The software shall be able to read from a configuration file the ‘start month’ and ‘end month’ for the interval to select from the I.37 UE Labels input in the format YYYY-MM.
TSS-UEA-002	The software shall be able to read from a configuration file a boolean ‘use land use’ parameter indicating whether land use information will be used for the usual environment aggregation.
TSS-UEA-003	The software shall be able to read an I.37 UE Labels Data Object from parquet files stored partitioned by start_date, end_date & user_id_modulo.
TSS-UEA-004	The software shall be able to read an I.31 Enriched Grid Data Object from parquet files stored partitioned by quadkey.
TSS-UEA-005	The software shall load all usual environment grid tiles for all devices from the I.37 UE Labels input data for the selected start and end month regardless of the label.
TSS-UEA-006	The software shall load a tile weight (tw) for each grid tile from the ‘prior_probabilty’ column of the I.31 Enriched Grid Data Object if ‘uniform_tile_weights’ parameter has been set as False. If this parameter has been set as True, all grid tiles shall be assigned tw = 1.
TSS-UEA-007	The software shall calculate, for each device, its device tile weight value (weight_td) for each of the device’s usual environment grid tiles. This is achieved by using the following formula for each tile i: weight_td (grid_i) = tw (grid_i) / Σj(tw (grid_j)) Where: • grid_i: is a target grid tile, i.e., a tile that is included in the current device’s usual environment, and for which we are calculating pue. • weight_td (grid_i): is the weight of the device in the target grid tile (grid_1). • tw (grid_i): is the tile weight for target grid tile (grid_1), either 1 or coming from the enriched grid data. • Σj(tw (grid_j)): is the sum of the tile weights of all the grid tiles in the device’s usual environment.
TSS-UEA-008	The software shall sum the weight_td values of all devices in each grid tile to obtain the aggregated weighted device counts of each tile.
TSS-UEA-009	The software shall produce an output I.44 Aggregated Usual Environments Data Object with the final usual environment count of each tile in parquet files partitioned by start_date and end_date.

4 TECHNOLOGY STACK

Based on the general software requirements (see section 3.1 General requirements), the technology stack of the software has been defined. **Apache Spark** framework has been chosen as it perfectly complies with the requirements. The only aspect not directly addressed by the Apache Spark framework is the 'spatial computations' requirement (TSS-GEN-017), as Spark doesn't natively support geospatial operations. However, **Apache Sedona** is an extension built on Apache Spark whose purpose is to perform efficient spatial computations with Spark. By adding this extension to Spark, a framework that satisfies all the requirements is met. Spark is natively written in the Scala programming language, nonetheless it supports bindings for multiple programming languages such as Java, Python and R. From these languages, the **Python** programming language is chosen since it performs exceptionally the requirements.

A comprehensive list of the technology stack selected for the development of the software together with the rationale behind its selection is presented in Table 5.

Table 5: Software technology stack

TECHNOLOGIES		RATIONALE	REMARKS
Operating System	Linux	Cloud environments and containers usually run Linux as operating system. Most Linux distributions are open-source and free. Linux is a reliable and secure operating system due to its design and open-source nature.	
Development	Software Language: Python 3.7+	Standard data science, data analytics and data engineering software language. It has multiple open-source, state-of-the-art data processing libraries, such as numpy, pandas and pyspark. Most popular software language at the moment, with more than 25% of total share[3]. This increases the probability of external users contributing to the open-source project. Supported by all popular IDEs. Supported by all cloud computing providers.	The Python version may be constrained by the cloud infrastructure of the MNO operator.
	Data processing engine: Spark (Pyspark)	Spark is an open-source data processing framework ideal for big data pipelines. It provides bindings for python with the pyspark library. Spark can be deployed in a single machine or a cluster depending on the data workload. Spark has a machine learning library which allows data scientists to train and deploy models at scale. Spark can be deployed in popular cloud managed clusters (e.g. AWS EMR, GCP Dataproc, Azure Hdinsight).	Due to privacy constraints, MNOs require that the software runs in their closed cloud environment. This restriction may limit the available computing resources for software execution. Based on project team previous experience working with MNOs, most of them usually have deployed a cloud-managed map-reduce cluster, such as AWS EMR or GCP Dataproc. Spark can seamlessly operate on these environments, which

[3] PYPL PopularitY of Programming Language index

TECHNOLOGIES		RATIONALE	REMARKS
		Spark provides native support for local file systems, distributed file systems (HDFS) and blob storage systems.	simplifies the future deployment of the software.
	Geospatial data processing framework: Apache Sedona	Apache Sedona is a framework built on top of Apache Spark for processing high workloads of geospatial data. Apache Sedona provides bindings for the python language. Apache Sedona provides standard spatial operations such as spatial joins, nearest neighbour searches, range queries and spatial indexes. As it is built on top of Apache Spark, geospatial data can be incorporated in machine learning models.	
Code & Components Orchestration	Custom module	An ad-hoc/custom orchestration module has been developed for the pipeline. It is designed as a modular piece that could be replaced in the future by a more complex/sophisticated engine if needed.	
Data	File Format: ● Parquet/GeoParquet	It allows the possibility of working with both centralized and distributed computing systems. Standard and recommended file format of the Spark framework.	
	Data storage: ● Centralized environment ○ Local file-system ● Distributed environment ○ HDFS (Hadoop) ● Cloud environment	The data storage should be invisible to the software. If the system has been setup correctly, the software should be able to read data from the given path locations. Spark provides support for local file systems, distributed file systems (HDFS) and blob storage systems.	

TECHNOLOGIES		RATIONALE	REMARKS
	○ Blob storage (AWS S3, GCP Cloud storage, Azure Blob storage)		
Testing	Pytest	Ability to run multiple tests in parallel for optimised test suite execution times. Easy-to-use syntax. Automatic test discovery. Support for HTML reports on coverage and testing results.	
Source Control	GIT	As the use of GitHub is a requirement, the source control engine should be git, as it is the main engine supported by the platform. Git is the most popular source control engine.	
Code Documentation	Code style: ● PEP8	PEP8 is the standard coding style for python software. It makes code more maintainable and readable. It is ideal for open-source projects, as it facilitates contribution due to the fact that all the code has a homogeneous and well-known style.	
	Docstring style: ● Google Docstring Style	Readable and compact. Ideal for small docstrings. Most popular docstring format. Makes contribution easier. Supports automatic generation deployment of documentation in HTML files.	
	Code Documentation engine: ● MkDocs	MkDocs uses Markdown, a lightweight markup language, for content creation, making it easy for users to write and update documentation.	

TECHNOLOGIES	RATIONALE	REMARKS
	MkDocs includes a built-in development server, enabling users to preview their documentation locally before publishing it, facilitating iterative improvements. Multiple plugin support: search-bar, table of contents, versioning, tabs… Code modules automatic documentation generation from python docstrings in Google style.	

5 DESIGN

This chapter describes all the software design decisions considered in the development of the software. First, general design aspects are presented (see section 5.1 General design), providing information about the data design, software design, infrastructure design, version control and software artefacts design. Secondly, the design considerations of each software component are presented (see section 5.2 Component design).

5.1 GENERAL DESIGN

5.1.1 DATA DESIGN

Big Data demands a meticulous and strategic approach to data design decisions within the pipeline architecture. In the realm of processing vast volumes of information, every design choice reverberates across the entire ecosystem, influencing the efficiency, scalability, and ultimately, the success of the data pipeline. The decisions made in the early stages of data design impact considerably the pipeline's ability to handle, analyse, and derive meaningful insights from massive datasets.

5.1.1.1 DATA FORMAT

All data processed by the pipeline components will be in **(geo)parquet** format as it is the ideal format for working with the spark framework. It may be the case that some input data is not presented in (geo)parquet format (e.g. csv, json, etc.). Hence, a format transformation process is needed. In this regard, a data ingestion process is defined which incorporates all data into the system in the desired (geo)parquet format.

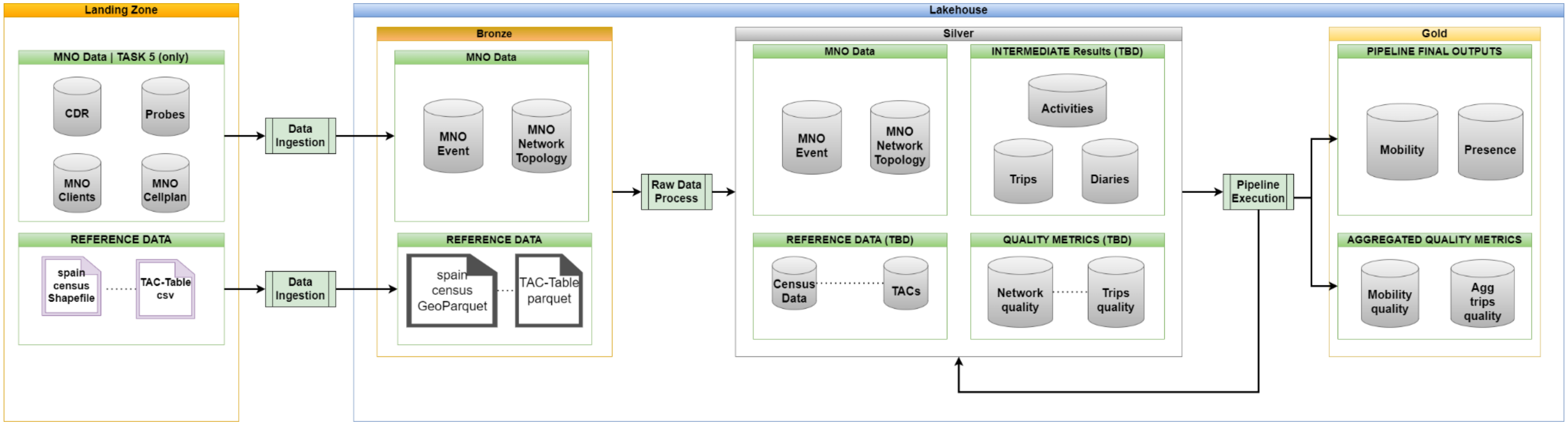
5.1.1.2 DATA STORAGE

As the solution is designed to work in a cloud architecture, its blob storage can be defined as a *Lakehouse* which guarantees that all data in it is accessible by any computing infrastructure deployed within the cloud. When executing locally, the OS filesystem can be used as *Lakehouse*. A medallion architecture is proposed for classifying the data within the *Lakehouse* based in three levels in which each one represents a more advanced level of data processing:

- **Bronze**: almost raw data ingested in the desired format. Mainly MNO data and contextual data.
- **Silver:** data model of the project. Enriched MNO data, intermediate outputs and quality metrics and quality warnings datasets.
- **Gold:** final aggregated outputs (final indicators) of the pipeline.

As previously mentioned, it is expected that not all input datasets will be in the (geo)parquet format, so a **landing zone** has been defined to address this aspect. In this zone all data as it is obtained is centralized so it can be then ingested into the bronze layer of the *Lakehouse*.

Figure 2: Medallion architecture scheme



5.1.1.3 MNO DATA DELIVERY

Event and network topology data for each day will be provided by MNOs at the Bronze level.

This data must be stored with the following folder structure: bronze/<country>/<mno>/<data_type>/year=<YYYY>/month=<MM>/day=<DD>/*.parquet.

For example: **bronze/es/orange/events/year=2023/month=01/day=01/event_data.parquet.**

Under the day folder multiple parquet files could be stored. However, it is recommended to use parquet files with sizes of 512MB-1GB as defined in the official parquet documentation [1].

References:

[1] Apache parquet file-format configurations. Available: https://parquet.apache.org/docs/file-format/configurations/ [Accessed Nov. 23, 2023]

5.1.2 SOFTWARE DESIGN

The aim of this section is to describe all the software design decisions for the execution of a big data pipeline that processes and generates data in the data model described in section 5.1.1 Data design.

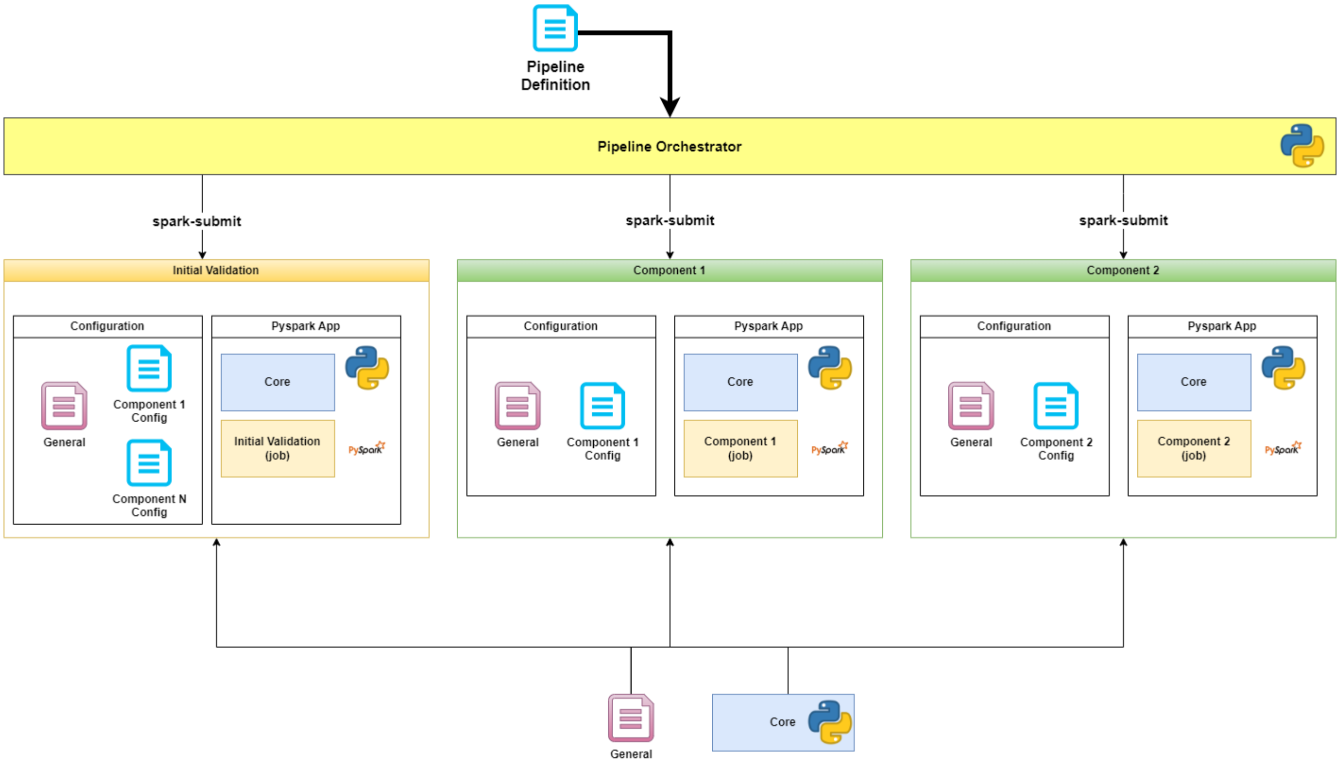
5.1.2.1 PIPELINE DESIGN

Processing multiple Big Data pipelines is the main functionality of the software to be developed. These pipelines can be divided into independent modules/components in which each one performs an ETL process. For this purpose the **isolated components** design principle has been applied which consists on having independent software processing units that do not share in-memory data and that, as long as they have all required input data, they can be executed without any dependency of other components. With this approach, components can be developed independently and will integrate without problems in a pipeline as long as the data objects definitions are adhered to. Furthermore, if the pipeline execution fails, the pipeline execution can be restored from the component that failed as all the previous components will have been executed correctly. Additionally, in the context of a *PySpark* application, using isolated components grants that the temporary and cache data of a Spark session will be completely cleaned after each execution.

Having each component as a single *PySpark* application allows the project to easily integrate with orchestration software. A component is defined as a Python package composed of a '*Core*' sub-package containing common functionalities of all components like logging, configuration, abstract classes and interfaces and a '*Components*' sub-package in which each component will represent a spark job. Besides the code, a spark job submission will include configuration text files. Two or more configuration files will be used for each component which can be categorized into two types:

- **General**: general & common configuration of the pipeline.
- **Component**: specific component configuration file(s) that can override some general configuration parameters.

Figure 3: Pipeline orchestration scheme



5.1.2.2 COMPONENTS DESIGN

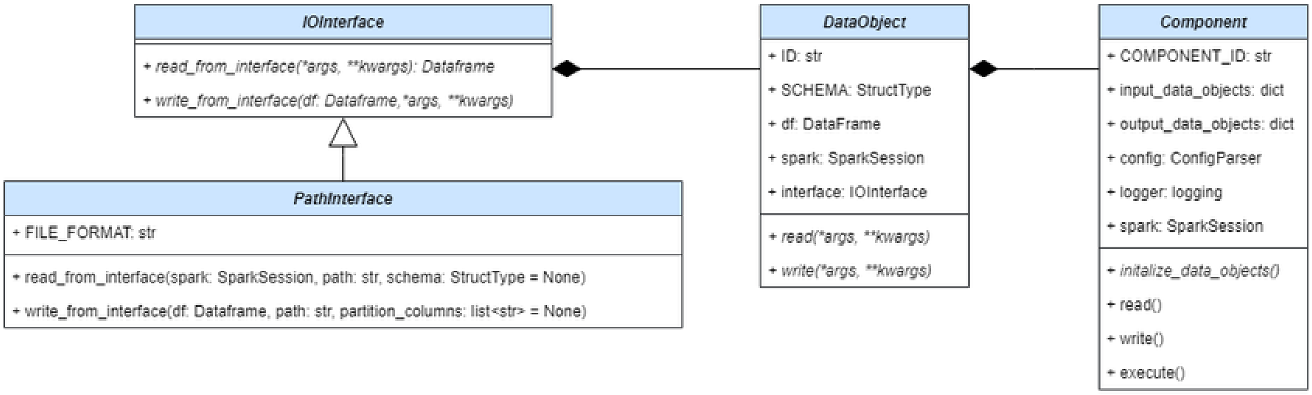
While every component executed in the pipeline corresponds to a single module to be executed as a spark-job, we can define different types of components depending on their purpose.

- **Ingestion:** these components are in charge of getting raw data in different file formats (csv, json, text, shapefile, etc.) and introducing it into the *Lakehouse* in a common file format: parquet & geoparquet. For demonstration/testing purposes, synthetic data is generated to simulate a pipeline execution. The synthetic data generating components is considered as an ingestion component.
- **Initial Validation:** it is a single component that performs pipeline setup verification checks. Its main purpose is to provide a 'fail fast' functionality for preventing small errors that will break the pipeline halfway like missing a configuration file for a component at a later stage of the pipeline. The verifications performed are:
  - Configuration files are valid;
  - Data can be read and written;
  - Component classes can be initialised.
- **Execution:** components that perform the functional logic of the pipeline. Each component performs ETL processes for a single functional step in the pipeline. It is recommended not to include too much functionality in a single component and instead split it up between execution components.
- **Quality:** these components perform validation and quality processes and store these analyses in the *Lakehouse*.

5.1.2.3 CLASS DIAGRAM

While the software will execute each component as a separate spark-job, the whole application can be conceived as a single python program which provides a component selection for single executions. The proposed software architecture is based in a '*Core*' package which contains the abstract classes and interfaces, '*DataObjects*' and common functionalities like the configuration, spark session and logging management.

Figure 4: Component and DataObject class diagram

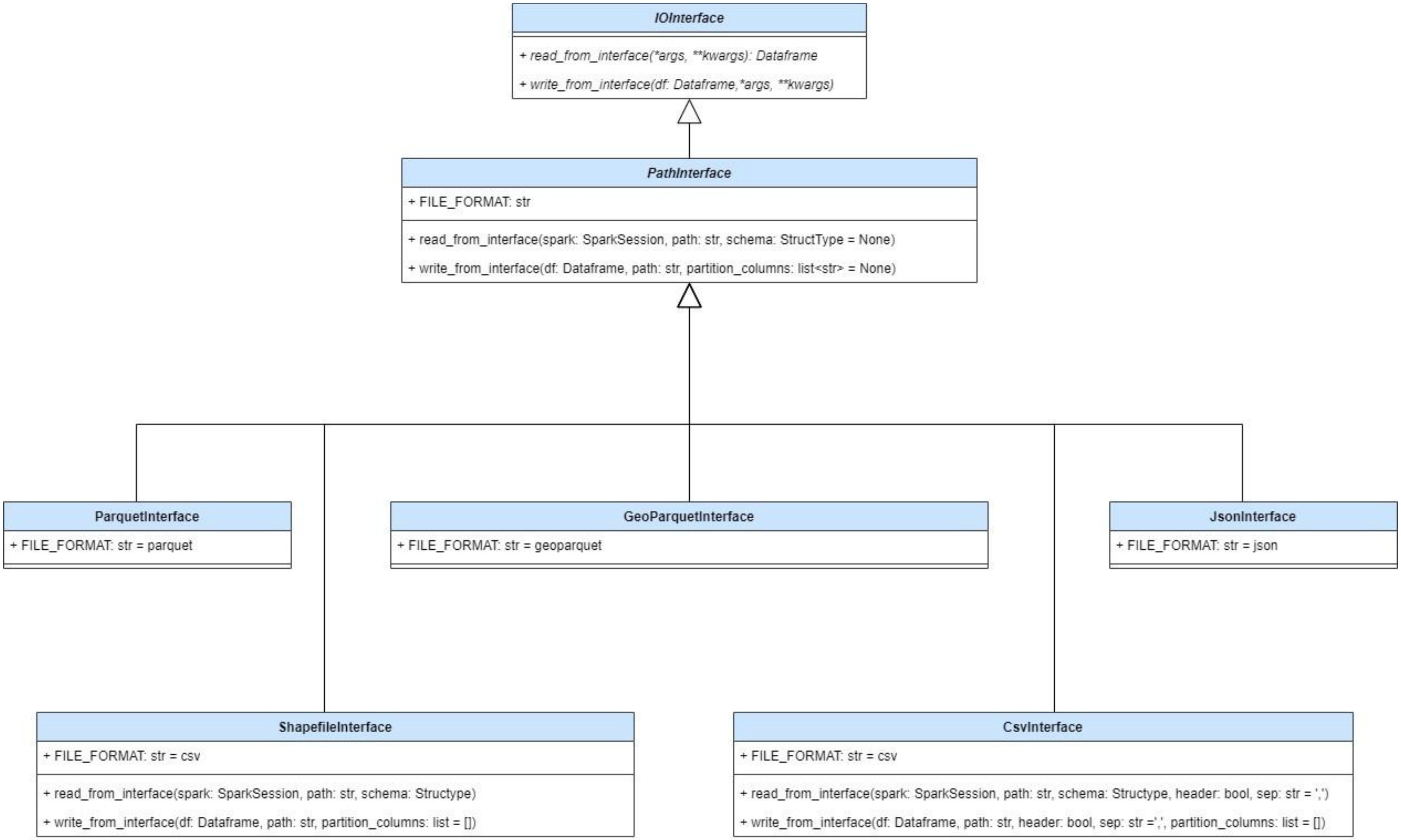

IOInterface
+ read_from_interface(*args, **kwargs): Dataframe
+ write_from_interface(df: Dataframe,*args, **kwargs)
PathInterface
+ FILE_FORMAT: str
+ read_from_interface(spark: SparkSession, path: str, schema: StructType = None)
+ write_from_interface(df: Dataframe, path: str, partition_columns: list<str> = None)
DataObject
+ ID: str
+ SCHEMA: StructType
+ df: DataFrame
+ spark: SparkSession
+ interface: IOInterface
+ read(*args, **kwargs)
+ write(*args, **kwargs)
Component
+ COMPONENT_ID: str
+ input_data_objects: dict
+ output_data_objects: dict
+ config: ConfigParser
+ logger: logging
+ spark: SparkSession
+ initalize_data_objects()
+ read()
+ write()
+ execute()

The most important class of the application is the '*Component*' class which is the abstract class that performs the read, transform and write operations. One of the key aspects of this software architecture is the use of the '*DataObject*' classes. Every data source is accessible for read and write operations through a '*DataObject*' class. This prevents multiple read and write definitions of a data source used in multiple components, eases the scalability of software and guarantees consistency in read and write operations as the same schema is always used. The '*DataObject*' class contains an '*IOInterface*' class which abstracts input and output operations perform on data sources. In the realm of Big Data is common that data can be given through different file types (csv, json, parquet…), databases or APIs; having a class that abstracts this access grants modularity and scalability as the incorporation/change of a data source only implies a modification in the '*IOInterface*' used by the '*DataObject*'. Thanks to the Spark framework, different file formats are read with the same code with minimum changes. An intermediate abstract class, called '*PathInterface*', that inherits from '*IOInterface*' is defined in order to prevent having duplication of code for reading and writing a file. From this class concrete classes for reading different file formats can be defined. The supported file formats are:

- Parquet
- Json
- Csv
- Shapefile
- Geoparquet.

The concrete classes for this file access require only a change to the FILE_FORMAT variable as the main logic is inherited by the parent class reducing code duplication. For special cases, like shapefiles, specific logic for reading a shapefile is needed as the Sedona Framework is used.

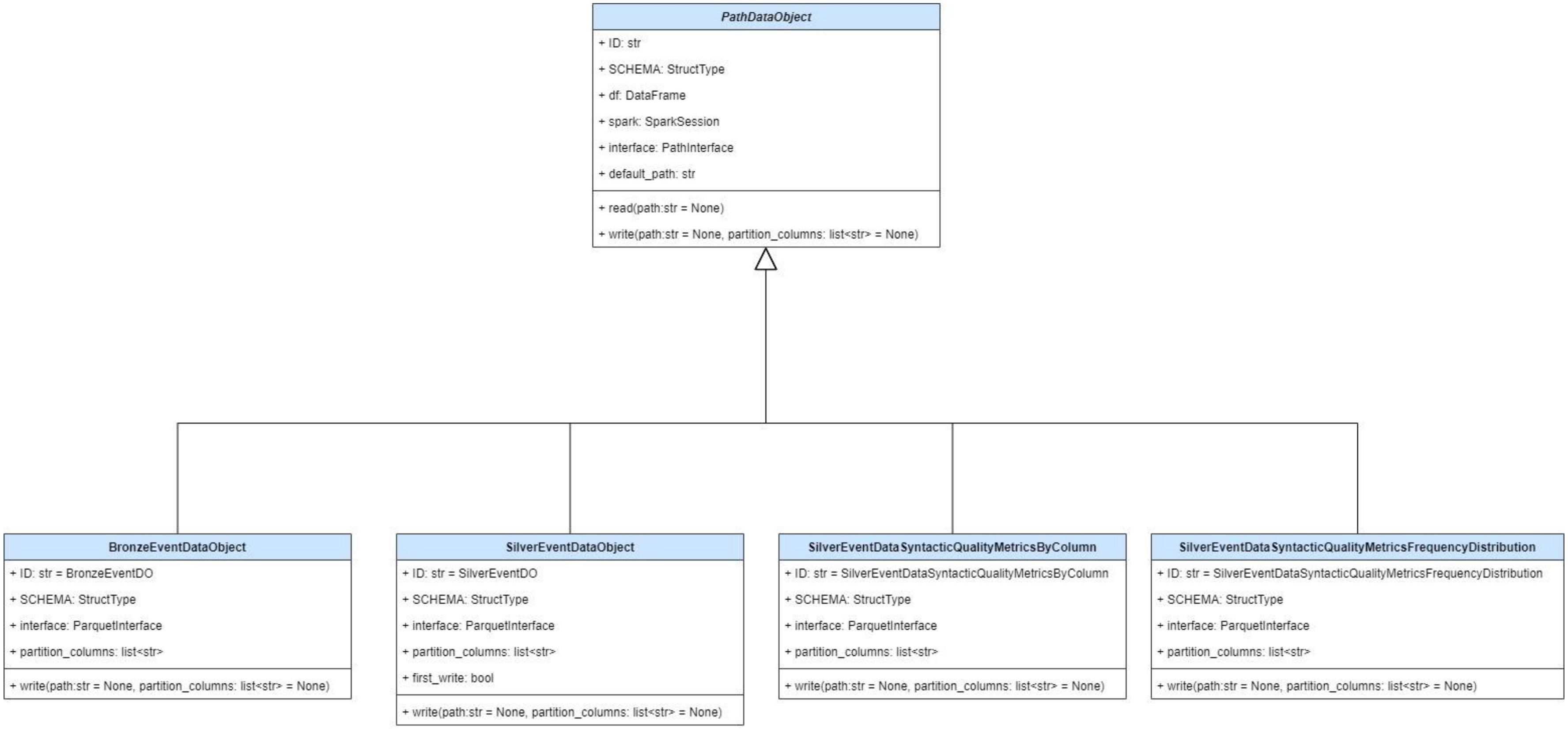
Figure 5: IO Interface class diagram



IOInterface
+ read_from_interface(*args, **kwargs): Dataframe
+ write_from_interface(df: Dataframe,*args, **kwargs)
PathInterface
+ FILE_FORMAT: str
+ read_from_interface(spark: SparkSession, path: str, schema: StructType = None)
+ write_from_interface(df: Dataframe, path: str, partition_columns: list<str> = None)
ParquetInterface
+ FILE_FORMAT: str = parquet
GeoParquetInterface
+ FILE_FORMAT: str = geoparquet
JsonInterface
+ FILE_FORMAT: str = json
ShapefileInterface
+ FILE_FORMAT: str = csv
+ read_from_interface(spark: SparkSession, path: str, schema: Structype)
+ write_from_interface(df: Dataframe, path: str, partition_columns: list = [])
CsvInterface
+ FILE_FORMAT: str = csv
+ read_from_interface(spark: SparkSession, path: str, schema: Structype, header: bool, sep: str = ',')
+ write_from_interface(df: Dataframe, path: str, header: bool, sep: str =',', partition_columns: list = [])

After defining all I/O interfaces, the '*DataObjects*' of the application can be defined. A '*DataObject*' class is defined for each '*DataObject*' defined in Annex I. These classes contain information about the data they are modeling, like a unique ID to identify the data, the data schema or the data type. They also hold the spark DataFrame and provide an easy and centralized way for developers to read and write data of a data object. This prevents multiple read and write implementations of a data object that is used in multiple components. Figure 5 shows an example of four different data object class diagrams.

Figure 6: DataObjects example class diagram


PathDataObject
+ ID: str
+ SCHEMA: StructType
+ df: DataFrame
+ spark: SparkSession
+ interface: PathInterface
+ default_path: str
+ read(path:str = None)
+ write(path:str = None, partition_columns: list<str> = None)
BronzeEventDataObject
+ ID: str = BronzeEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
SilverEventDataObject
+ ID: str = SilverEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ first_write: bool
+ write(path:str = None, partition_columns: list<str> = None)
SilverEventDataSyntacticQualityMetricsByColumn
+ ID: str = SilverEventDataSyntacticQualityMetricsByColumn
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
SilverEventDataSyntacticQualityMetricsFrequencyDistribution
+ ID: str = SilverEventDataSyntacticQualityMetricsFrequencyDistribution
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)

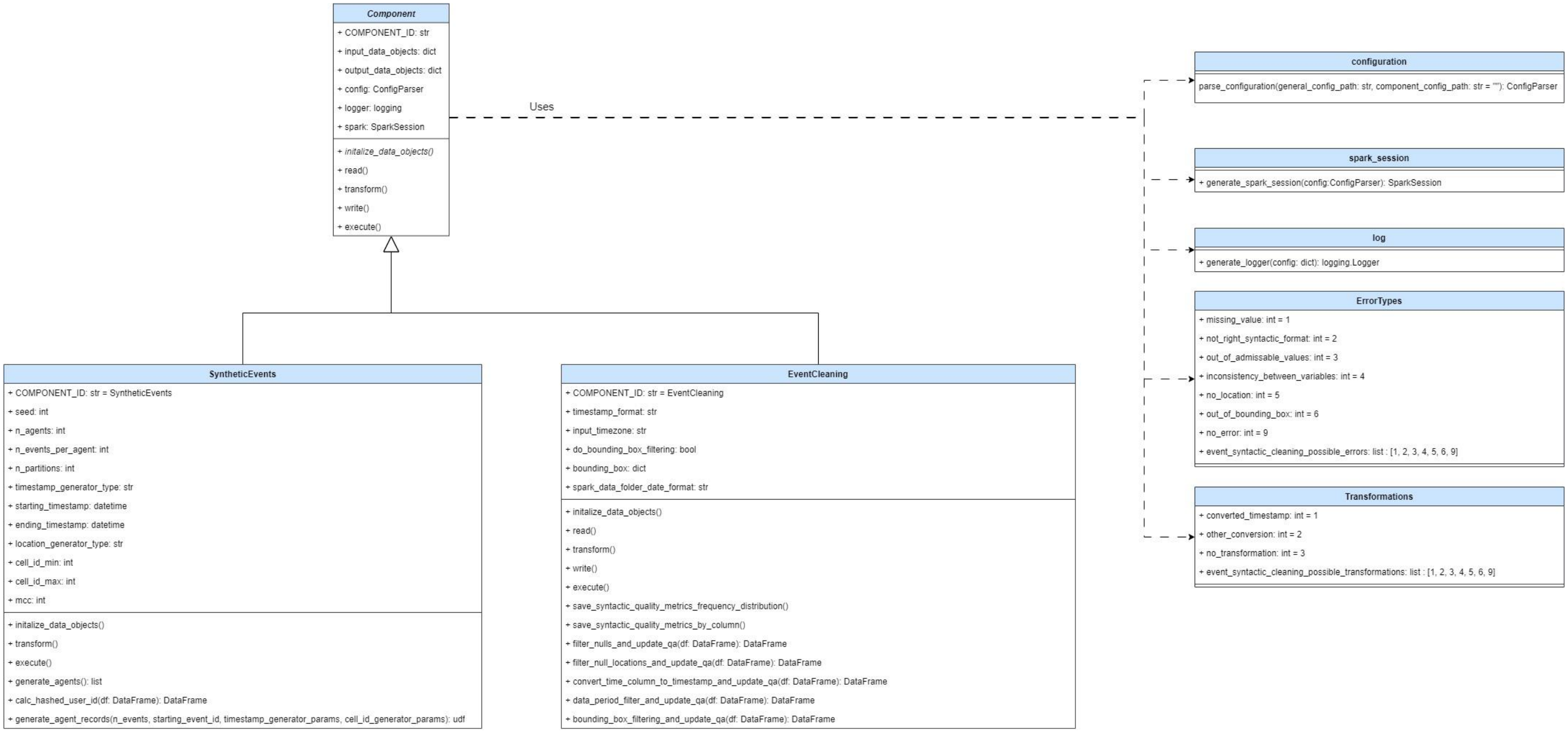
After modeling all the data objects that will be used in the release of the software, the components that make use of this data and perform the transformations are implemented. Each component can be thought as an abstract class which will read data, transform data and write data. They are the main point of execution as each component represents a step of the pipeline. All components also use some common functionalities which consist of:

- Reading configuration files
- Starting a Spark Session
- Initialize a Logger

Component classes will use the '*DataObjects*' associated to them to read and write data. All this logic can be centralised in an abstract class which delegates to its inherited classes the only responsibility to implement the concrete data transformations. This approach keeps codes modular and clean. Furthermore, it eases the development as developers only need to take care of implementing the logic of the transformations of the component.

In Section 5.2 Component design all components of the software are described with a class diagram that represents all the concrete objects that interact in the execution of the component.

Figure 7: Concrete component implementation class diagram


Component
+ COMPONENT_ID: str
+ input_data_objects: dict
+ output_data_objects: dict
+ config: ConfigParser
+ logger: logging
+ spark: SparkSession
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
Uses
configuration
parse_configuration(general_config_path: str, component_config_path: str = ""): ConfigParser
spark_session
+ generate_spark_session(config:ConfigParser): SparkSession
log
+ generate_logger(config: dict): logging.Logger
ErrorTypes
+ missing_value: int = 1
+ not_right_syntactic_format: int = 2
+ out_of_admissable_values: int = 3
+ inconsistency_between_variables: int = 4
+ no_location: int = 5
+ out_of_bounding_box: int = 6
+ no_error: int = 9
+ event_syntactic_cleaning_possible_errors: list : [1, 2, 3, 4, 5, 6, 9]
Transformations
+ converted_timestamp: int = 1
+ other_conversion: int = 2
+ no_transformation: int = 3
+ event_syntactic_cleaning_possible_transformations: list : [1, 2, 3, 4, 5, 6, 9]
SyntheticEvents
+ COMPONENT_ID: str = SyntheticEvents
+ seed: int
+ n_agents: int
+ n_events_per_agent: int
+ n_partitions: int
+ timestamp_generator_type: str
+ starting_timestamp: datetime
+ ending_timestamp: datetime
+ location_generator_type: str
+ cell_id_min: int
+ cell_id_max: int
+ mcc: int
+ initalize_data_objects()
+ transform()
+ execute()
+ generate_agents(): list
+ calc_hashed_user_id(df: DataFrame): DataFrame
+ generate_agent_records(n_events, starting_event_id, timestamp_generator_params, cell_id_generator_params): udf
EventCleaning
+ COMPONENT_ID: str = EventCleaning
+ timestamp_format: str
+ input_timezone: str
+ do_bounding_box_filtering: bool
+ bounding_box: dict
+ spark_data_folder_date_format: str
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
+ save_syntactic_quality_metrics_frequency_distribution()
+ save_syntactic_quality_metrics_by_column()
+ filter_nulls_and_update_qa(df: DataFrame): DataFrame
+ filter_null_locations_and_update_qa(df: DataFrame): DataFrame
+ convert_time_column_to_timestamp_and_update_qa(df: DataFrame): DataFrame
+ data_period_filter_and_update_qa(df: DataFrame): DataFrame
+ bounding_box_filtering_and_update_qa(df: DataFrame): DataFrame

5.1.2.4 CONFIGURATION DESIGN

One critical aspect of building robust big data pipelines is the management of configuration settings. To achieve enhanced flexibility, maintainability, and scalability, a prudent approach is to utilise a combination of a general configuration file and component-specific configuration files.

GENERAL CONFIGURATION FILE

The general configuration file serves as the overarching blueprint for the entire big data pipeline. It encapsulates settings that are applicable across multiple components, providing a centralised and standardised approach to configuration management. This file contains global execution settings, like logger settings, path values and spark-session settings. By consolidating these shared settings in a single file, the pipeline gains consistency and becomes more adaptable to changes in the overall infrastructure.

COMPONENT-SPECIFIC CONFIGURATION FILES

Complementing the general configuration file, each individual component within the big data pipeline is associated with its own specific configuration file. These files contain parameters tailored to the unique requirements and characteristics of each component. For instance, a data ingestion component might have settings related to data sources, formats, and ingestion frequencies, while an execution component may have parameters governing data transformation logic.

ADVANTAGES OF USING SEPARATED CONFIGURATION FILES

- **a. Modularity and maintainability:** separating configurations into distinct files promotes a modular design, allowing developers to focus on the specific requirements of each component. This modularity not only simplifies development but also streamlines maintenance efforts. When modifications or updates are necessary, developers can address specific components without the need to navigate through an extensive monolithic configuration file.
- **b. Ease of collaboration:** in collaborative development environments, multiple teams or individuals may be responsible for different components of a big data pipeline. Using separated configuration files facilitates parallel development and reduces the risk of conflicts. Each team can work on their respective configurations independently, minimising the chances of unintentional interference.
- **c. Scalability:** as big data pipelines evolve and expand, the addition of new components or the modification of existing ones is inevitable. Separated configuration files accommodate this scalability seamlessly. Developers can introduce new configurations for new components without disrupting the settings of existing ones, promoting a scalable and extensible architecture. Furthermore, developers can override general settings parameters in the component specific settings for testing purposes.
- **d. Version control:** by organising configurations in a modular fashion, version control becomes more effective. Changes to specific components can be tracked independently, providing a clear audit trail of configuration modifications over time. This enhances traceability, simplifies debugging, and facilitates the rollback to previous configurations if needed.

5.1.2.5 LOGGING DESIGN

The logger is initialised after reading the configuration file. A single python logging object is created logging into the standard out (*stdout)* file descriptor. Furthermore, the software can also create a log file in the local filesystem of the master machine besides the writing to *stdout*. This functionality is activated via configuration and is recommended for local mono-cluster deployments. In cloud environments, like AWS EMR and GCP *Dataproc,* the logs written in *stdout* file descriptor can be saved into their respective blob storage.

Each component shall log the configuration that will use at the start of its execution in the log file.

5.1.3 INFRASTRUCTURE DESIGN

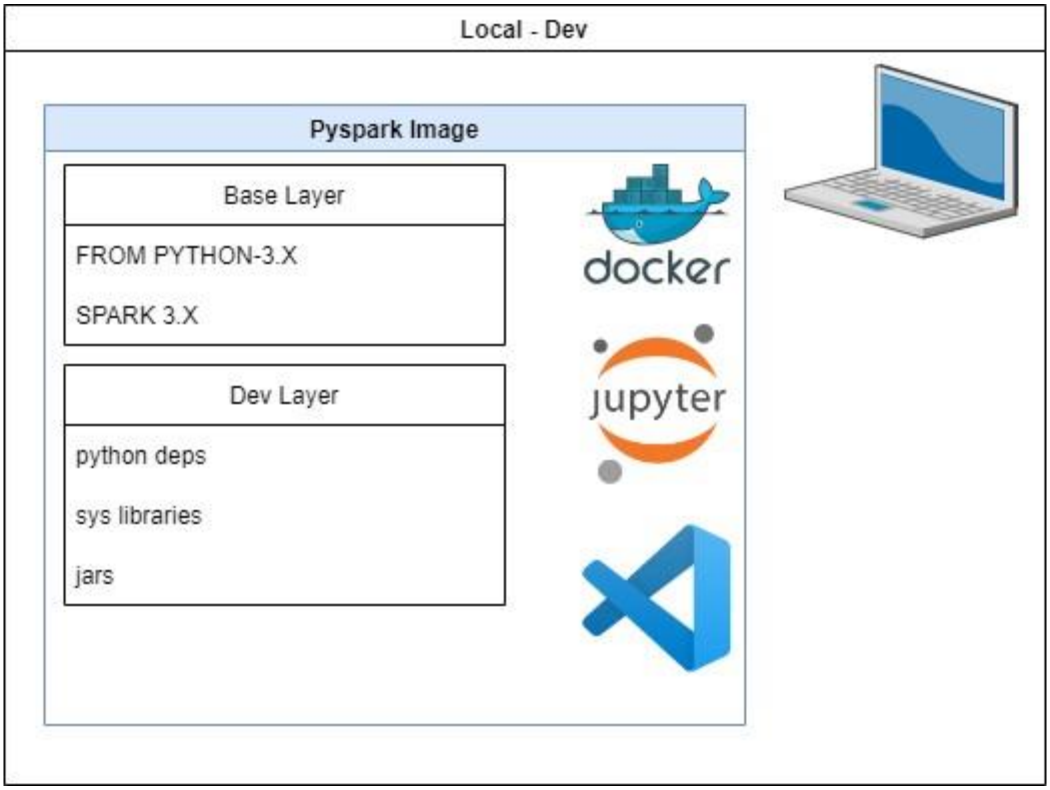
The software developed in the project processes big data pipelines which, due to the expected large volume of the data, distributed computing and distributed file systems frameworks will be used. Spark & HDFS, respectively, are the proposed open-source frameworks. They can be executed in centralised environments and distributed environments.

5.1.3.1 DEVELOPMENT/TESTING ENVIRONMENT

When developing the software, it is important that it can be executed locally so developers can perform an agile developing cycle. Thankfully both Spark & HDFS frameworks can be deployed locally in a centralised system of a single computer. However, in the case of a local execution, the OS filesystem can be used instead of HDFS for simplicity as the data used in this case should not be large.

For a local execution, docker technology is proposed as it allows to completely isolate software dependencies from the host machine inside the container. Containers can be conceived as virtual machines that only have the indispensable libraries for executing the software. With this approach, users only need to have docker installed in their system to execute the application. Furthermore, it allows to test different versions of libraries in an agile manner as multiple containers with different environments can be created in order to verify the software execution across multiple library versioning combination.

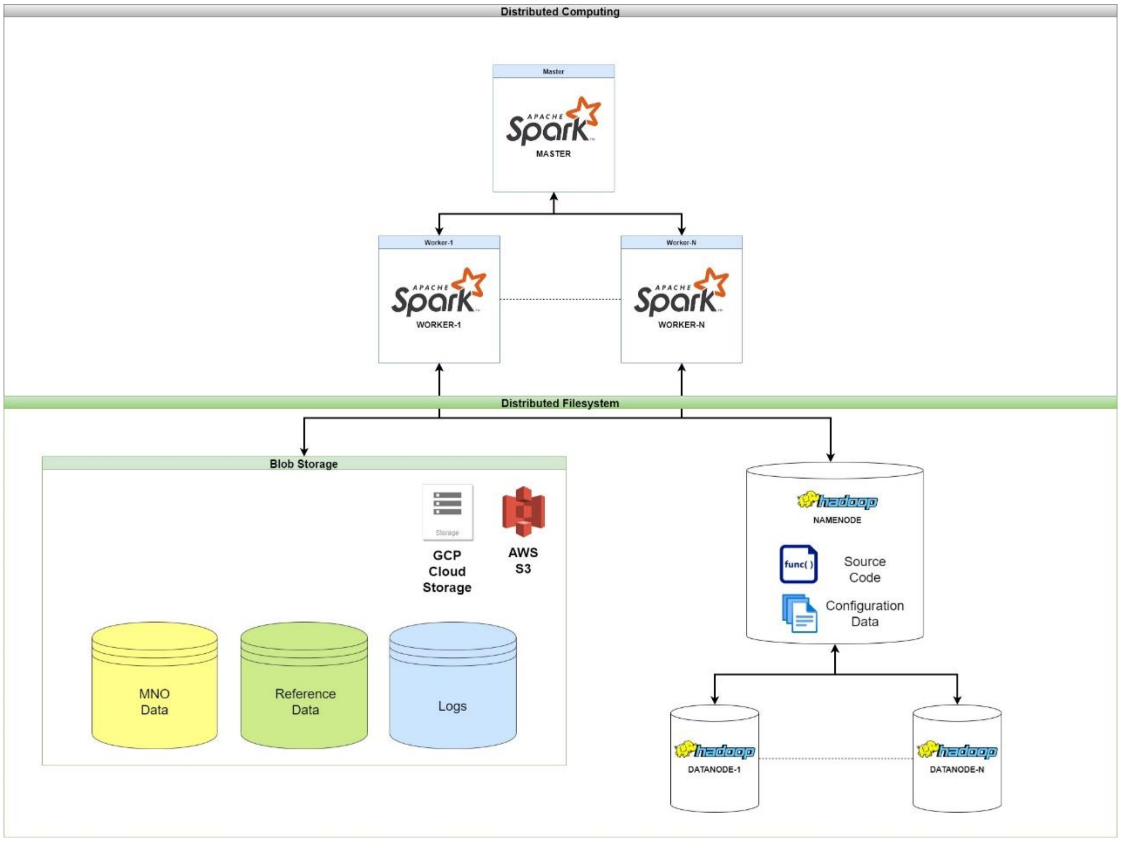
Figure 8: Standalone docker deployment



5.1.3.2 PRODUCTION ENVIRONMENT

Thanks to cloud providers like AWS and GCP, computing environments that easily scale can be deployed. Furthermore, the software needs to be executed in the MNO cloud environments due to data privacy constraints. Based on the project team previous experience working with MNOs, many of them usually provide a cloud-managed map-reduce cluster, such as AWS EMR or GCP Dataproc. These clusters follow a driver-executor architecture ideal for the Spark and Hadoop frameworks. In Figure 8, a representation of a distributed computing deployment in the MNO-Cloud is represented.

Figure 9: Distributed computing deployment



5.1.4 VERSION CONTROL

The code shall be maintained and released following the semantic versioning strategy. It consists on using three numeric levels for code versioning classified as **Major.Minor.Patch.** For example: 1.2.4.

Each level classifies the changes in the source code:

- **Major:** changes that are not backward compatible.
- **Minor:** changes that are backward compatible. Example: performance improvements or new functionalities.
- **Patches:** bug fixes.

Levels are increased sequentially by one. When a level increases it resets the value for levels to its right to zero (example: increasing the minor version for version 1.2.4. will result in 1.3.0.)

Generally, when the software is in development before its official first release (beta stage) the major version is set to zero (example: version 0.7.3.)

Pre-release metadata can be added to the version by appending a hyphen to the end (example: version 1.0.0-alpha1)

5.1.5 SOFTWARE ARTEFACTS DESIGN

5.1.5.1 SOURCE CODE

\ CODING STYLE

PEP 8, which stands for Python Enhancement Proposal 8, is the coding style proposed for writing clean, readable, and maintainable Python code. It was created to promote consistency in Python code and make it easier for developers to collaborate on projects. PEP 8 provides the following **advantages**:

1. **Readability**: PEP 8 enforces a consistent and easy-to-read coding style. This makes it easier for developers to understand and maintain the code, which is especially important for collaborative projects or when revisiting your own code in the future.
2. **Consistency**: PEP 8 helps ensure that Python code looks and feels consistent across different projects and teams. This consistency simplifies code reviews and reduces the learning curve when working on new projects.
3. **Collaboration**: when multiple developers work on a project, using PEP 8 ensures that everyone follows the same coding conventions. This can prevent misunderstandings and disagreements about coding style and improves code quality and maintainability.
4. **Tooling and automation**: many code editors and integrated development environments (IDEs) provide built-in or third-party tools for checking and formatting code according to PEP 8. These tools can automatically highlight or fix violations, making it easy to follow the style guide.
5. **Debugging**: code that follows PEP 8 is often easier to debug, as it has a consistent structure and naming conventions. This can save you time when troubleshooting issues.
6. **Community standards**: PEP 8 is widely accepted in the Python community, and most Python developers are familiar with its conventions. Adhering to PEP 8 makes it easier for you to collaborate with other developers and participate in open-source projects.
7. **Future-proofing**: following PEP 8 helps future-proof your code. As Python evolves, adhering to established coding standards makes it easier to update your code to newer Python versions and libraries.

PEP 8 official guide is available at the following link: https://peps.python.org/pep-0008/

\ DOCSTRING STYLE

Adhering to a unique docstring style guarantees consistency within software development in a project. Google Docstrings are the most popular convention for docstrings which facilitates readability and collaboration in open-source projects. Furthermore, Google Docstrings provide the following **benefits**:

1. **Clarity and readability:** Google Docstrings provide a structured format that includes sections for a function's description, parameters, return values, and examples. This format enhances the clarity and readability of your code documentation, making it easier for both developers and automated documentation tools to understand your code.
2. **Consistency:** Google Docstrings provide a consistent way to document your code. When multiple developers work on a project, using a standardized docstring format ensures that all functions and classes are documented in a similar and predictable way.
3. **Auto-generation:** many documentation tools and IDEs can parse Google Docstrings and automatically generate documentation from them. For example, tools like Sphinx and Doxygen can create HTML or PDF documentation from the source code docstrings.
4. **IDE support:** several Python Integrated Development Environments (IDEs), such as PyCharm and VSCode, can use Google Docstrings to provide code suggestions, autocompletion, and inline documentation. This can be a significant productivity boost for developers.

5. **API documentation:** Google Docstrings are suitable for generating API documentation. This makes it easier to extract and publish the project's API documentation for others to use.
6. **Help for code review:** when reviewing code, especially in a collaborative setting, well-documented functions with Google Docstrings can provide reviewers with a clear understanding of the purpose of each function and its expected inputs and outputs. This can lead to more effective code reviews.
7. **Self-documentation:** Google Docstrings serve as a form of self-documentation for your code. They provide valuable information about how to use the functions and classes without needing to dig into the implementation details.

Google Docstrings official guide is available in the following link:
https://google.github.io/styleguide/pyguide.html#38-comments-and-docstrings

5.1.5.2 TESTING

\ EXECUTION ARTEFACTS

- **Testing code**: code written in Python containing the unit-tests of the project. Each module should have, at least, a testing file with a battery of tests. Both 'happy path' and error cases should be tested. Tests should evolve as code evolves.
- **Testing resources:** files needed to execute all the unit-tests of the project. They can be divided in the following categories:

  - **Configuration:** configuration data needed for the execution of tests.
  - **Testing Data:** small input data and output expected data needed by tests.
  - **Automation scripts (optional):** scripts that execute the tests and generate the testing reports.

\ TESTING REPORT

Report containing the results of the testing execution. This report is automatically generated with the *automation scripts* mentioned in the previous section. It is defined as:

- **Test execution report:** file indicating which tests have been passed, failed or skipped. It should also include the version of the software used, test logs and execution time.

5.2 COMPONENT DESIGN

*[**Remark** - This section contains the design for the components available in the release 0.3 of the software.]*

5.2.1 EVENTCLEANING

5.2.1.1 MODULE DESCRIPTION

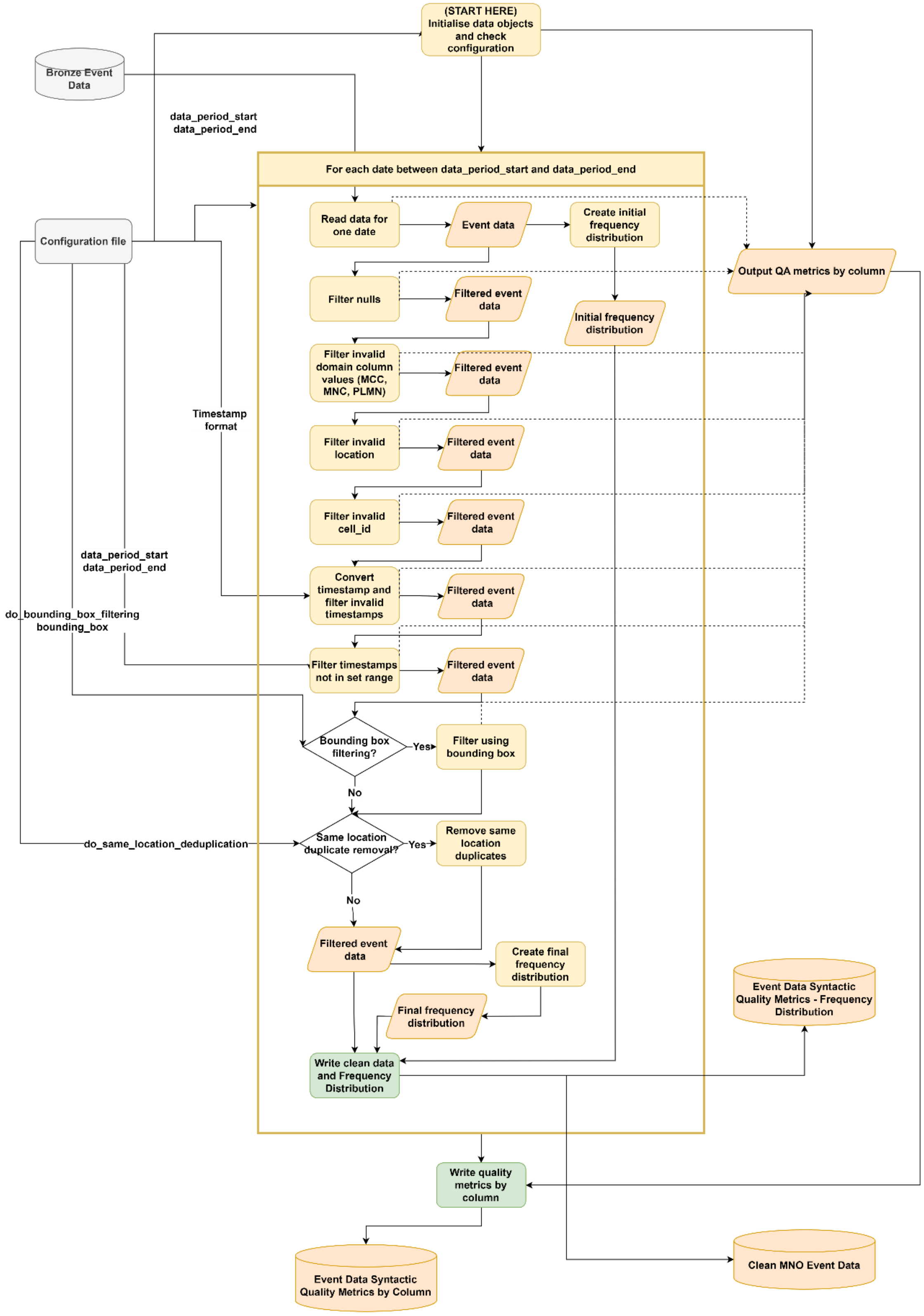
- **Module Name:** EventCleaning
- **Objectives:** the objective of this method is to perform syntactic checks on the raw event data from the MNO. Data not matching the expected syntax will be removed. Based on the removed records, quality metrics will be created.
- **Functionality:**
  Functionality details may be found in the software requirements: 3.2.6 EventCleaning
- **Data Inputs and Outputs:**
  - Input:
    I.1 MNO Event Data – Raw

- Outputs:
  I.2 MNO Event Data – Syntactically Cleaned
  I.4 MNO Event Data Syntactic Quality Metrics – frequency distribution
  I.3 MNO Event Data Syntactic Quality Metrics – by column

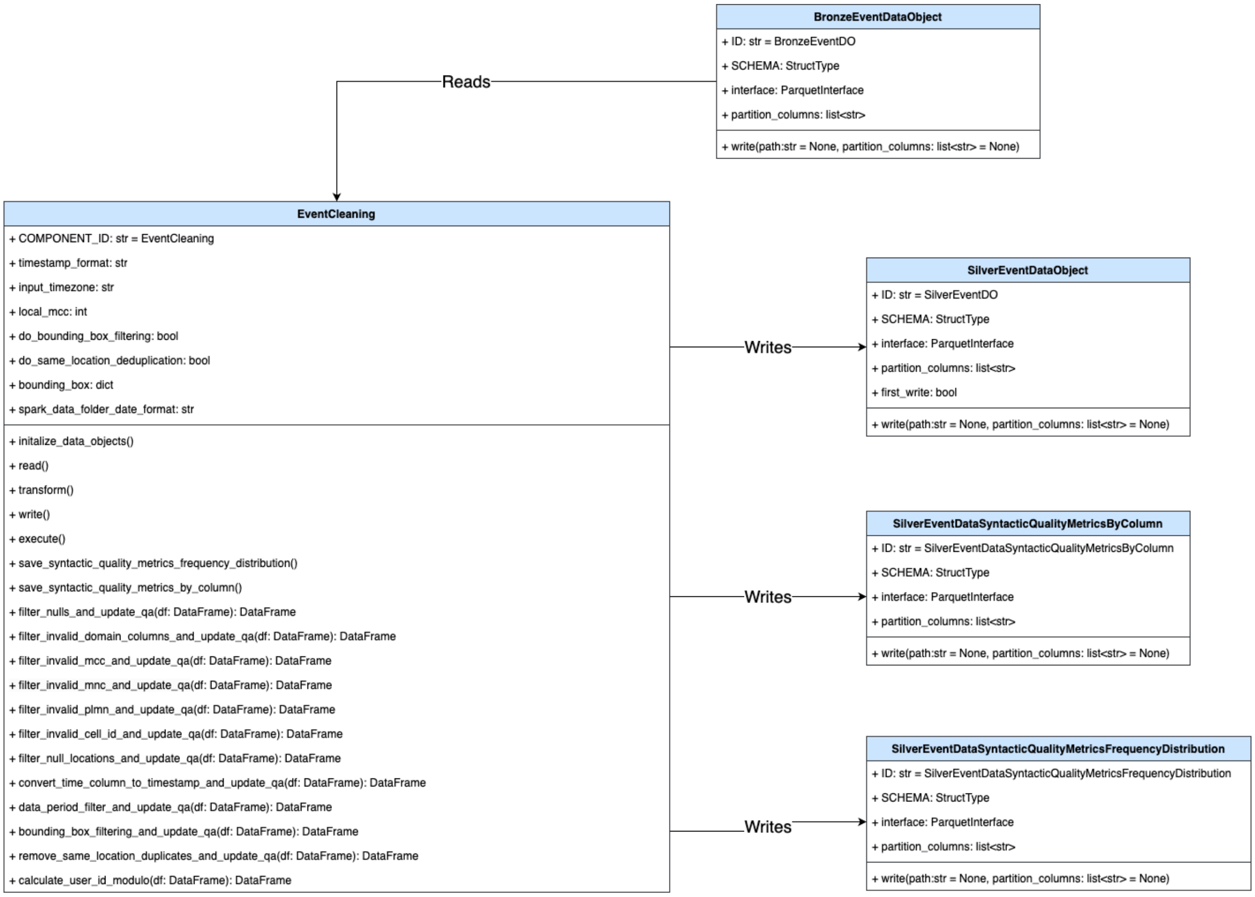
5.2.1.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
  The data is processed in one-day chunks. It is expected that the MNOs will provide data that is already separated by date. All of the following steps are run for each date in the data.
  - Create quality metrics data objects.
  - Filter out rows that contain nulls in user_id or timestamp columns and update quality metrics.
  - Filter out rows that do not have valid domain columns (each row must have mcc and mnc, or plmn).
  - Infer the domain of each row:
    - If plmn is not null - domain is outbound
    - If mcc is equal to local_mcc from configuration - domain is domestic
    - Otherwise - domain is inbound
  - Filter out domestic and inbound rows that have an invalid MCC code (has to be a number between 100 and 999) and update quality metrics.
  - Filter out domestic and inbound rows that have an invalid MNC code (has to be a number numerical with 2 or 3 digits, can also be 00) and update quality metrics.
  - Filter out outbound rows that have an invalid PLMN code (has to be a number between 10000 and 99900) and update quality metrics.
  - Filter out domestic and inbound rows that do not have a valid location. A row has to have a cell_id or both latitude and longitude columns as not nulls to be considered valid. Update quality metrics.
  - Filter out domestic and inboundrows with invalid cell_id. A valid cell_id contains 14 or 15 numerical digits. Update quality metrics.
  - Convert timestamp column to internal timestamp type according to timestamp_format from configuration and filter out rows where timestamp does not match the given format. Update quality metrics.
  - Filter out rows, where the timestamp is not between data_period_start and data_period_end. Update quality metrics.
  - If do_bounding_box_filtering is set to True in configuration:
    For rows with latitude and longitude: Filter out rows where value is out of bounds for bounding box defined by bounding_box in configuration. Update quality metrics
  - If do_same_location_duplicate_removal is set to True in configuration:
    Remove all rows that have identical values in the columns: timestamp, user_id, cell_id, latitude, longitude, plmn
    For rows with latitude and longitude: Filter out rows where value is out of bounds for bounding box defined by bounding_box in configuration. Update quality metrics
  - Calculate the modulo of the user_id column, to be used for partitioning the data, so that each partition would contain a similar number of users.
  - Write silver event data object and quality metrics

- **Data flow diagram:**


(START HERE)
Initialise data objects and check configuration
Bronze Event Data
data_period_start
data_period_end
For each date between data_period_start and data_period_end
Configuration file
Read data for one date
Event data
Create initial frequency distribution
Output QA metrics by column
Filter nulls
Filtered event data
Initial frequency distribution
Filter invalid domain column values (MCC, MNC, PLMN)
Filtered event data
Timestamp format
Filter invalid location
Filtered event data
Filter invalid cell_id
Filtered event data
data_period_start
data_period_end
Convert timestamp and filter invalid timestamps
Filtered event data
do_bounding_box_filtering
bounding_box
Filter timestamps not in set range
Filtered event data
Bounding box filtering?
Yes
Filter using bounding box
No
do_same_location_deduplication
Same location duplicate removal?
Yes
Remove same location duplicates
No
Filtered event data
Create final frequency distribution
Event Data Syntactic Quality Metrics - Frequency Distribution
Final frequency distribution
Write clean data and Frequency Distribution
Write quality metrics by column
Event Data Syntactic Quality Metrics by Column
Clean MNO Event Data

- **Class diagram:**


BronzeEventDataObject
+ ID: str = BronzeEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Reads
EventCleaning
+ COMPONENT_ID: str = EventCleaning
+ timestamp_format: str
+ input_timezone: str
+ local_mcc: int
+ do_bounding_box_filtering: bool
+ do_same_location_deduplication: bool
+ bounding_box: dict
+ spark_data_folder_date_format: str
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
+ save_syntactic_quality_metrics_frequency_distribution()
+ save_syntactic_quality_metrics_by_column()
+ filter_nulls_and_update_qa(df: DataFrame): DataFrame
+ filter_invalid_domain_columns_and_update_qa(df: DataFrame): DataFrame
+ filter_invalid_mcc_and_update_qa(df: DataFrame): DataFrame
+ filter_invalid_mnc_and_update_qa(df: DataFrame): DataFrame
+ filter_invalid_plmn_and_update_qa(df: DataFrame): DataFrame
+ filter_invalid_cell_id_and_update_qa(df: DataFrame): DataFrame
+ filter_null_locations_and_update_qa(df: DataFrame): DataFrame
+ convert_time_column_to_timestamp_and_update_qa(df: DataFrame): DataFrame
+ data_period_filter_and_update_qa(df: DataFrame): DataFrame
+ bounding_box_filtering_and_update_qa(df: DataFrame): DataFrame
+ remove_same_location_duplicates_and_update_qa(df: DataFrame): DataFrame
+ calculate_user_id_modulo(df: DataFrame): DataFrame
Writes
SilverEventDataObject
+ ID: str = SilverEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ first_write: bool
+ write(path:str = None, partition_columns: list<str> = None)
Writes
SilverEventDataSyntacticQualityMetricsByColumn
+ ID: str = SilverEventDataSyntacticQualityMetricsByColumn
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Writes
SilverEventDataSyntacticQualityMetricsFrequencyDistribution
+ ID: str = SilverEventDataSyntacticQualityMetricsFrequencyDistribution
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)

- **Code Structure:**
  The code structure follows the format set by the core package, and the general repository structure.

  The location of the module script in the repository is as follows:

```
/multimno_internal/
└── multimno
    └── components
        └── execution
            └── event_cleaning
                └── event_cleaning.py
```

event_cleaning.py contains one class named EventCleaning which is a subclass of Component.

The EventCleaning class overrides all methods in the Component class:

__init__ method initialises the data objects and reads the necessary values from config file.
read method is responsible for reading the data from one date into memory.
write method is responsible for writing event data and frequency distribution quality metrics for the date currently being processed.
transform performs all necessary filtering and transformations for daily data and updates the quality metrics data objects. transform contains calls to many other smaller functions that perform the actual data manipulation.
execute is responsible for calling read, write and transform for each unique date in the dataset. The processing is done date-by-date. Only the data from one date is being processed at any given time.

5.2.2 EVENTQUALITYWARNINGS

5.2.2.1 MODULE DESCRIPTION

- **Module Name:** EventQualityWarnings
- **Objectives:** the objective of this method is to create a flexible/dynamic tool that will compute Quality Warnings checks based on two outputs - Quality Metrics Frequency Distribution and Quality Metrics By Column. The flexibility is provided by the option of specifying what group of QWs to compute, what value for different thresholds to choose and most importantly it is able to compute Quality Warnings after both MNO Event Cleaning and Event Deduplication stages meaning that this component does the job of Event Data Syntactic Quality Warnings and Event Deduplication Quality Warnings. The component is supposed to write two Data Objects - Log Table with unified structure of representing errors' information and For Plots which stores data needed to create graphs of three variables' distribution along with some other statistical measures - initial frequency, total frequency, and error rate by date.
- **Functionality:**
  the process of managing Quality Warnings is segmented into three major categories: QWs related to the daily sizes of data (both raw and preprocessed); the error rate of event data across various granularity levels (by date, by date and cell_id, by date and user_id, by date and cell_id and user_id); and quality assessments of error types (missing values, values out-of-range, deduplication of identical locations, and etc.).
  Functionality details may be found in the software requirements: 3.2.7 EventQualityWarnings
- **Data Inputs and Outputs:**
  - Input:
    In both Quality Warnings cases the Component expects two inputs: Event Data Quality Metrics Frequency Distribution and Event Data Quality Metrics By Column
    - I.3 MNO Event Data Syntactic Quality Metrics – by column
    - I.4 MNO Event Data Syntactic Quality Metrics – frequency distribution
  - Output:
    - I.5 MNO Event Data Quality Warnings – log table
    - I.22 MNO Event Data Quality Warnings – for plots

5.2.2.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
  The whole Component heavily relies on the input Quality Metrics. It is important to specify the correct time period for Quality Warnings taking into account the lookback period timeline, in order to have covered enough previous data for the calculation of the Quality Warnings. For example, if the Quality Metrics were computed for the period of [2023-01-01, 2023-01-15] and a lookback period is set to be a week (7 days) then the first date of the time reference/period for Quality Warnings would be 2023-01-

08, and the end date should be any date later than the start date and earlier than or the same as the last date of the calculated Quality Metrics (i.e. 2023-01-15 in the given example).

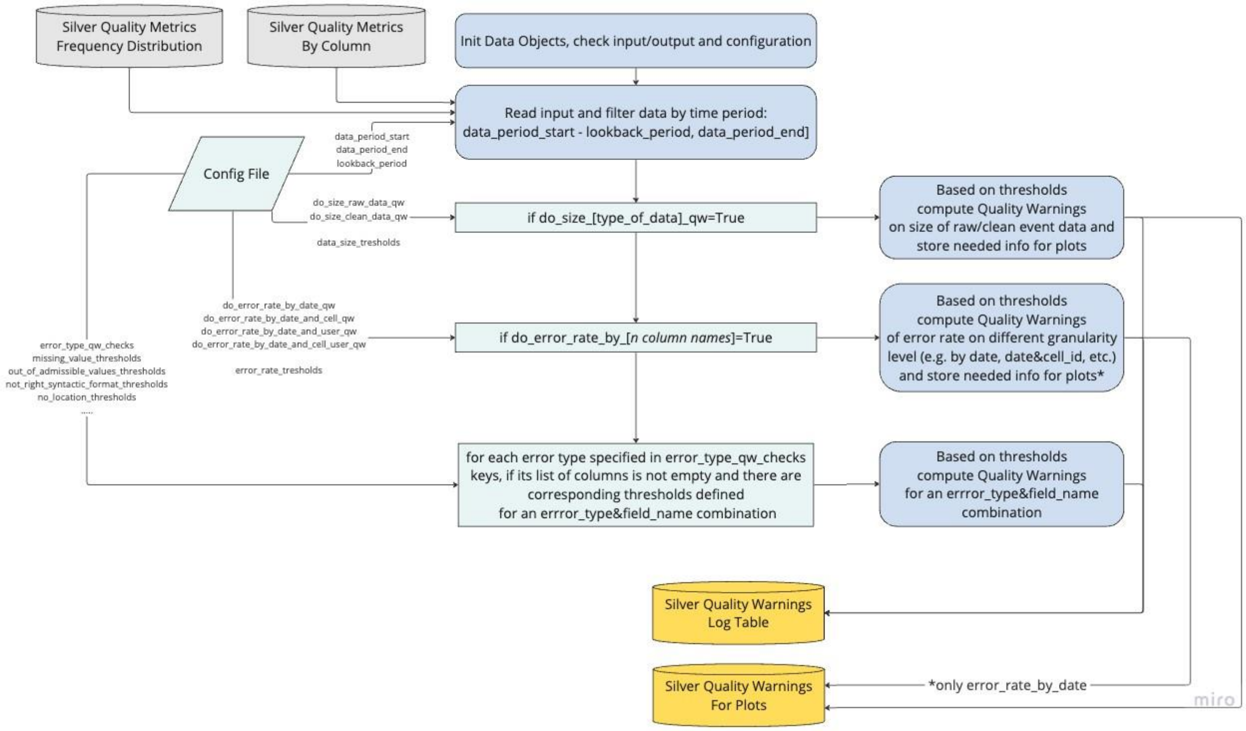
The whole process of execution of Quality Warnings is divided into three large groups: Quality Warnings regarding daily size of data; error rate of event data on different granularity levels; and quality checks of error types (e.g. missing value, out-of-range, deduplication same locations). The first two groups (size and error rate) solely use Silver Event Data Syntactic Quality Metrics - Frequency Distribution object and each sub-Quality-Warning within the mentioned sets is invoked by boolean value (basically, if True do something). The last group requires two inputs Frequency Distribution and Silver Event Data Syntactic Quality Metrics - By Column although the later holds the most important information. Also, the logic of this group differs; namely, the algorithm loops through each unique combination of error_type&field_name and performs same types of Quality Warnings, meaning the input is changing (e.g. number of errors for combination missing_value&user_id, or out_of_range&mcc), while the Quality Warnings process stays the same. The Component description step-by-step is presented below:

- Initialise EventQualityWarnings Component. Create attributes based on corresponding config (cleaning and deduplication have their own, separate config), check existence of input, initialise corresponding output Data Objects, if clear_destination_directory clear all Component's output
- Read Quality Metrics for the period: [data_period_start - lookback_period, data_period_end]. In config lookback_period is specified as string but in component it gets numerical representation.
- **Perform Quality Warning regarding data size (regards only Event Cleaning Quality Warnings)**, it could be either raw size (initial frequency) or clean size (final frequency), both are run if their corresponding config boolean params (do_size_raw_data_qw and do_size_clean_data_qw) are set to True. The QW involved: checking if a size within a range of two absolute numbers (upper and lower limit) and between [mean+X*std, mean-X*std] boundaries, average and standard deviation are calculated based on previous data of lookback_period length. Correspondingly three configurable thresholds (for each type of size) are involved: absolute upper/lower limits and the number of stds appropriate to deviate from mean. The information of wrong entries (please refer to its structure in Methodology Section) is stored in Log Table and apart from that data to plot graphs is being calculated. Important to mention that for Log Table the period would be as specified in config [data_period_start, data_period_end], while For Plots it should be [data_period_start - lookback_period, data_period_end].
- **Perform error rate Quality Warnings (regards only MNO Event Cleaning Quality Warnings)** which is computed by formula: **Error rate** = (Total initial frequency - Total final frequency) / Total initial frequency*100. The error rate is then checked on three warnings: should not be higher than some absolute number; should not be higher than average of previous error rates by some X%, should not be higher than mean + X*std. Again, for average and standard deviation information of previous days is used. The error rate Quality Warnings are computed on different granularity level (by date, by date and cell_id, by date and user_id, by date and cell_id and user_id). Each warning for each granularity level has their own configurable thresholds. The decision on running error rate Quality Warnings on each level is decided by its own boolean config param (e.g. do_error_rate_by_date_qw, do_error_rate_by_date_and_cell_qw, and so on). The information of wrong entries is stored in Log Table and apart from that data to plot graphs is being calculated but only for error rate by date.
- Perform final set of QWs - **error type Quality Warnings (regards both MNO Event Cleaning Quality Warnings and Event Deduplication Quality Warnings)**. They have the same checks as for error rate Quality Warnings - absolute upper limit, not over the average by X%, and

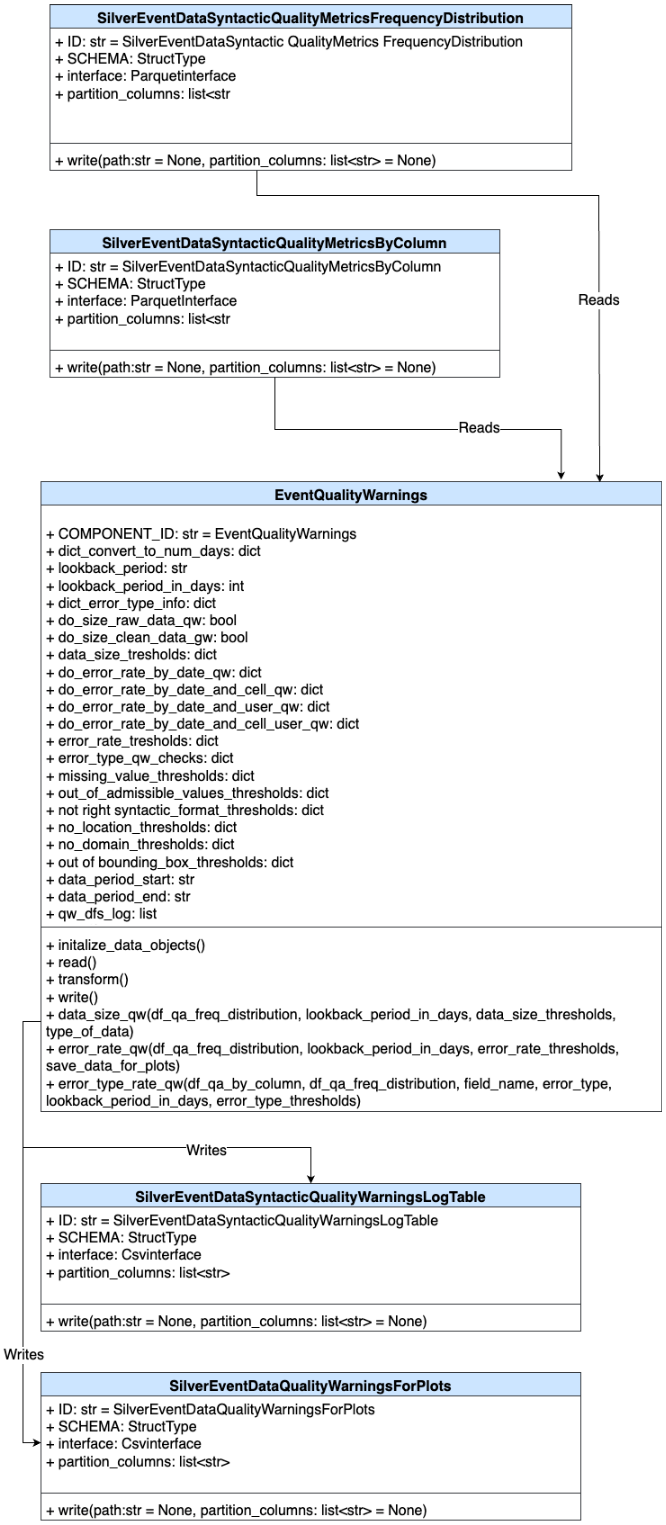
mean+X*std limit (and under the hood uses the same function as for error rate QWs). However, the logic of invoking these checks is different instead of using boolean value, the code does it dynamically looping through unique combination of error_type&field_name (one error type can have many field names) which is specified in a config param: error_type_qw_checks . Each stated error_type should have another config param to define thresholds for its field_names (e.g. missing_value_thresholds), for a more detailed description please refer to configuration. The information of wrong entries is stored in Log Table, no data for plots is saved.

- Write silver output data objects of the Component: SilverEventDataSyntacticQualityWarningsLogTable and SilverEventDataSyntacticQualityWarningsForPlots as CSVs partitioned by date

- **Data flow diagram:**

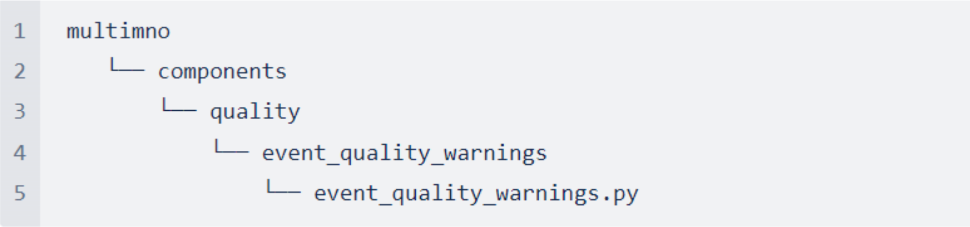

Silver Quality Metrics Frequency Distribution
Silver Quality Metrics By Column
Init Data Objects, check input/output and configuration
Read input and filter data by time period: data_period_start - lookback_period, data_period_end]
data_period_start
data_period_end
lookback_period
Config File
do_size_raw_data_qw
do_size_clean_data_qw
data_size_tresholds
if do_size_[type_of_data]_qw=True
Based on thresholds compute Quality Warnings on size of raw/clean event data and store needed info for plots
do_error_rate_by_date_qw
do_error_rate_by_date_and_cell_qw
do_error_rate_by_date_and_user_qw
do_error_rate_by_date_and_cell_user_qw
error_rate_tresholds
if do_error_rate_by_[n column names]=True
Based on thresholds compute Quality Warnings of error rate on different granularity level (e.g. by date, date&cell_id, etc.) and store needed info for plots*
error_type_qw_checks
missing_value_thresholds
out_of_admissible_values_thresholds
not_right_syntactic_format_thresholds
no_location_thresholds
.....
for each error type specified in error_type_qw_checks keys, if its list of columns is not empty and there are corresponding thresholds defined for an errror_type&field_name combination
Based on thresholds compute Quality Warnings for an errror_type&field_name combination
Silver Quality Warnings Log Table
Silver Quality Warnings For Plots
*only error_rate_by_date
miro

- **Class diagram:**


SilverEventDataSyntacticQualityMetricsFrequencyDistribution
+ ID: str = SilverEventDataSyntactic QualityMetrics FrequencyDistribution
+ SCHEMA: StructType
+ interface: Parquetinterface
+ partition_columns: list<str
+ write(path:str = None, partition_columns: list<str> = None)
Reads
SilverEventDataSyntacticQualityMetricsByColumn
+ ID: str = SilverEventDataSyntacticQualityMetricsByColumn
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str
+ write(path:str = None, partition_columns: list<str> = None)
Reads
EventQualityWarnings
+ COMPONENT_ID: str = EventQualityWarnings
+ dict_convert_to_num_days: dict
+ lookback_period: str
+ lookback_period_in_days: int
+ dict_error_type_info: dict
+ do_size_raw_data_qw: bool
+ do_size_clean_data_gw: bool
+ data_size_tresholds: dict
+ do_error_rate_by_date_qw: dict
+ do_error_rate_by_date_and_cell_qw: dict
+ do_error_rate_by_date_and_user_qw: dict
+ do_error_rate_by_date_and_cell_user_qw: dict
+ error_rate_tresholds: dict
+ error_type_qw_checks: dict
+ missing_value_thresholds: dict
+ out_of_admissible_values_thresholds: dict
+ not right syntactic_format_thresholds: dict
+ no_location_thresholds: dict
+ no_domain_thresholds: dict
+ out of bounding_box_thresholds: dict
+ data_period_start: str
+ data_period_end: str
+ qw_dfs_log: list
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ data_size_qw(df_qa_freq_distribution, lookback_period_in_days, data_size_thresholds, type_of_data)
+ error_rate_qw(df_qa_freq_distribution, lookback_period_in_days, error_rate_thresholds, save_data_for_plots)
+ error_type_rate_qw(df_qa_by_column, df_qa_freq_distribution, field_name, error_type, lookback_period_in_days, error_type_thresholds)
Writes
SilverEventDataSyntacticQualityWarningsLogTable
+ ID: str = SilverEventDataSyntacticQualityWarningsLogTable
+ SCHEMA: StructType
+ interface: Csvinterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Writes
SilverEventDataQualityWarningsForPlots
+ ID: str = SilverEventDataQualityWarningsForPlots
+ SCHEMA: StructType
+ interface: Csvinterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)

- **Code Structure:**
  The code structure follows the format set by the core package, and the general repository structure.

  The location of the module script in the repository is as follows:



```
multimno
    └── components
          └── quality
                └── event_quality_warnings
                      └── event_quality_warnings.py
```

`event_quality_warnings.py` contains one class named `EventQualityWarnings` which is a subclass of `Component`.

The `EventQualityWarnings` class overwrites all methods in the `Component` class:

`__init__` method initialises the data objects and reads the necessary values from config file.
`read` method is responsible for reading the data from Event Quality Metrics
`write` method is responsible for writing outputs of the component.

`transform` performs all specified Quality Warning checks
`execute` is responsible for calling `read`, `write` and `transform`

5.2.3 EVENTDEDUPLICATION

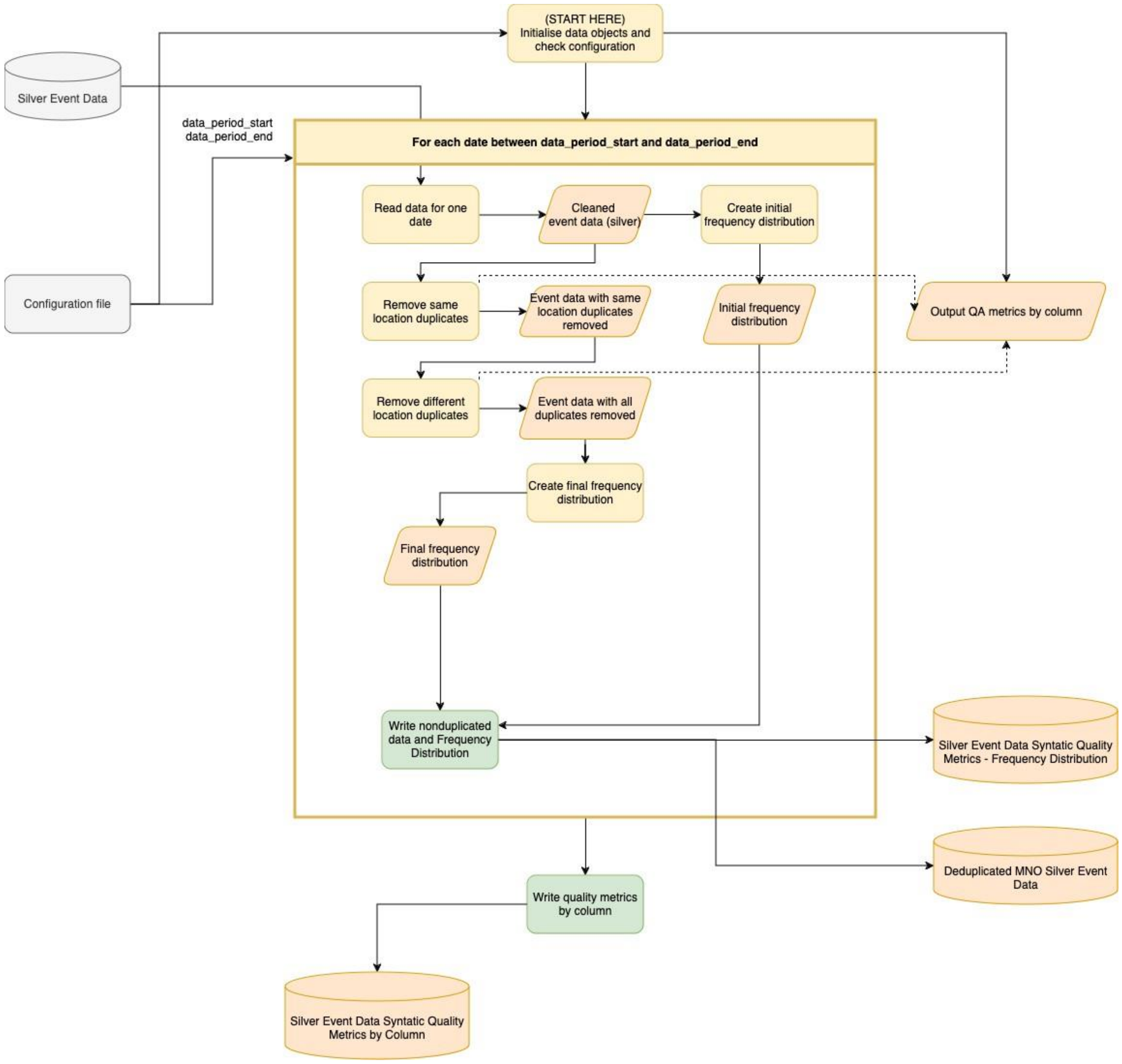
5.2.3.1 MODULE DESCRIPTION

- **Module Name:** EventDeduplication
- **Objectives:** the objective of the method is to process event data, so that duplicate records are removed, and to create quality metrics based on detected duplicated rows.
  These quality metrics follow the standard structure of syntactic quality metrics, and include variables such as initial frequency, total frequency, and error rate by date.
- **Functionality:** the Event Deduplication module retrieves and removes the duplicated records in the device level event data. It distinguishes between same and different location duplicates. It produces frequency and column-wise statistics for removed duplicates. Quality metrics produced per column are produces for each date in the configured period.
  Functionality details may be found in the software requirements: 3.2.8 EventDeduplication
- **Data Inputs and Outputs:**
  - Input:
    - I.2 MNO Event Data – Syntactically Cleaned
  - Output:
    - I.3 MNO Event Data Syntactic Quality Metrics – by column
    - I.4 MNO Event Data Syntactic Quality Metrics – frequency distribution
    - I.6 MNO Event Data – Deduplicated

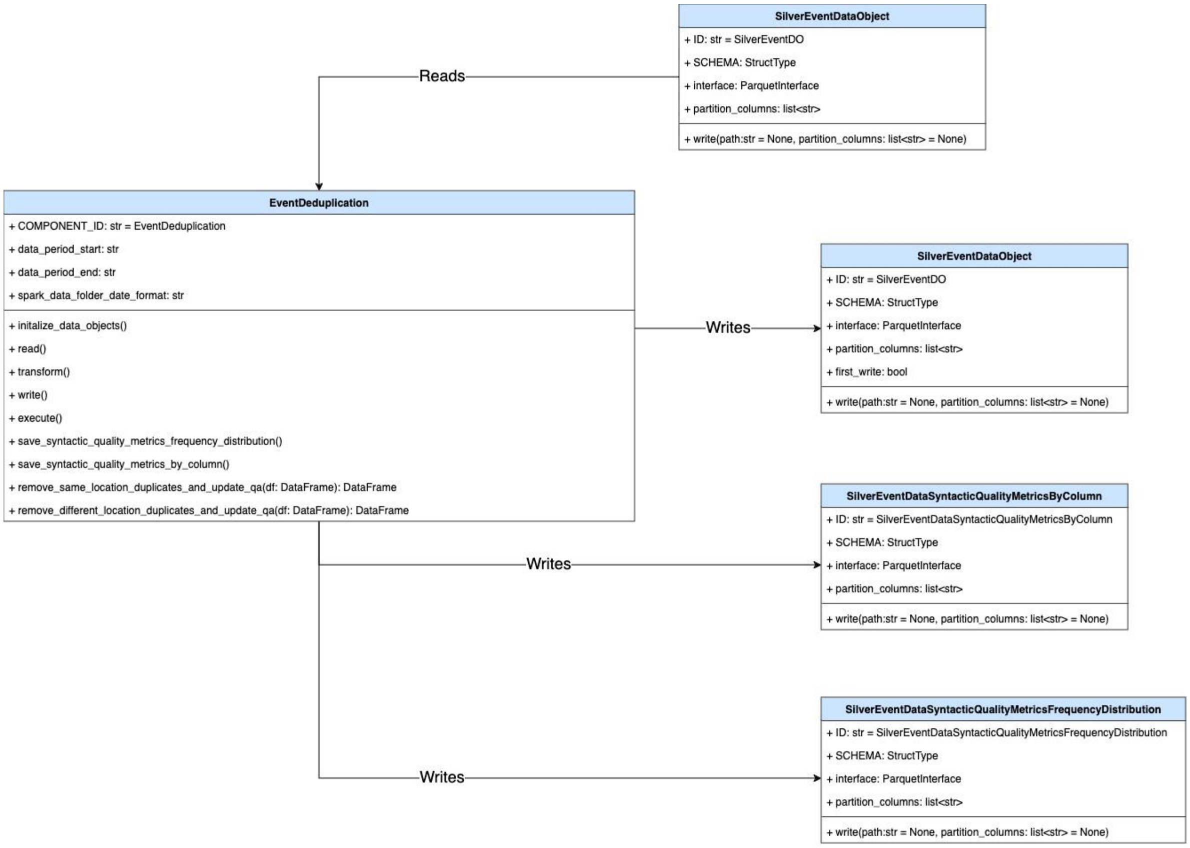
5.2.3.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
  - Iteration over selected dates. All the processes described below are performed on a date bases, at each level of iteration.
  - Reading in event data for all subscribers for a given date.
  - Processing the data.
  - Selection of one row in cases of same location duplicates. These are duplicate rows that have identical location information (cell_id, longitude and latitude) and timestamp information. As the rows are identical, only one row is kept.
  - Removal of different location duplicates from the data. These are duplicate rows that have identical location information timestamp information but may have different location information. All rows in cases of these duplicates are removed.
  - Counting the number of rows that have been changed by either of the two duplicate removal techniques.
  - Calculating the initial frequency before and after duplicate removal.
  - Writing deduplicated quality metrics per column.
  - Writing deduplicated records.
  - Writing silver output data objects of the Component: SilverEventDataSyntacticQualityMetricsByColumn considering the error codes for deduplication.

- **Data flow diagram:**

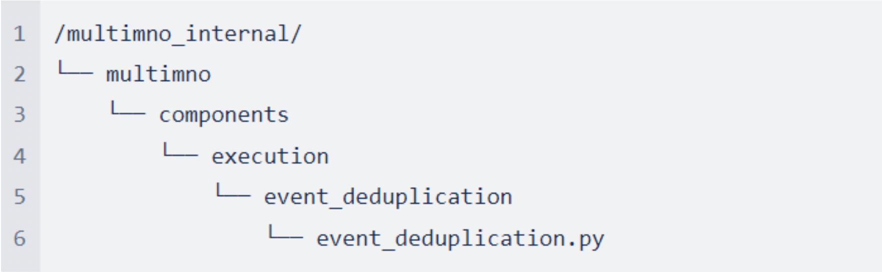

(START HERE)
Initialise data objects and check configuration
Silver Event Data
data_period_start
data_period_end
Configuration file
For each date between data_period_start and data_period_end
Read data for one date
Cleaned event data (silver)
Create initial frequency distribution
Remove same location duplicates
Event data with same location duplicates removed
Initial frequency distribution
Output QA metrics by column
Remove different location duplicates
Event data with all duplicates removed
Create final frequency distribution
Final frequency distribution
Write nonduplicated data and Frequency Distribution
Silver Event Data Syntatic Quality Metrics - Frequency Distribution
Deduplicated MNO Silver Event Data
Write quality metrics by column
Silver Event Data Syntatic Quality Metrics by Column

- **Class diagram:**


SilverEventDataObject
+ ID: str = SilverEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Reads
EventDeduplication
+ COMPONENT_ID: str = EventDeduplication
+ data_period_start: str
+ data_period_end: str
+ spark_data_folder_date_format: str
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
+ save_syntactic_quality_metrics_frequency_distribution()
+ save_syntactic_quality_metrics_by_column()
+ remove_same_location_duplicates_and_update_qa(df: DataFrame): DataFrame
+ remove_different_location_duplicates_and_update_qa(df: DataFrame): DataFrame
Writes
SilverEventDataObject
+ ID: str = SilverEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ first_write: bool
+ write(path:str = None, partition_columns: list<str> = None)
Writes
SilverEventDataSyntacticQualityMetricsByColumn
+ ID: str = SilverEventDataSyntacticQualityMetricsByColumn
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Writes
SilverEventDataSyntacticQualityMetricsFrequencyDistribution
+ ID: str = SilverEventDataSyntacticQualityMetricsFrequencyDistribution
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)

- **Code Structure:**
  The code structure follows the format set by the core package, and the general repository structure.

  The location of the module script in the repository is as follows:



```
/multimno_internal/
└── multimno
    └── components
        └── execution
            └── event_deduplication
                └── event_deduplication.py
```

`event_deduplication.py` contains one class named `EventDeduplication` which is a subclass of `Component`.

The `EventDeduplication` class overwrites all methods in the `Component` class:

`__init__` method initialises the data objects and reads the necessary values from config file.
`read` method is responsible for reading the data from Event Quality Metrics
`write` method is responsible for writing outputs of the component.

`transform` performs all specified Quality Warning checks
`execute` is responsible for calling `read`, `write` and `transform`

5.2.4 NETWORKCLEANING

5.2.4.1 MODULE DESCRIPTION

- **Module Name:** NetworkCleaning
- **Objectives:** this module is responsible for performing syntax checks on Network Topology Data to remove erroneous entries and to produce corresponding syntax quality metrics.
- **Functionality:** this module finds and removes entries where one field presents one of the following errors (when applicable): missing or null value, cannot be parsed, and out-of-range value. It also counts the number of errors before and after performing these checks, as well as the number of times each type of error appeared in each field.
  Functionality details may be found in the software requirements: 3.2.1 NetworkCleaning
  At this moment in time, only the processing of cell locations with physical properties is implemented.
- **Data Inputs and Outputs:**
  - Input:
    - I.7 Cell Locations with Physical Properties - Raw
  - Output
    - I.8 Cell Locations with Physical Properties – Cleaned
    - I.9 MNO Network Topology Data Quality Metrics
    - I.26 MNO Network Topology Top Frequent Erros
    - I.27 MNO Network Topology Row Error Metrics

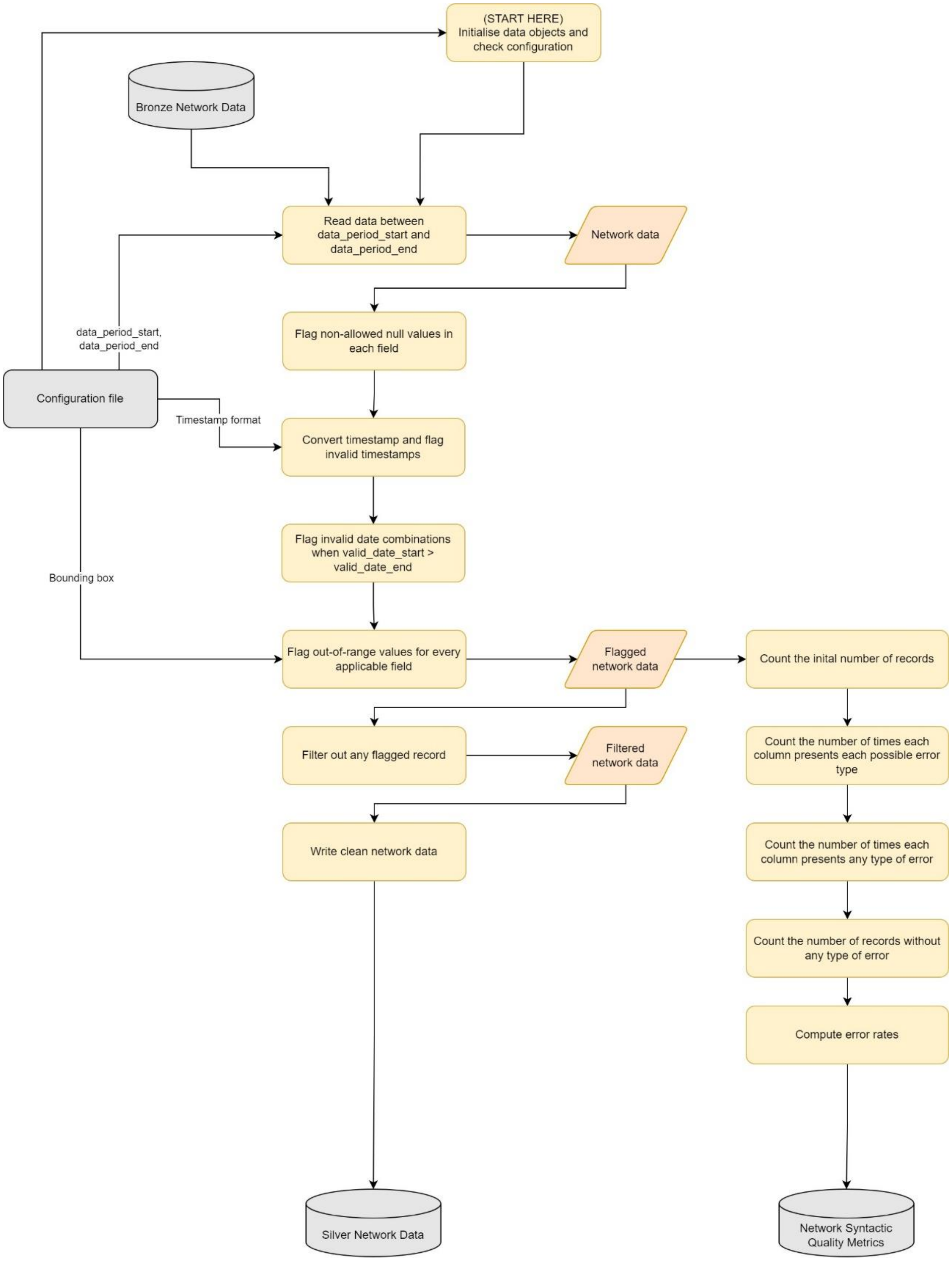
5.2.4.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:** The raw data is processed for the set of dates within an date interval specified via configuration file. It is assumed that the raw input data is partitioned by *year, month, day* columns, and the processes below work separately on each date's data.
  - Create quality metrics data object.
  - Filter out rows outside of the date interval.
  - Create an empty list `auxiliar_columns` that will contain the names of auxiliar columns that will keep track of each possible type of error in each field of a row.
  - Missing values:
    - Create one boolean column per field name, and set it to True if, for a given row, the value in a specific field is null.
      - Exception: `azimuth_angle` is expected to be null when `directionality` is equal to 0 – this is not computed as a 'mising'-null value and will not be computed as such for the corresponding quality metric later on.
      - Note that `valid_date_end` is allowed to be null. Nevertheless, the null values of this column will be checked and reflected in the corresponding boolean column.
    - Add all these columns names to `auxiliar_columns`.
  - Parsing errors:
    - Only `valid_date_start` and `valid_date_end` can have this type of error.
    - Parse the above-mentioned columns from string type to timestamp type.

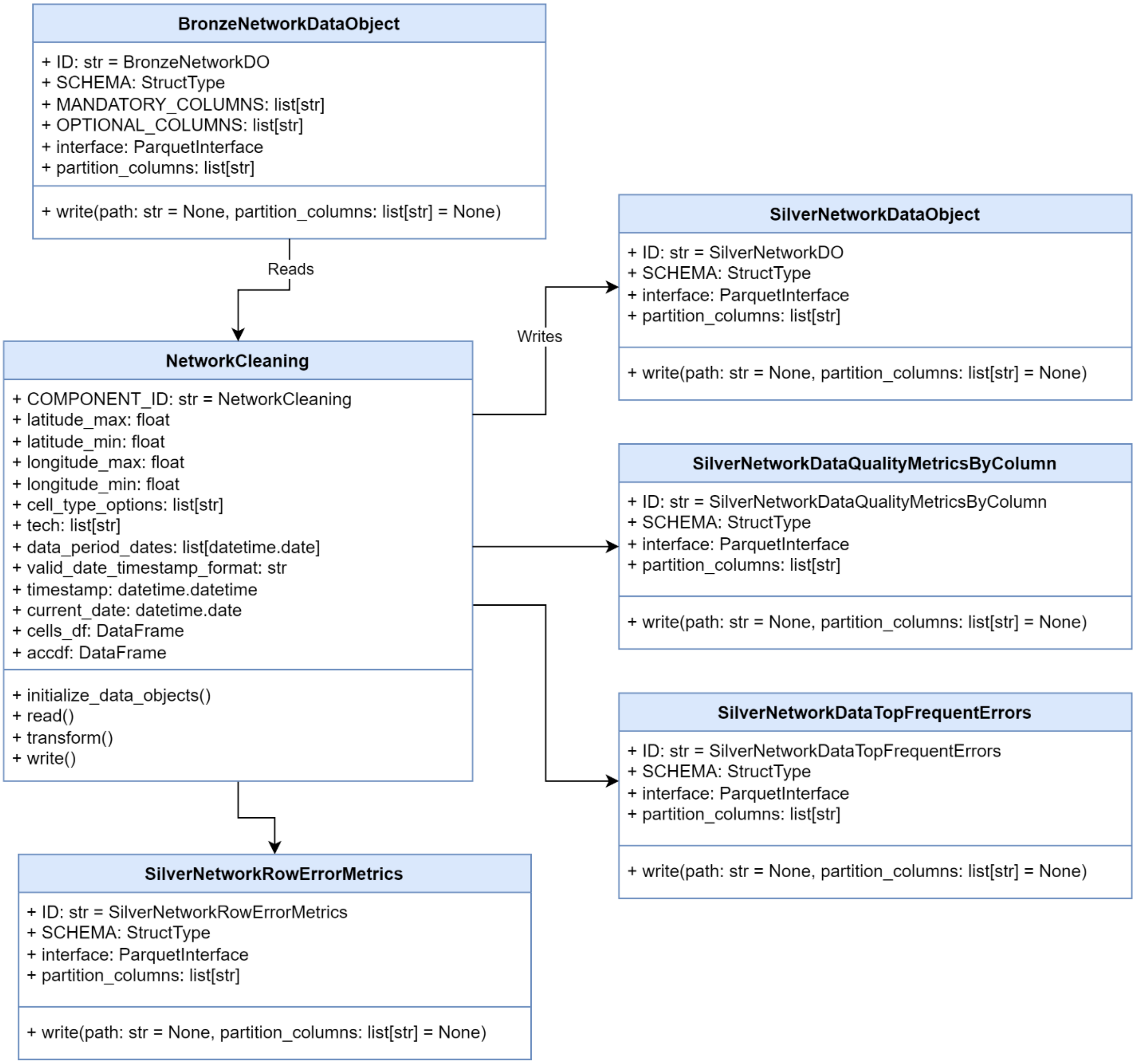
  - When a non-null value cannot be parsed with the specified timestamp format, the function employed, `pyspark.sql.functions.to_timestamp`, returns a null value.
  - Create one boolean column for each of these two fields and set it to True if the original value is non-null and the parsing failed.
  - Add the two column names to `auxiliar_columns`.
- <u>Out-of-range/out-of-bounds values</u>:
  - First, check for incoherent dates: this occurs when `valid_date_start` is a later point in time than `valid_date_end`.
    - Create a new boolean column equal to True when both `valid_date_start` and `valid_date_end` are not null, and `valid_date_start > valid_date_end`.
    - Add the new column's name to `auxiliar_columns`.
  - Now, check of out-of-range values for the rest of the variables. A new boolean column will be created for each of them, True when the value is out-of-range, and their names are added to `auxiliar_columns`.
    - `cell_id`: check if the string has a length different from 14 or 15 characters (to be improved to check for CGI/eCGI rules).
    - `latitude`: check if the value is outside a configuration-specified interval.
    - `longitude`: check if the value is outside a configuration-specified interval.
    - `antenna_height`: check if the value is less or equal to 0 (i.e., non-positive).
    - `directionality`: check if the value is not equal to 0 or to 1.
    - `azimuth_angle`: whenever directionality is equal to 1, check if the value is lower than 0 or higher than 360.
    - `elevation_angle`: check if the value is lower than -90 or higher than 90.
    - `horizontal_beam_width`: check if the value is lower than 0 or higher than 360.
    - `vertical_beam_width`: check if the value is lower than 0 or higher than 360.
    - `power`: check if the value is lower than 0.
    - `range`: check if the value is lower than 0.
    - `frequency`: check if the value is lower than 0.
    - `technology`: check if the value is not one of '5G', 'LTE', 'UMTS', and 'GSM'.
    - `cell_type`: check if the value is not one of the possible admitted values specified via configuration file.
- For each field, create a new boolean column and set it to True if, for a given row, the field does not have any type of error. Add these column names to `auxiliar_columns`.
  - Exception: the column `valid_date_end` is allowed to have null values. Thus, the null-boolean-column corresponding to `valid_date_end` is not considered for the computation of this boolean column.
- Create a new boolean column to_preserve and set it to True if a given row does not have any type of error in any of its fields.
  - Exception: the column `valid_date_end` is allowed to have null values. Thus, the null-boolean-column corresponding to `valid_date_end` is not considered for the computation of this boolean column.
- Compute quality metrics:

    - Count the number of True values in each of the columns whose names are stored in `auxiliar_columns`. These are the error-related quality metrics for each date considered.
    - For each date, count the number of rows present in the raw input data.
    - Count the number of times that the True value appears in the `to_preserve` column for each date considered. These are the number of rows present after the syntactic checks are performed.
  - Compute row error metrics:
    - Count the total number of rows that are going to be deleted, that is, any row that has any type of error in any of its mandatory fields.
    - Count the total number of rows that have at least one type of error in any of its fields, irrespective of whether that field is mandatory or optional.
  - Compute top frequent invalid values: the absolute frequency of each invalid value in a given field is computed. Ordered from most to least frequent, the accumulated sum of percentage of each error with respect to the total number of errors is calculated. Based on what is requested via configuration file, this information is saved in two different ways:
    - If the top *k* most frequent invalid values were requested as an absolute number, the *k* most frequent combinations of field and invalid value are saved.
    - If the topmost frequent invalid values were requested as a percentage number *k*, the most frequent combinations of field and invalid value that cover at least *k* percentage of all invalid values are saved.
  - Filter out rows with the column `to_preserve` as a mask, select only the original columns (so no `auxiliar_column` or `to_preserve` is kept), and save the result. This is the clean dataset after syntactic checks.

- **Data flow diagram:**


(START HERE)
Initialise data objects and check configuration
Bronze Network Data
Read data between data_period_start and data_period_end
Network data
data_period_start, data_period_end
Flag non-allowed null values in each field
Configuration file
Timestamp format
Convert timestamp and flag invalid timestamps
Flag invalid date combinations when valid_date_start > valid_date_end
Bounding box
Flag out-of-range values for every applicable field
Flagged network data
Count the inital number of records
Filter out any flagged record
Filtered network data
Count the number of times each column presents each possible error type
Write clean network data
Count the number of times each column presents any type of error
Count the number of records without any type of error
Compute error rates
Silver Network Data
Network Syntactic Quality Metrics

- **Class diagram:**


BronzeNetworkDataObject
+ ID: str = BronzeNetworkDO
+ SCHEMA: StructType
+ MANDATORY_COLUMNS: list[str]
+ OPTIONAL_COLUMNS: list[str]
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
Reads
Writes
SilverNetworkDataObject
+ ID: str = SilverNetworkDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
NetworkCleaning
+ COMPONENT_ID: str = NetworkCleaning
+ latitude_max: float
+ latitude_min: float
+ longitude_max: float
+ longitude_min: float
+ cell_type_options: list[str]
+ tech: list[str]
+ data_period_dates: list[datetime.date]
+ valid_date_timestamp_format: str
+ timestamp: datetime.datetime
+ current_date: datetime.date
+ cells_df: DataFrame
+ accdf: DataFrame
+ initialize_data_objects()
+ read()
+ transform()
+ write()
SilverNetworkDataQualityMetricsByColumn
+ ID: str = SilverNetworkDataQualityMetricsByColumn
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
SilverNetworkDataTopFrequentErrors
+ ID: str = SilverNetworkDataTopFrequentErrors
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
SilverNetworkRowErrorMetrics
+ ID: str = SilverNetworkRowErrorMetrics
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
/multimno/
└── components
    └── execution
        └── network_cleaning
            └── network_cleaning.py
```

- `network_cleaning.py` contains one class named `NetworkCleaning` which is a subclass of `Component`. The `NetworkCleaning` class overrides some of the methods of `Component`:
    - The `__init__` method first call its parent's `__init__` method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - `Transform` performs all necessary filtering and transformations pertaining to the syntactic checks for daily raw network data and computes the associated updates the quality metrics data objects.

5.2.5 NETWORKQUALITYWARNINGS

5.2.5.1 MODULE DESCRIPTION

- **Module Name:** NetworkQualityWarnings
- **Objectives:** the task of this module is to analyse the quality metrics resulting from the network syntacitc checks process and identify anomalous situations that may require further investigation.
- **Functionality:** the module computes statistics on the quality metrics over a specified lookback period and compares them with present values. When anomalous situations are identified, warnings are produced, as well as data to easily create plots that summarise the evolution of metrics over time and the frequency of each type of error.
  Functionality details may be found in the software requirements: 3.2.2 NetworkQualityWarnings
- **Data Inputs and Outputs:**
    - Input:
        - I.9 MNO Network Topology Data Quality Metrics
    - Output:
        - I.10 MNO Network Topology Data Quality Warnings – log table
        - I.23 MNO Network Syntactic Quality Warnings Line Plot Data
        - I.24 MNO Network Syntactic Quality Warnings Pie Plot Data

5.2.5.2 DEVELOPMENT DESIGN

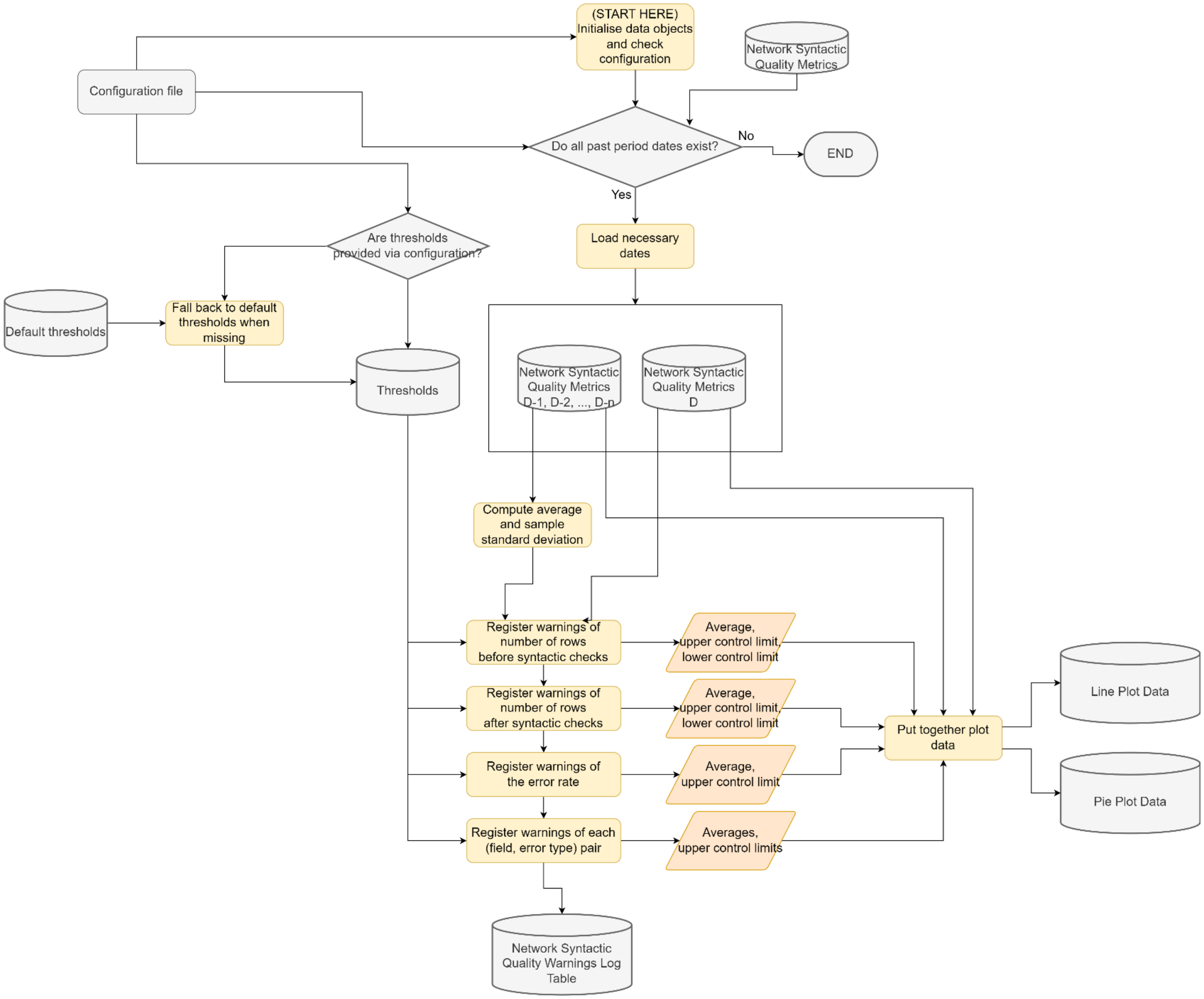
- **Key Algorithms/Processes:**
    - Create the data objects.
    - The thresholds to be used for raising warnings that are specified via configuration file are read and their types and values are validated. In the case that a specific threshold is not present, its default value is used instead.
    - Read the length of the lookback period from the configuration file, as well as the date to be studied. If the study date is not present, or any of the lookback dates are not present, an exception is raised and the execution stops.
    - Compute the necessary statistics, namely the average and the sample standard deviation, of each quality metric over the lookback period.
    - Load the values of the quality metrics for the study date.
    - Register warnings regarding the number of rows before the syntactic checks when:
        - The study date's number of rows is greater than the average number of rows over the previous period by more than a specified threshold percentage.
        - The study date's number of rows is smaller than the average number of rows over the previous period by more than a specified threshold percentage.

  - The study date's number of rows is greater than the average number of rows over the previous period by a specified number of standard deviations – that is, greater than the *upper control limit*.
  - The study date's number of rows is smaller than the average number of rows over the previous period by a specified number of standard deviations – that is, smaller than the *lower control limit*.
  - The study date's number of rows is greater than a specified absolute threshold.
  - The study date's number of rows is smaller than a specified absolute threshold.
- Register warnings regarding the number of rows after the syntactic checks when:
  - The study date's number of rows is greater than the average number of rows over the previous period by more than a specified threshold percentage.
  - The study date's number of rows is smaller than the average number of rows over the previous period by more than a specified threshold percentage.
  - The study date's number of rows is greater than the average number of rows over the previous period by a specified number of standard deviations – that is, greater than the *upper control limit*.
  - The study date's number of rows is smaller than the average number of rows over the previous period by a specified number of standard deviations – that is, smaller than the *lower control limit*.
  - The study date's number of rows is greater than a specified absolute threshold.
  - The study date's number of rows is smaller than a specified absolute threshold.
- Register warnings regarding the overall error rate in the syntactic checks process when:
  - The study date's error rate is greater than the average error rate over the previous period by more than a specified threshold percentage.
  - The study date's error rate is greater than the average error rate over the previous period by a specified number of standard deviations – that is, greater than the *upper control limit*.
  - The study date's error rate is greater than a specified absolute threshold.
- Register warning regarding the number of errors that each field presented in each error type (separately):
  - The study date's number of errors of a given error type in a given field is greater than the average number of errors of that error type in that field over the previous period by more than a specified threshold percentage.
  - The study date's number of errors of a given error type in a given field is greater than the average number of errors of that error type in that field over the previous period by a specified number of standard deviations – that is, greater than the *upper control limit*.
  - The study date's number of errors of a given error type in a given field is greater than a specified absolute threshold.
- Write every registered warning into a log table, specifying:
  - the study date,
  - the date in which the quality warnings component is being executed,
  - the value of the metric that raised the warning,
  - the condition that had to be fulfilled to raise the warning,
  - the threshold with which the metric was compared, and
  - a warning text giving context to the warning.
- Using the statistics computed previously, prepare the data needed to plot the required graphs:
  - Line plot showing the evolution of the number of rows before the syntactic checks over the lookback period and the study date, together with the average over the previous

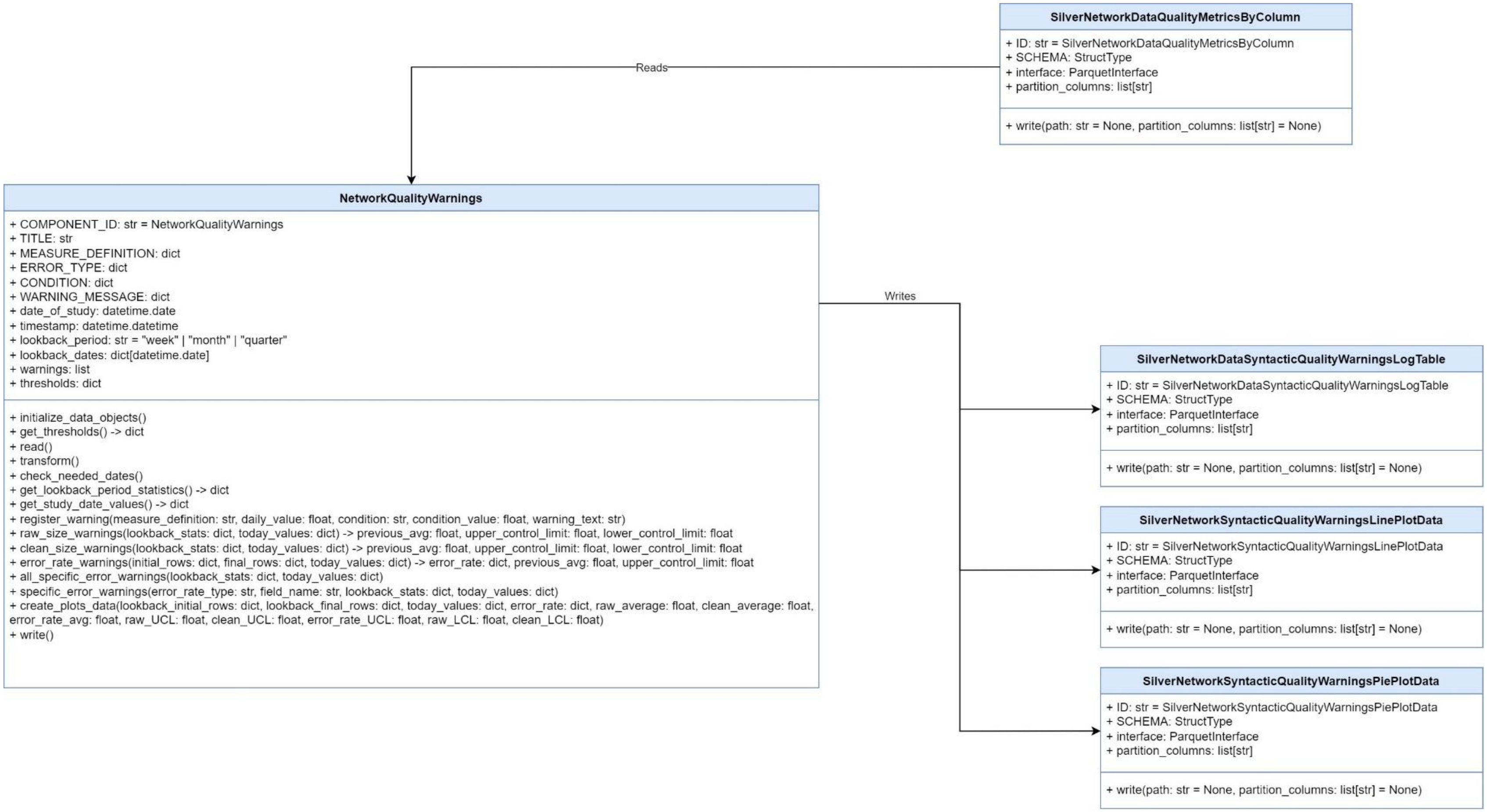
period, the upper control limit, and the lower control limit. Save the data needed to make this graph into parquet format.

- Line plot showing the evolution of the number of rows after the syntactic checks over the lookback period and the study date, together with the average over the previous period, the upper control limit, and the lower control limit. Save the data needed to make this graph into a parquet file.
- Line plot showing the evolution of the error rate over the lookback period and the study date, together with the average over the previous period and the upper control limit. Save the data needed to make this graph into a parquet file.
- For each field, a pie chart showing the percentage distribution of each type of error present in that field for the study date. Save the data needed to make this graph into respective parquet files.

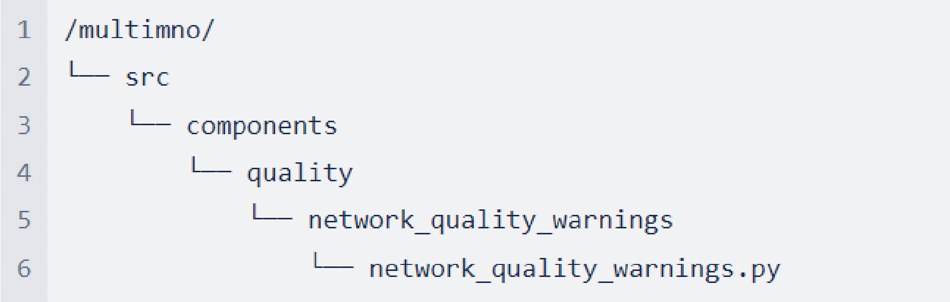
- **Data flow diagram:**


(START HERE) Initialise data objects and check configuration
Network Syntactic Quality Metrics
Configuration file
Do all past period dates exist?
No
END
Yes
Load necessary dates
Are thresholds provided via configuration?
Default thresholds
Fall back to default thresholds when missing
Thresholds
Network Syntactic Quality Metrics D-1, D-2, ..., D-n
Network Syntactic Quality Metrics D
Compute average and sample standard deviation
Register warnings of number of rows before syntactic checks
Average, upper control limit, lower control limit
Register warnings of number of rows after syntactic checks
Average, upper control limit, lower control limit
Register warnings of the error rate
Average, upper control limit
Register warnings of each (field, error type) pair
Averages, upper control limits
Put together plot data
Line Plot Data
Pie Plot Data
Network Syntactic Quality Warnings Log Table

- **Class diagram:**


SilverNetworkDataQualityMetricsByColumn
+ ID: str = SilverNetworkDataQualityMetricsByColumn
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
Reads
NetworkQualityWarnings
+ COMPONENT_ID: str = NetworkQualityWarnings
+ TITLE: str
+ MEASURE_DEFINITION: dict
+ ERROR_TYPE: dict
+ CONDITION: dict
+ WARNING_MESSAGE: dict
+ date_of_study: datetime.date
+ timestamp: datetime.datetime
+ lookback_period: str = "week" | "month" | "quarter"
+ lookback_dates: dict[datetime.date]
+ warnings: list
+ thresholds: dict
+ initialize_data_objects()
+ get_thresholds() -> dict
+ read()
+ transform()
+ check_needed_dates()
+ get_lookback_period_statistics() -> dict
+ get_study_date_values() -> dict
+ register_warning(measure_definition: str, daily_value: float, condition: str, condition_value: float, warning_text: str)
+ raw_size_warnings(lookback_stats: dict, today_values: dict) -> previous_avg: float, upper_control_limit: float, lower_control_limit: float
+ clean_size_warnings(lookback_stats: dict, today_values: dict) -> previous_avg: float, upper_control_limit: float, lower_control_limit: float
+ error_rate_warnings(initial_rows: dict, final_rows: dict, today_values: dict) -> error_rate: dict, previous_avg: float, upper_control_limit: float
+ all_specific_error_warnings(lookback_stats: dict, today_values: dict)
+ specific_error_warnings(error_rate_type: str, field_name: str, lookback_stats: dict, today_values: dict)
+ create_plots_data(lookback_initial_rows: dict, lookback_final_rows: dict, today_values: dict, error_rate: dict, raw_average: float, clean_average: float, error_rate_avg: float, raw_UCL: float, clean_UCL: float, error_rate_UCL: float, raw_LCL: float, clean_LCL: float)
+ write()
Writes
SilverNetworkDataSyntacticQualityWarningsLogTable
+ ID: str = SilverNetworkDataSyntacticQualityWarningsLogTable
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
SilverNetworkSyntacticQualityWarningsLinePlotData
+ ID: str = SilverNetworkSyntacticQualityWarningsLinePlotData
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
SilverNetworkSyntacticQualityWarningsPiePlotData
+ ID: str = SilverNetworkSyntacticQualityWarningsPiePlotData
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:



```
/multimno/
└── src
    └── components
        └── quality
            └── network_quality_warnings
                └── network_quality_warnings.py
```

  - network_quality_warnings.py contains one class named NetworkQualityWarnings which is a subclass of Component. It overrides the following methods:
    - The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - The transform method handles all the logic behind the component.
    - The write method writes the quality warnings log table containing the computed warnings. It also writes into parquet files the data required to produce the defined plots.
  - The NetworkCleaning component also has the following methods:
    - get_thresholds handles the logic of reading configuration-specified thresholds and the usage of default threshold whenever a specific threshold value is not specified.
    - check_needed_dates verifies that both the study date and the dates in the lookback period are all present in the metrics data and throws an exception when some date is missing.
    - get_lookback_period_statistics computes the average and sample standard deviation of the quality metrics over the lookback period
    - get_study_date_values retrieves and computes the metric values of the study date.
    - register_warning is a method that abstracts away the creation of a warning in the log table, taking as arguments all necessary information and putting it in the correct format.
    - raw_size_warnings contains the logic behind the warning computation regarding the number of rows before the syntactic checks.
    - clean_size_warnings contains the logic behind the warning computation regarding the number of rows after the syntactic checks.
    - error_rate_warnings contains the logic behind the warning computation regarding the error rate detected in the syntactic checks process.
    - all_specific_error_warnings loops over all field and error type specific errors in order to compute their warnings.
    - specific_error_warning contains the logic behind the warning computation regarding a specific (field, error type) pair determined in the all_specific_error_warnings method.
    - create_plots_data gathers and formats the data required to produce the necessary plots of the component.
  - NetworkCleaning also contains as attributes different sets of formattable strings used in the generation of the log table. These include MEASURE_DEFINITION, CONDITION, TITLE, WARNING_MESSAGE.

5.2.6 SIGNALSTRENGTHMODELING

5.2.6.1 MODULE DESCRIPTION

- **Module Name:** SignalStrengthModeling
- **Objectives:** responsible for modeling the signal strength propagation in a cellular network.
- **Functionality:** takes as input a configuration file and a set of data representing the network's cells and their physical properties. The component then calculates the signal strength at various points of a reference grid, taking into account factors such as the distance to the cell, physical properties of the cell, the azimuth and elevation angles of the cell, the directionality of the cell and physical environment. Functionality details may be found in the software requirements: 3.2.3 SignalStrengthModeling
- **Data Inputs and Outputs:**
  - Inputs:
    - I.8 Cell Locations with Physical Properties – Cleaned
    - I.11 Reference Grid
  - Outputs:
    - I.12 Cells Signal Strengths

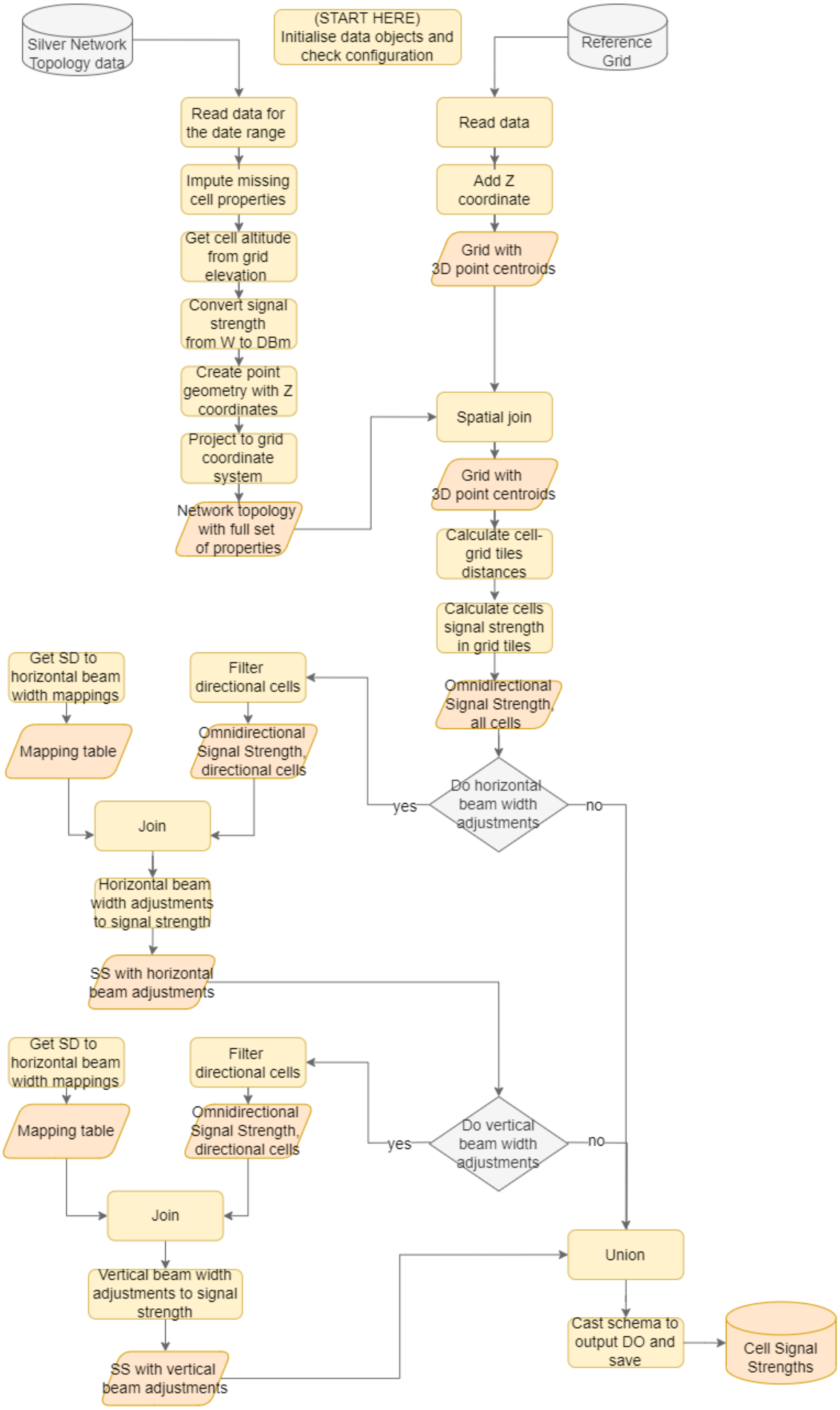
5.2.6.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**

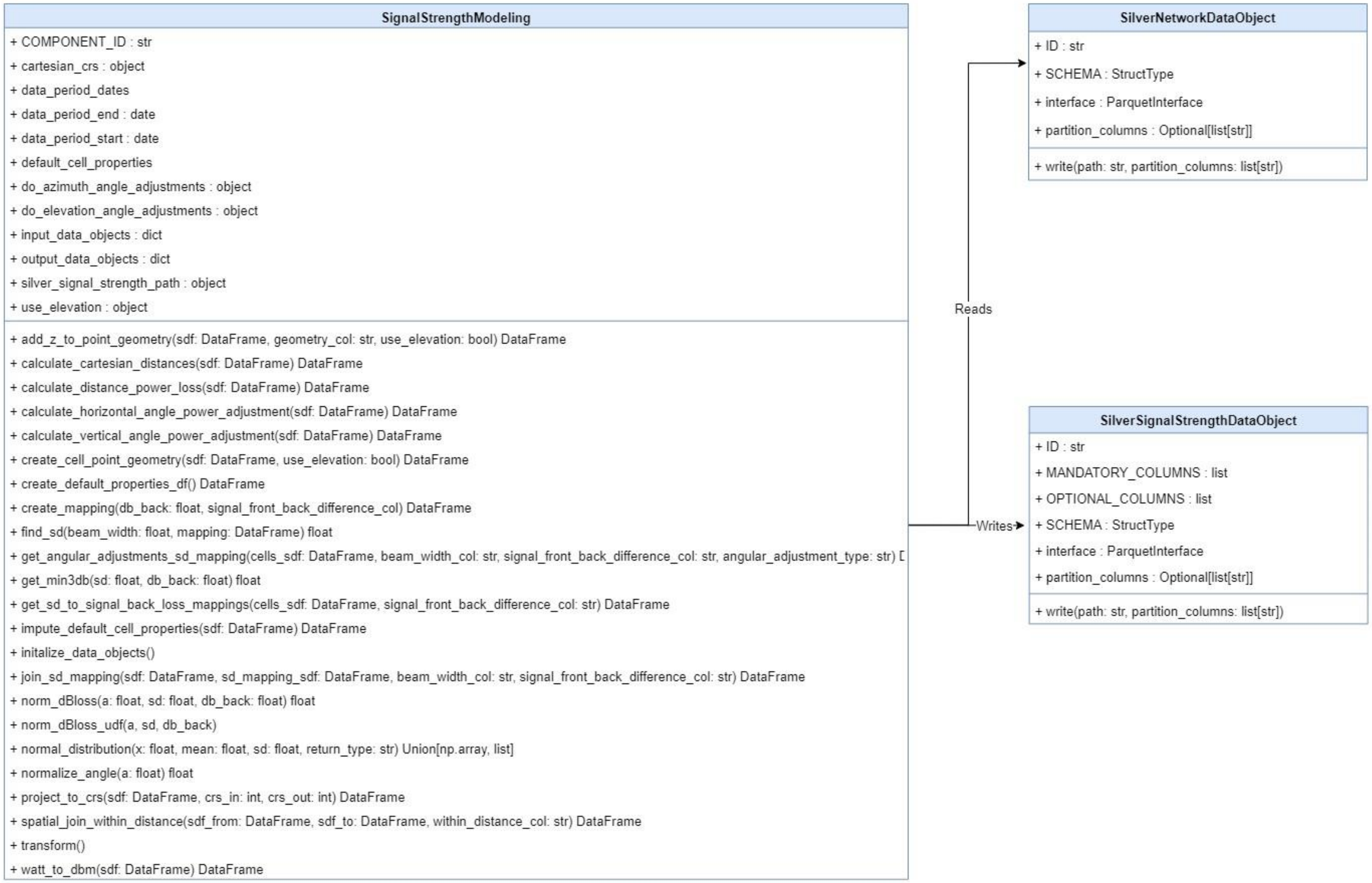
1. Initialisation. Read all necessary config parameters, check the availability of input data, read it in data objects.
2. Prepare input datasets:
    1. Filter input network data to include only date range specified in config.
    2. Add Z coordinate to grid centroids. If elevation is used Z coordinate is assigned based on a grid elevation property. If elevation is not used assign Z = 0.
    3. Check that all necessary cells' physical properties are present and impute missing properties. Based on a cell type, missing properties are filled in by taking default values for this cell type defined in config file. If cell type is not defined or not present in config file default cell properties are assigned.
    4. Convert signal strength values from W to DBm.
    5. Create cell point geometries. If elevation is used, set Z coordinate as altitude + antenna height. If elevation is not used, set Z coordinate as 0 + antenna height.
    6. Project coordinate system of cell geometries to coordinate system of the reference grid.
3. Spatial join of cells to grid centroids. Join is done based on spatial intersection of a buffer polygons around cell points of a radius equal to the maximum cell range with grid centroids.
4. Calculate planar and 3D cartesian distances between cell point and all joined grid centroids.
5. Calculate signal strength in grid tiles. Using power and Path Loss Exponent cell properties and distances to joined grid tiles within cell range calculate signal strength in every grid tile with signal strength propagation equation.
6. Perform horizontal beam width adjustments to signal strength for directional cells. Optional depending on config parameter.
    1. Get mapping table with standard deviations in signal strength in all horizontal angles for all combinations of horizontal beam widths and signal strength differences between front and back of antennas.
    2. Filter only directional cells from cell-grid dataset.
    3. Join mapping table with standard deviations to cell-grid dataset.

    4. Calculate signal strength adjustments based on relative azimuth angle and the distance between grid tiles and a cell using joined standard deviation value for a cell horizontal beam width.
7. Perform vertical beam width adjustments to signal strength for directional cells. Optional depending on config parameter.
    1. Get mapping table with standard deviations in signal strength in all vertical angles for all combinations of vertical beam widths and signal strength differences between front and back of antennas.
    2. Filter only directional cells from cell-grid dataset.
    3. Join mapping table with standard deviations to cell-grid dataset.
    4. Calculate signal strength adjustments based on elevation angle and the distance between grid tiles and a cell using joined standard deviation value for a cell vertical beam width.
8. Union directional and non-directional cell-grid datasets.
9. Convert cell-grid dataset schema to match the output data object schema and save to storage.

- **Data flow diagram:**


Silver Network Topology data
(START HERE) Initialise data objects and check configuration
Reference Grid
Read data for the date range
Impute missing cell properties
Get cell altitude from grid elevation
Convert signal strength from W to DBm
Create point geometry with Z coordinates
Project to grid coordinate system
Network topology with full set of properties
Read data
Add Z coordinate
Grid with 3D point centroids
Spatial join
Grid with 3D point centroids
Calculate cell-grid tiles distances
Calculate cells signal strength in grid tiles
Omnidirectional Signal Strength, all cells
Do horizontal beam width adjustments
yes
no
Get SD to horizontal beam width mappings
Mapping table
Filter directional cells
Omnidirectional Signal Strength, directional cells
Join
Horizontal beam width adjustments to signal strength
SS with horizontal beam adjustments
Get SD to horizontal beam width mappings
Mapping table
Filter directional cells
Omnidirectional Signal Strength, directional cells
Do vertical beam width adjustments
yes
no
Join
Vertical beam width adjustments to signal strength
SS with vertical beam adjustments
Union
Cast schema to output DO and save
Cell Signal Strengths

- **Class diagram:**



SignalStrengthModeling

+ COMPONENT_ID : str
+ cartesian_crs : object
+ data_period_dates
+ data_period_end : date
+ data_period_start : date
+ default_cell_properties
+ do_azimuth_angle_adjustments : object
+ do_elevation_angle_adjustments : object
+ input_data_objects : dict
+ output_data_objects : dict
+ silver_signal_strength_path : object
+ use_elevation : object

+ add_z_to_point_geometry(sdf: DataFrame, geometry_col: str, use_elevation: bool) DataFrame
+ calculate_cartesian_distances(sdf: DataFrame) DataFrame
+ calculate_distance_power_loss(sdf: DataFrame) DataFrame
+ calculate_horizontal_angle_power_adjustment(sdf: DataFrame) DataFrame
+ calculate_vertical_angle_power_adjustment(sdf: DataFrame) DataFrame
+ create_cell_point_geometry(sdf: DataFrame, use_elevation: bool) DataFrame
+ create_default_properties_df() DataFrame
+ create_mapping(db_back: float, signal_front_back_difference_col) DataFrame
+ find_sd(beam_width: float, mapping: DataFrame) float
+ get_angular_adjustments_sd_mapping(cells_sdf: DataFrame, beam_width_col: str, signal_front_back_difference_col: str, angular_adjustment_type: str) D
+ get_min3db(sd: float, db_back: float) float
+ get_sd_to_signal_back_loss_mappings(cells_sdf: DataFrame, signal_front_back_difference_col: str) DataFrame
+ impute_default_cell_properties(sdf: DataFrame) DataFrame
+ initalize_data_objects()
+ join_sd_mapping(sdf: DataFrame, sd_mapping_sdf: DataFrame, beam_width_col: str, signal_front_back_difference_col: str) DataFrame
+ norm_dBloss(a: float, sd: float, db_back: float) float
+ norm_dBloss_udf(a, sd, db_back)
+ normal_distribution(x: float, mean: float, sd: float, return_type: str) Union[np.array, list]
+ normalize_angle(a: float) float
+ project_to_crs(sdf: DataFrame, crs_in: int, crs_out: int) DataFrame
+ spatial_join_within_distance(sdf_from: DataFrame, sdf_to: DataFrame, within_distance_col: str) DataFrame
+ transform()
+ watt_to_dbm(sdf: DataFrame) DataFrame

SilverNetworkDataObject

+ ID : str
+ SCHEMA : StructType
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]

+ write(path: str, partition_columns: list[str])

SilverSignalStrengthDataObject

+ ID : str
+ MANDATORY_COLUMNS : list
+ OPTIONAL_COLUMNS : list
+ SCHEMA : StructType
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]

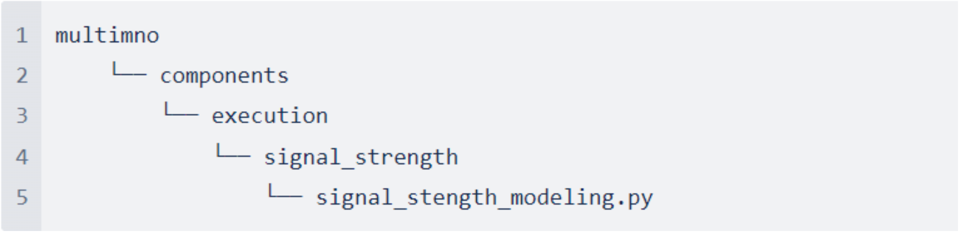
+ write(path: str, partition_columns: list[str])

-

- **Code Structure:**

The code structure follows the format set by the core package, and the general repository structure.

The location of the module script in the repository is as follows:



```
multimno
     └── components
           └── execution
                 └── signal_strength
                       └── signal_stength_modeling.py
```

signal_stength_modeling.py contains one class named SignalStrengthModelingwhich is a subclass of Component.

The SignalStrengthModelingclass overrides transform method of base Component class.
transform method performs all necessary filtering and transformations of network topology data for signal strengths modeling by sequentially calling other methods that perform the actual data manipulation.

5.2.7 CELLFOOTPRINTESTIMATION

5.2.7.1 MODULE DESCRIPTION

- **Module Name:** CellFootprintEstimation
- **Objectives:** convert cells signal strength to signal dominance (cell footprint). Optionally calculate cells intersection groups.
- **Functionality:** takes as input a configuration file and Signal Strength Data. The component then calculates the signal dominance per grid tile and applies any combination out of 3 pruning methods depending on config parameters. Optionally component also produces Cell Intersection Groups. Functionality details may be found in the software requirements: 3.2.4 CellFootprintEstimation
- **Data Inputs and Outputs:**
    - Input:
        - I.12 Cells Signal Strengths
    - Outputs:
        - I.13 Cell Footprints
        - I.14 Cell Intersection Groups

5.2.7.2 DEVELOPMENT DESIGN

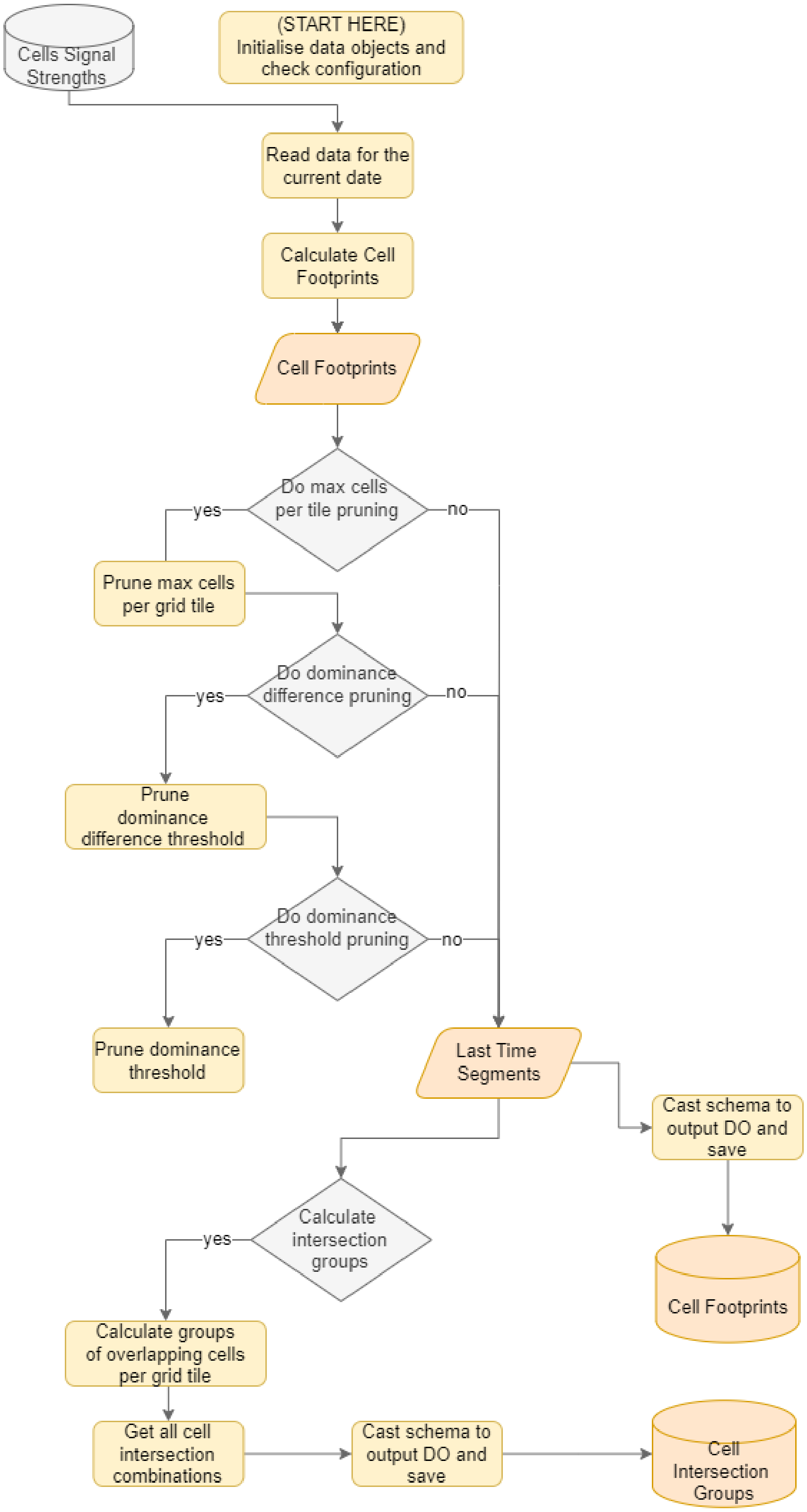
- **Key Algorithms/Processes:**

1. Initialisation. Read all necessary config parameters, check the availability of input data, read it in data objects.
2. Prepare input datasets:
    1. Filter input Signal Strength data to include only date range specified in config.
3. Calculate signal dominance (cell footprint) from signal strength values.
4. Apply set of pruning methods depending on configuration:

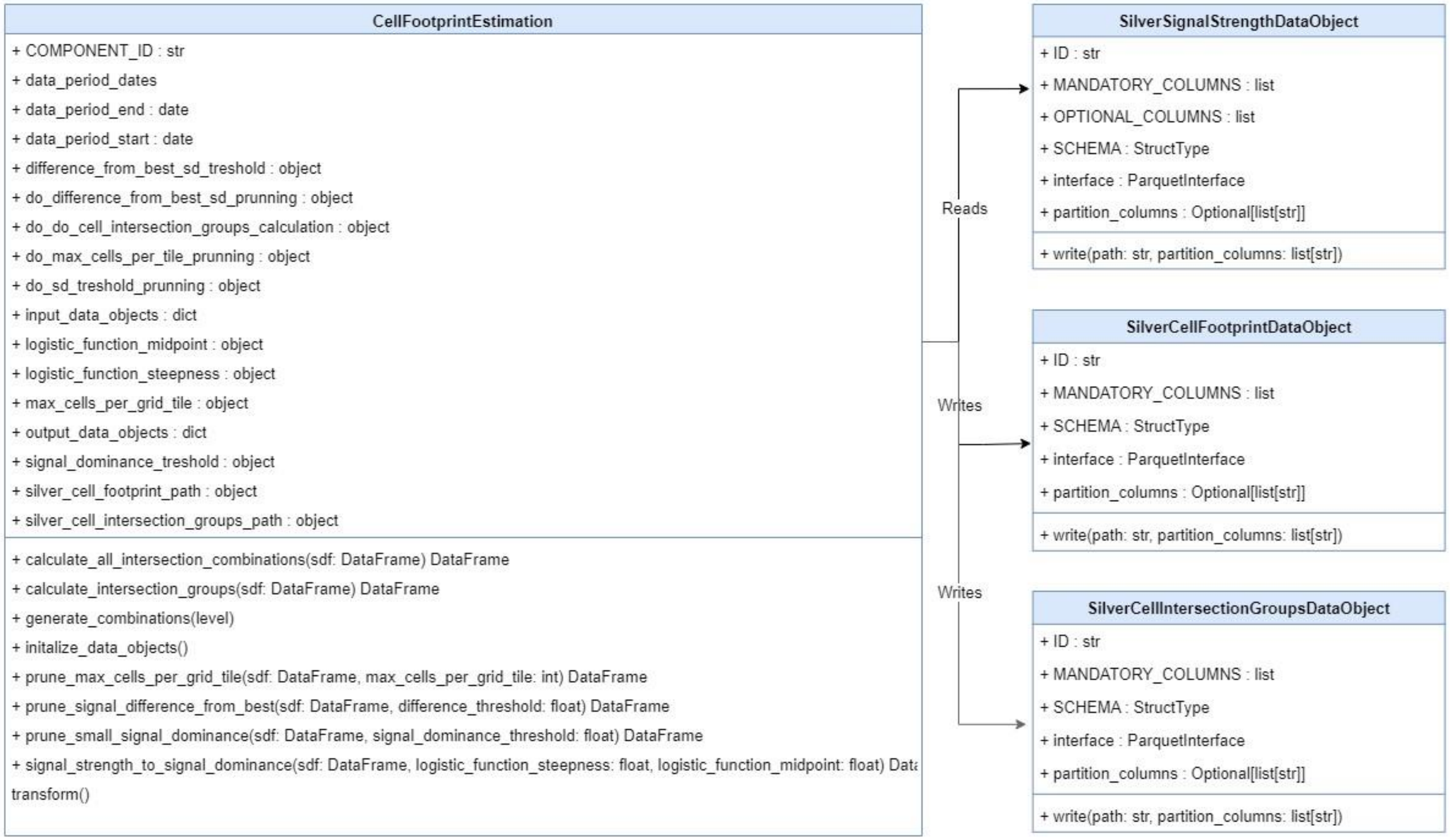
   1. Maximum cells per grid tile. Keep predefined number of contributing to overall tile's signal dominance cells per grid tile. Optional step depending on configuration.
   2. Threshold difference from the best signal dominance. Always keep best signal dominance cell per grid tile. Then calculate the difference of all other cells in this tile from the best and prune cells under predefined difference threshold. Optional step depending on configuration.
   3. Threshold signal dominance. Prune all cells with signal dominance value under threshold. Optional step depending on configuration.

5. Calculate Cell Intersection Groups. Optional step depending on configuration:
   1. Aggregate all contributing cells per grid tile into lists, drop duplicates.
   2. Extract all possible combinations of overlapping cells from previously calculated lists, drop duplicates.
6. Convert output datasets schema to match the output data objects schemas and save to storage.

- **Data flow diagram:**


Cells Signal Strengths
(START HERE) Initialise data objects and check configuration
Read data for the current date
Calculate Cell Footprints
Cell Footprints
Do max cells per tile pruning
yes
no
Prune max cells per grid tile
Do dominance difference pruning
yes
no
Prune dominance difference threshold
Do dominance threshold pruning
yes
no
Prune dominance threshold
Last Time Segments
Cast schema to output DO and save
Cell Footprints
Calculate intersection groups
yes
Calculate groups of overlapping cells per grid tile
Get all cell intersection combinations
Cast schema to output DO and save
Cell Intersection Groups

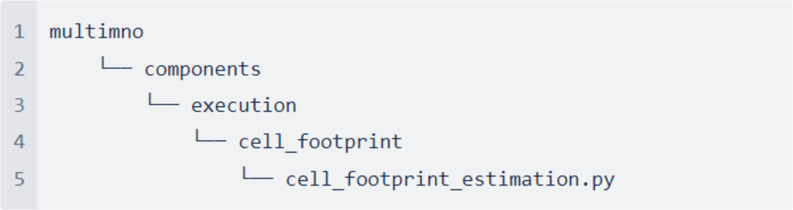
- **Class diagram:**


CellFootprintEstimation
+ COMPONENT_ID : str
+ data_period_dates
+ data_period_end : date
+ data_period_start : date
+ difference_from_best_sd_treshold : object
+ do_difference_from_best_sd_prunning : object
+ do_do_cell_intersection_groups_calculation : object
+ do_max_cells_per_tile_prunning : object
+ do_sd_treshold_prunning : object
+ input_data_objects : dict
+ logistic_function_midpoint : object
+ logistic_function_steepness : object
+ max_cells_per_grid_tile : object
+ output_data_objects : dict
+ signal_dominance_treshold : object
+ silver_cell_footprint_path : object
+ silver_cell_intersection_groups_path : object
+ calculate_all_intersection_combinations(sdf: DataFrame) DataFrame
+ calculate_intersection_groups(sdf: DataFrame) DataFrame
+ generate_combinations(level)
+ initalize_data_objects()
+ prune_max_cells_per_grid_tile(sdf: DataFrame, max_cells_per_grid_tile: int) DataFrame
+ prune_signal_difference_from_best(sdf: DataFrame, difference_threshold: float) DataFrame
+ prune_small_signal_dominance(sdf: DataFrame, signal_dominance_threshold: float) DataFrame
+ signal_strength_to_signal_dominance(sdf: DataFrame, logistic_function_steepness: float, logistic_function_midpoint: float) Data
transform()
Reads
Writes
Writes
SilverSignalStrengthDataObject
+ ID : str
+ MANDATORY_COLUMNS : list
+ OPTIONAL_COLUMNS : list
+ SCHEMA : StructType
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]
+ write(path: str, partition_columns: list[str])
SilverCellFootprintDataObject
+ ID : str
+ MANDATORY_COLUMNS : list
+ SCHEMA : StructType
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]
+ write(path: str, partition_columns: list[str])
SilverCellIntersectionGroupsDataObject
+ ID : str
+ MANDATORY_COLUMNS : list
+ SCHEMA : StructType
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]
+ write(path: str, partition_columns: list[str])

- **Code Structure:**

The code structure follows the format set by the core package, and the general repository structure.

The location of the module script in the repository is as follows:



```
multimno
    └── components
        └── execution
            └── cell_footprint
                └── cell_footprint_estimation.py
```

cell_footprint_estimation.py contains one class named CellFootprintEstimationwhich is a subclass of Component.

The CellFootprintEstimationwhich class overrides transform method of base Component class. transform method performs all necessary filtering and transformations of Signal Strength data to convert it to signal dominance (cell footprint) by sequentially calling other methods that perform the actual data manipulation.

5.2.8 CELLCONNECTIONPROBABILITYESTIMATION

5.2.8.1 MODULE DESCRIPTION

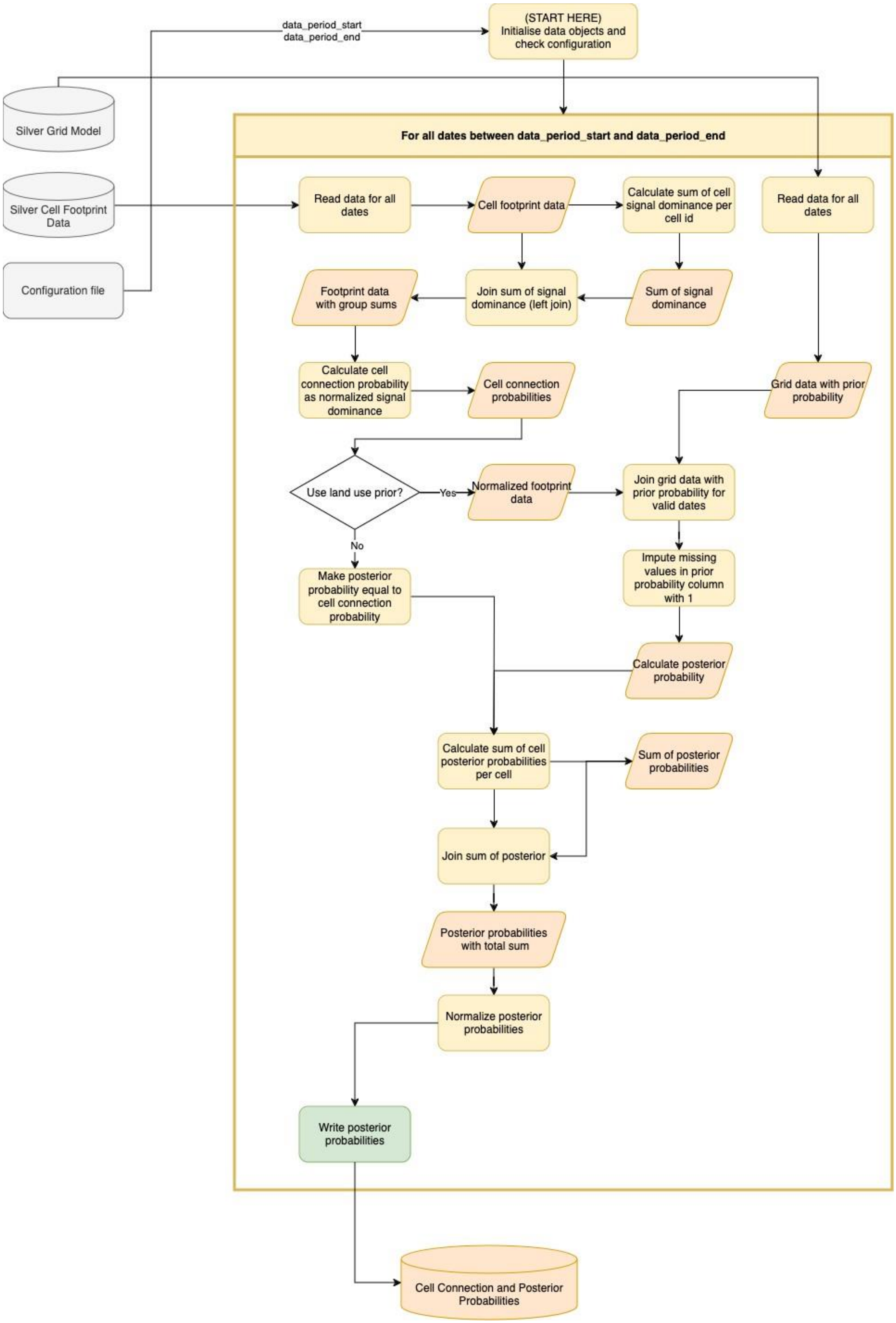
- **Module Name:** CellConnectionProbabilityEstimation
- **Objectives:** this module calculates cell connection probabilities based on the cell footprint values, and optionally applies the land use prior probabilities, to get the posterior probabilities for each cell id and grid id.
- **Functionality:**
  the component reads in cell footprint data, calculates cell connection probabilities for each grid, and then performs posterior calculation , using prior probability values from the reference grid, when so specified in the configuration.
  Functionality is outlined in the software requirement specifications:
  3.2.5 CellConnectionProbabilityEstimation
- **Data Inputs and Outputs:**
  - Input:
    - I.13 Cell Footprints
    - I.11 Reference Grid
  - Outputs:
    - I.15 Cell Connection and Posterior Probabilities

5.2.8.2 DEVELOPMENT DESIGN

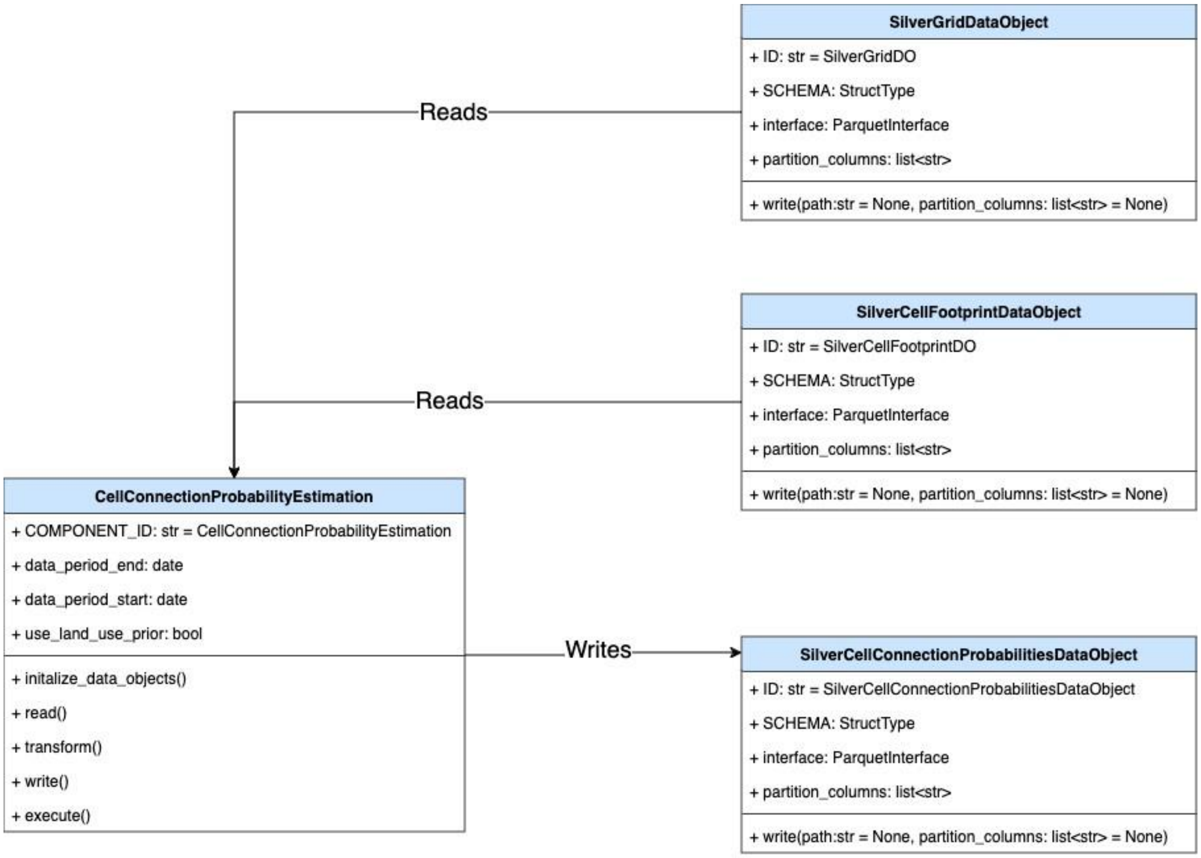
- **Key Algorithms/Processes:**
  The data is processed for the selected dates range. All following steps are run for each date:
  - Calculate sum of cell footprint for each grid_id.
  - Calculate cell connection probability as ratio of cell footprint to sum of footprint for grid_id.
  - If so set in configuration, join the grid data with prior probabilities to the result of previous step

- If so set in configuration, calculate posterior probabilities by multiplying prior probabilities and cell connection probabilities. Otherwise make posterior probabilities equal with cell connection probabilities.
- Calculate sum of cell connection probabilities for cell id.
- Normalise posterior probabilities using sum of previous step.

- **Data flow diagram:**


data_period_start
data_period_end
(START HERE)
Initialise data objects and check configuration
Silver Grid Model
Silver Cell Footprint Data
Configuration file
For all dates between data_period_start and data_period_end
Read data for all dates
Cell footprint data
Calculate sum of cell signal dominance per cell id
Read data for all dates
Footprint data with group sums
Join sum of signal dominance (left join)
Sum of signal dominance
Calculate cell connection probability as normalized signal dominance
Cell connection probabilities
Grid data with prior probability
Use land use prior?
Yes
Normalized footprint data
Join grid data with prior probability for valid dates
No
Make posterior probability equal to cell connection probability
Impute missing values in prior probability column with 1
Calculate posterior probability
Calculate sum of cell posterior probabilities per cell
Sum of posterior probabilities
Join sum of posterior
Posterior probabilities with total sum
Normalize posterior probabilities
Write posterior probabilities
Cell Connection and Posterior Probabilities

- **Class diagram:**


SilverGridDataObject
+ ID: str = SilverGridDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Reads
SilverCellFootprintDataObject
+ ID: str = SilverCellFootprintDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Reads
CellConnectionProbabilityEstimation
+ COMPONENT_ID: str = CellConnectionProbabilityEstimation
+ data_period_end: date
+ data_period_start: date
+ use_land_use_prior: bool
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
Writes
SilverCellConnectionProbabilitiesDataObject
+ ID: str = SilverCellConnectionProbabilitiesDataObject
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)

- **Code Structure:**
  The code structure follows the format set by the core package, and the general repository structure.

  The location of the module script in the repository is as follows:

```
/multimno_internal/
└─ multimno
    └─ components
        └─ execution
            └─ cell_connection_probability
                └─ cell_connection_probability.py
```

  - cell_connection_probability.py contains one class named CellConnectionProbabilityEstimation which is a subclass of Component.

  The CellConnectionProbabilityEstimation class overwrites __init__ andtransform in the Component class.

  __init__ method initialises the data objects and reads the necessary values from config file.
  transform performs all necessary transformations and calculation of cell probability estimation for the

entire period. transform does not contain any calls to smaller functions, but holds the entire processing flow.

5.2.9 SEMANTICCLEANING

5.2.9.1 MODULE DESCRIPTION

- **Module Name:** SemanticCleaning
- **Objectives:** the objective of this module is to perform checks to identify and flag semantically erroneous events of devices.
- **Functionality:** the semantic checks include the following checks:
  - Valid reference to a cell identifier: whether an event makes a reference to an existent cell that is operative at the event's timestamp, and if the cell exists, whether it was operative or not.
  - Illogical change of location of the device based on time and distance difference between consecutive events: some events are flagged as incorrect and others are flagged as suspicious. Functionality is outlined in the software requirement specifications: 3.2.9 SemanticCleaning

  At this moment in time, only the processing of cell locations with physical properties is implemented.
- **Data Inputs and Outputs:**
  - Input:
    - I.1 MNO Event Data - Raw
    - I.8 Cell Locations with Physical Properties – Cleaned
  - Output:
    - I.16 MNO Event Data – Semantically Cleaned
    - I.17 MNO Device Semantic Quality Metrics

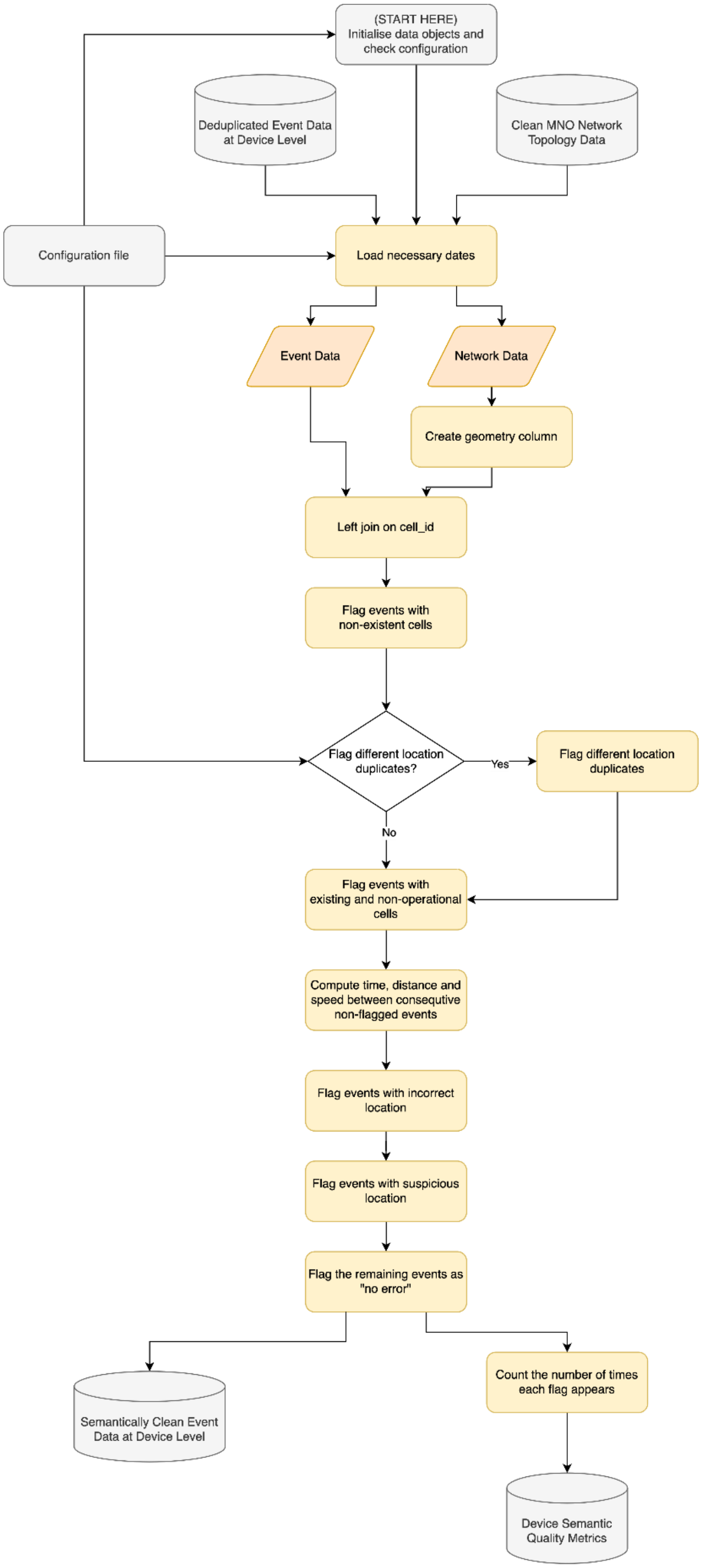
5.2.9.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
  - Create the data objects: events and cells.
  - Read from the configuration file the minimum distance and speed for which an event might be classified as semantically erroneous.
  - Create a geometry column with the latitude-longitude point of each cell.
  - Perform a left join between events and cells by the cell ID field. In this way, non-existent cell IDs appearing in the events data will be matched with null values.
  - Whenever the geometry column is null, flag these events with the flag corresponding to a non-existent cell.
  - Then, flag different location duplicates. Different location duplicates are cases where timestamp and user_id columns have identical values for more than two rows, but the combination of values for longitude, latitude and cell_id is not identical for the same selection of rows.
  - Then, flag events that refer to an existent cell that was not operative when the event was registered with the corresponding flag. The geometry column created above, which is just an auxiliar column, is set to null for these flagged events for convenience later on.
  - Next, semantically erroneous events regarding location will be flagged. For this it is necessary to compute the estimated distance and speed between two consecutive events *which have not been flagged*. This is achieved as follows:
    - Create two windows, both partitioned by year, month, day, user_id_modulo (these four are the partitoin variables of event data) and user_id, and ordered by timestamp. One window will comprise all events following the current position (from the current position plus one, to unbounded following), and the other will comprise all events

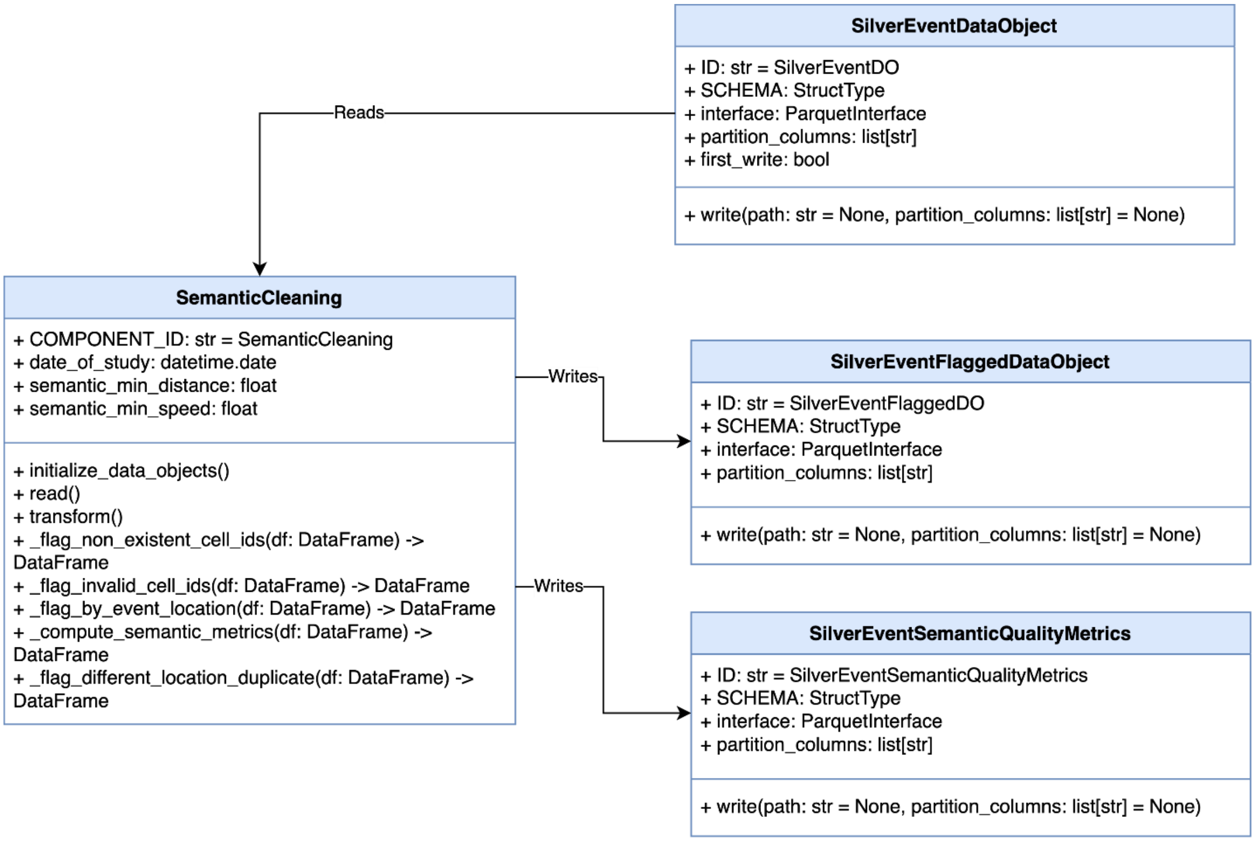
preceding the current position (from unbounded preceding, to the current position minus one).

  - Using these two windows *and 'skipping' all events previously flagged*, four auxiliary columns are created, containing the time difference to the next event, the time difference from the previous event, the distance to the following event, and the distance to the previous event, respectively.
  - Then, two additional columns are created with the estimated mean speed with respect to next and previous events respectively.
  - With all the necessary information already computed, events are now flagged:
    - Whenever the distance and speed to <u>both</u> the next and previous events surpass their thresholds specified via configuration, the event is flagged as an event with an incorrect location.
    - Whenever the distance and speed to <u>either, but not both</u>, the next or previous events surpass the thresholds specified via configuration, the event is flagged as an event with a suspicious location.
    - The first and last events of the day for a given device are compared with the second and second-to-last events of the day respectively. If the distance and speed thresholds are surpassed, they are flagged with a suspicious location.
- The rest of the events that have not been flagged until now are given the 'no error' flag.
- All auxiliary columns are removed and only those fields in the output event data object are left.
- The dataframe is cached or persisted into memory.
- Semantic metrics are computed: the now flagged event data is grouped by error flag and the number of occurrences of each flagged is counted.
- The output event data and semantic metrics are saved.

- **Data flow diagram:**


(START HERE) Initialise data objects and check configuration
Deduplicated Event Data at Device Level
Clean MNO Network Topology Data
Configuration file
Load necessary dates
Event Data
Network Data
Create geometry column
Left join on cell_id
Flag events with non-existent cells
Flag different location duplicates?
Yes
Flag different location duplicates
No
Flag events with existing and non-operational cells
Compute time, distance and speed between consequtive non-flagged events
Flag events with incorrect location
Flag events with suspicious location
Flag the remaining events as "no error"
Semantically Clean Event Data at Device Level
Count the number of times each flag appears
Device Semantic Quality Metrics

- **Class diagram:**


SilverEventDataObject
+ ID: str = SilverEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ first_write: bool
+ write(path: str = None, partition_columns: list[str] = None)
Reads
SemanticCleaning
+ COMPONENT_ID: str = SemanticCleaning
+ date_of_study: datetime.date
+ semantic_min_distance: float
+ semantic_min_speed: float
+ initialize_data_objects()
+ read()
+ transform()
+ _flag_non_existent_cell_ids(df: DataFrame) -> DataFrame
+ _flag_invalid_cell_ids(df: DataFrame) -> DataFrame
+ _flag_by_event_location(df: DataFrame) -> DataFrame
+ _compute_semantic_metrics(df: DataFrame) -> DataFrame
+ _flag_different_location_duplicate(df: DataFrame) -> DataFrame
Writes
SilverEventFlaggedDataObject
+ ID: str = SilverEventFlaggedDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
Writes
SilverEventSemanticQualityMetrics
+ ID: str = SilverEventSemanticQualityMetrics
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
/multimno/
└── src
    └── components
        └── execution
            └── event_semantic_cleaning
                └── event_semantic_cleaning.py
```

  - event_semantic_cleaning.py contains one class named SemanticCleaning which is a subclass of Component. It overrides the following methods:
    - The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - The transform method handles all the logic behind the component.
  - The SemanticCleaning component also has the following methods:
    - _flag_non_existent_cell_ids handles the check and flagging of references to non-existent cell IDs.
    - _flag_invalid_cell_ids handles the check and flagging of references to existent cell IDs that were not operative in the moment an event was registered.

        - _flag_by_event_location handles the check and flagging of events with an incorrect or suspicious location.
        - _compute_semantic_metrics handles the counting of occurences of each flag and formatting them as the quality metrics.
        - _flag_different_location_duplicate handles the detection and flagging of different location duplicates.

5.2.10 SEMANTICQUALITYWARNINGS

5.2.10.1 MODULE DESCRIPTION

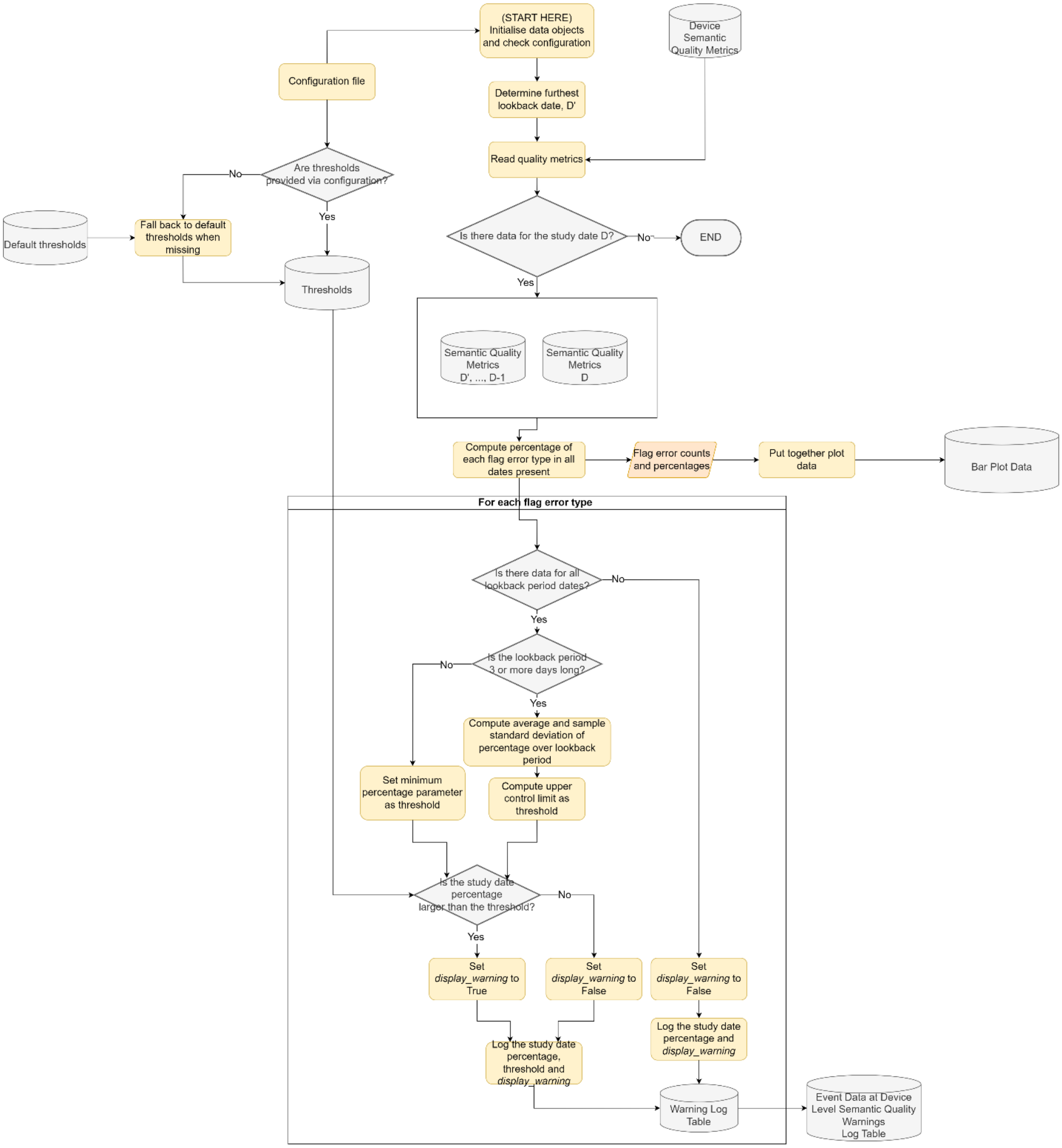
- **Module Name:** SemanticQualityWarnings
- **Objectives:** this module analyses the semantic quality metrics produced in the event semantic checks at device level process in order to identify anomalous situations that may need to be investigated further.
- **Functionality:** the module computes statistics on the semantic event quality metrics over a specified lookback period and compares them with present values. When anomalous situations are identified, warnings are produced, as well as data to easily create plots that summarise the evolution of metrics over time and the frequency of each type of error.
  Functionality is outlined in the software requirement specifications: 3.2.10 SemanticQualityWarnings
- **Data Inputs and Outputs:**
    - Input:
        - I.17 MNO Device Semantic Quality Metrics
    - Output:
        - I.18 MNO Event Data at device Level Semantic Quality Warnings – log table
        - I.25 Event Data at Device Level Semantic Quality Warnings Bar Plot Data

5.2.10.2 DEVELOPMENT DESIGN

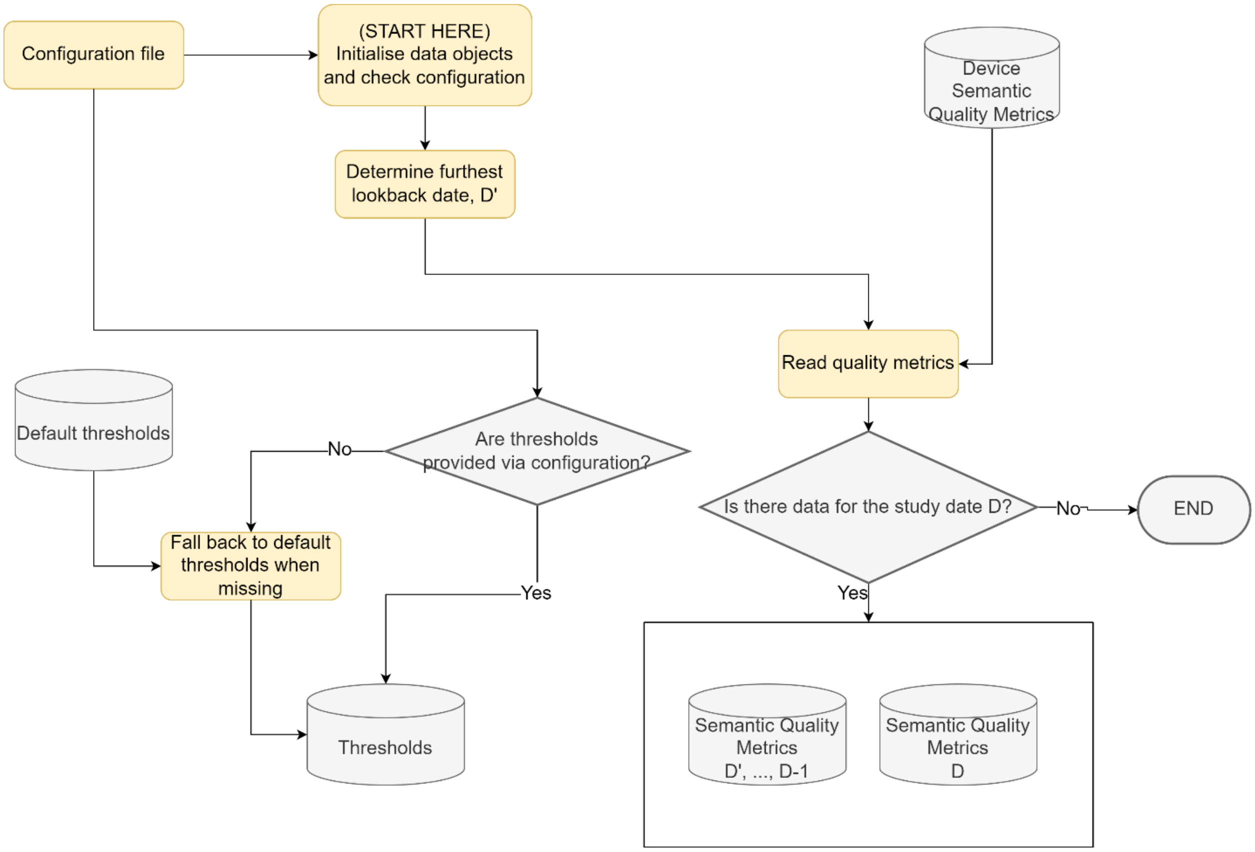
- **Key Algorithms/Processes:**
    - Create the data objects.
    - The thresholds to be used for raising warnings that are specified via configuration file are read and their types and values are validated. In the case that a specific threshold is not present, its default value is used instead.
    - Since each metric might have a different lookback period, the date furthest into the past is found, and data is read between this date and the study date.
    - If there are no metrics for the study date, an exception is raised.
    - Compute the percentage of each type of flag error, including the 'no error' flag, for each date read.
    - For each error flag (excluding 'no error' flag) do:
        - Check that the corresponding metric is present for all the lookback period dates of this error:
            - If one of them is missing, no warning is to be raised.
            - If they are all present, but the lookback period is lower than 3, use the 'min_percentage' parameter as the threshold for the warning raising condition.
            - If they are all present and the lookback period is equal or greater than 3, compute the average and sample standard deviation of the percentage of this flag error over its lookback period. Then, compute the threshold to raise a

        warning as the upper control limit (i.e. average plus the standard deviation multiplied by the ‘min_sd’ parameter).
        - If the percentage of the study date is greater than the threshold, raise a warning.
    - Log the percentage value of the present day. In the case that a threshold was computed, log it as well. If a warning is to be raised, log a True value, or False otherwise.
- Format all logged data in the required format and write it to file.
- Using the statistics computed previously, prepare the data needed to plot the required graphs:
    - Bar plot showing the absolute count of each flag error type for each date over the longest lookback period plus the study date considered in the process. Save the data needed to make this graph into a parquet file.
    - Bar plot showing the percentage of each flag error type for each date over the longest lookback period plus the study date considered in the process. Save the data needed to make this graph into a parquet file.

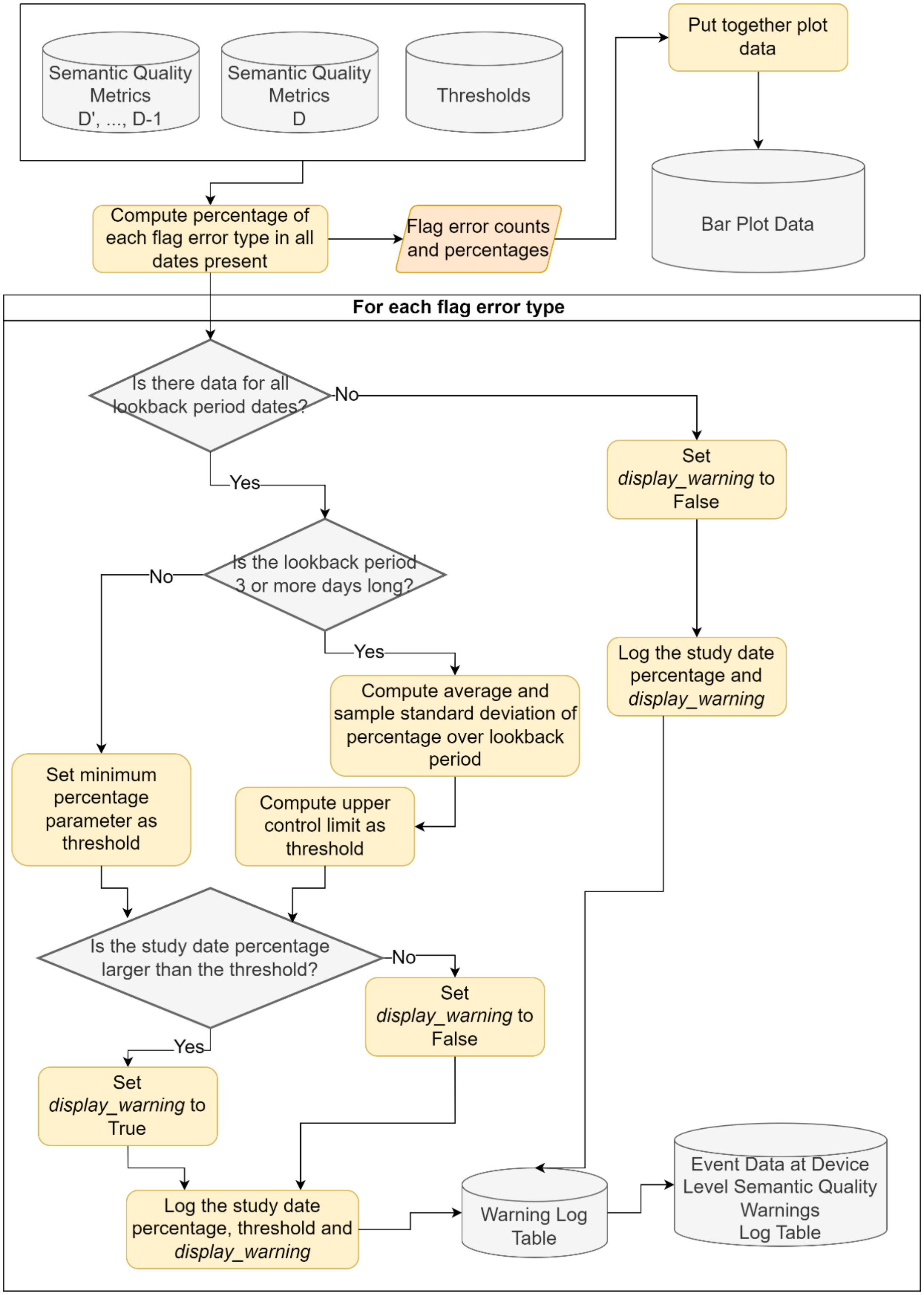
- **Data flow diagram (full view):**


(START HERE)
Initialise data objects and check configuration
Device Semantic Quality Metrics
Configuration file
Determine furthest lookback date, D'
Read quality metrics
Are thresholds provided via configuration?
No
Yes
Default thresholds
Fall back to default thresholds when missing
Thresholds
Is there data for the study date D?
No
END
Yes
Semantic Quality Metrics D', ..., D-1
Semantic Quality Metrics D
Compute percentage of each flag error type in all dates present
Flag error counts and percentages
Put together plot data
Bar Plot Data
For each flag error type
Is there data for all lookback period dates?
No
Yes
Is the lookback period 3 or more days long?
No
Yes
Compute average and sample standard deviation of percentage over lookback period
Set minimum percentage parameter as threshold
Compute upper control limit as threshold
Is the study date percentage larger than the threshold?
No
Yes
Set display_warning to True
Set display_warning to False
Set display_warning to False
Log the study date percentage and display_warning
Log the study date percentage, threshold and display_warning
Warning Log Table
Event Data at Device Level Semantic Quality Warnings Log Table

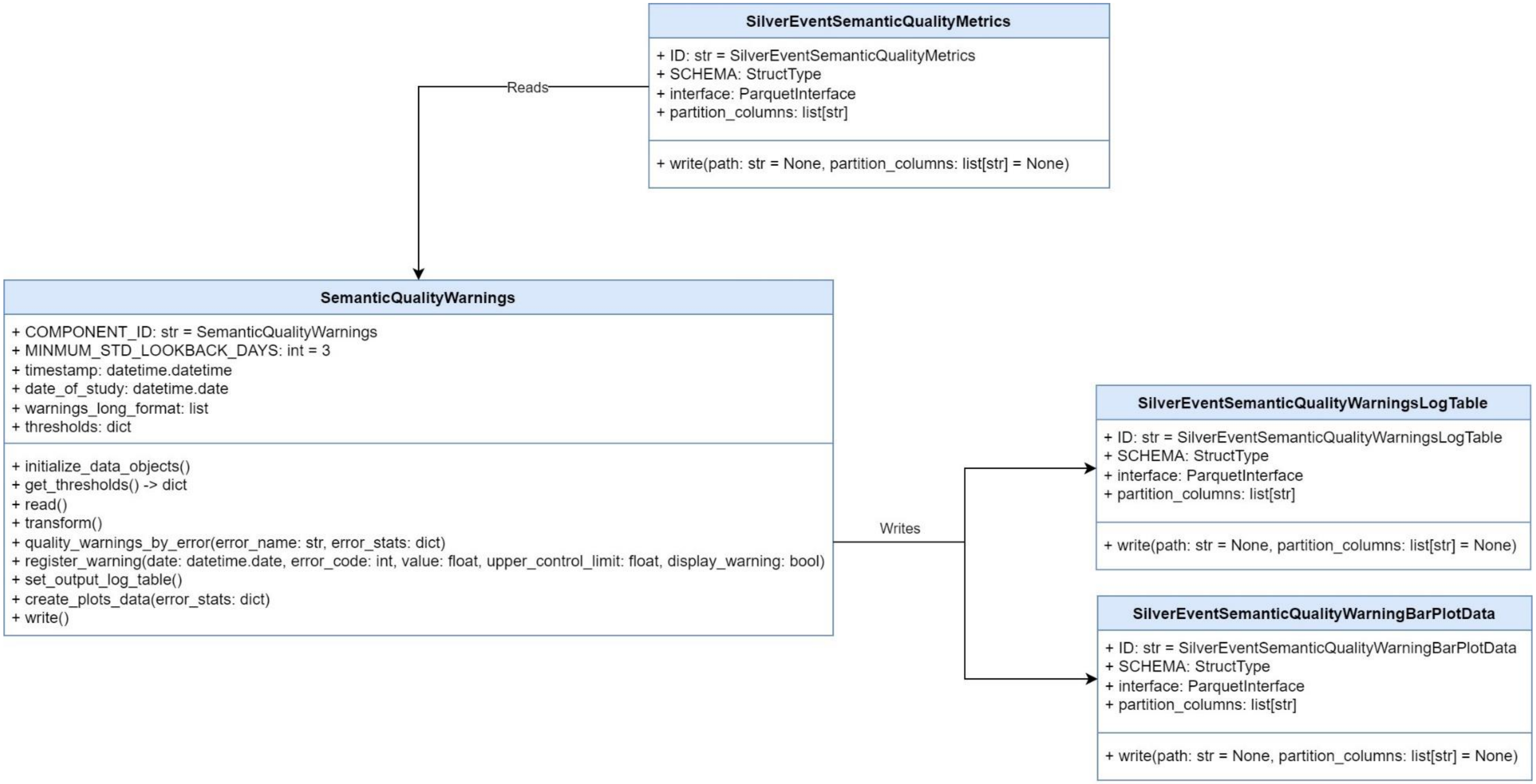
- **Data flow diagram (part I):**


Configuration file
(START HERE) Initialise data objects and check configuration
Determine furthest lookback date, D'
Device Semantic Quality Metrics
Read quality metrics
Default thresholds
Are thresholds provided via configuration?
No
Yes
Fall back to default thresholds when missing
Thresholds
Is there data for the study date D?
No
END
Yes
Semantic Quality Metrics D', ..., D-1
Semantic Quality Metrics D

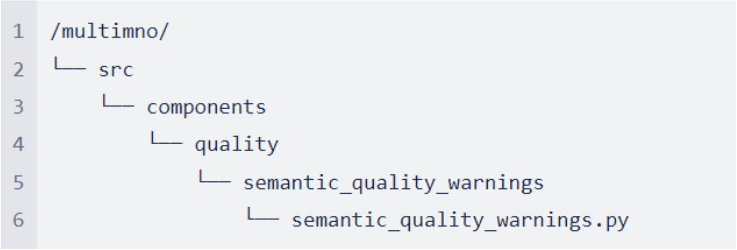
- **Data flow diagram (part II):**


Semantic Quality Metrics D', ..., D-1
Semantic Quality Metrics D
Thresholds
Put together plot data
Bar Plot Data
Compute percentage of each flag error type in all dates present
Flag error counts and percentages
For each flag error type
Is there data for all lookback period dates?
No
Yes
Set *display_warning* to False
Is the lookback period 3 or more days long?
No
Yes
Compute average and sample standard deviation of percentage over lookback period
Log the study date percentage and *display_warning*
Set minimum percentage parameter as threshold
Compute upper control limit as threshold
Is the study date percentage larger than the threshold?
No
Yes
Set *display_warning* to False
Set *display_warning* to True
Log the study date percentage, threshold and *display_warning*
Warning Log Table
Event Data at Device Level Semantic Quality Warnings Log Table

- **Class diagram:**


SilverEventSemanticQualityMetrics
+ ID: str = SilverEventSemanticQualityMetrics
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
Reads
SemanticQualityWarnings
+ COMPONENT_ID: str = SemanticQualityWarnings
+ MINMUM_STD_LOOKBACK_DAYS: int = 3
+ timestamp: datetime.datetime
+ date_of_study: datetime.date
+ warnings_long_format: list
+ thresholds: dict
+ initialize_data_objects()
+ get_thresholds() -> dict
+ read()
+ transform()
+ quality_warnings_by_error(error_name: str, error_stats: dict)
+ register_warning(date: datetime.date, error_code: int, value: float, upper_control_limit: float, display_warning: bool)
+ set_output_log_table()
+ create_plots_data(error_stats: dict)
+ write()
Writes
SilverEventSemanticQualityWarningsLogTable
+ ID: str = SilverEventSemanticQualityWarningsLogTable
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
SilverEventSemanticQualityWarningBarPlotData
+ ID: str = SilverEventSemanticQualityWarningBarPlotData
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:



```
/multimno/
└── src
    └── components
        └── quality
            └── semantic_quality_warnings
                └── semantic_quality_warnings.py
```

  - semantic_quality_warnings.py contains one class named SemanticQualityWarnings which is a subclass of Component. It overrides the following methods:
    - The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - The transform method handles all the logic behind the component.
    - The write method writes the quality warnings log table containing the computed warnings. It also writes into parquet files the data required to produce the defined plots.
  - The SemanticQualityWarnings also has the following methods:
    - get_thresholds handles the logic of reading configuration-specified thresholds and the usage of default threshold whenever a specific threshold value is not specified.
    - quality_warnings_by_error: method that handles the logic for computing the necessary statistics and raising a warning for a specific error flag in the study date.
    - register_warning is a method that abstracts away the creation of a warning in the log table, taking as arguments all necessary information and putting it in the correct format.
    - set_output_log_table formats the warnings into the expected table format.
    - create_plots_data gathers and formats the data required to produce the necessary plots of the component.

5.2.11 DEVICEACTIVITYSTATISTICS

5.2.11.1 MODULE DESCRIPTION

- **Module Name:** DeviceActivityStatistics
- **Objectives:** This module uses data on individual devices after and produces metrics to assess the usability of the devices for specific procedures or use cases based on the activity statistics.
- **Functionality:**
- **Data Inputs and Outputs:**
  - Input:
    I.16 MNO Event Data – Semantically Cleaned
    I.8 Cell Locations with Physical Properties – Cleaned
  - Outputs:
    I.19 Device Activity Statistics

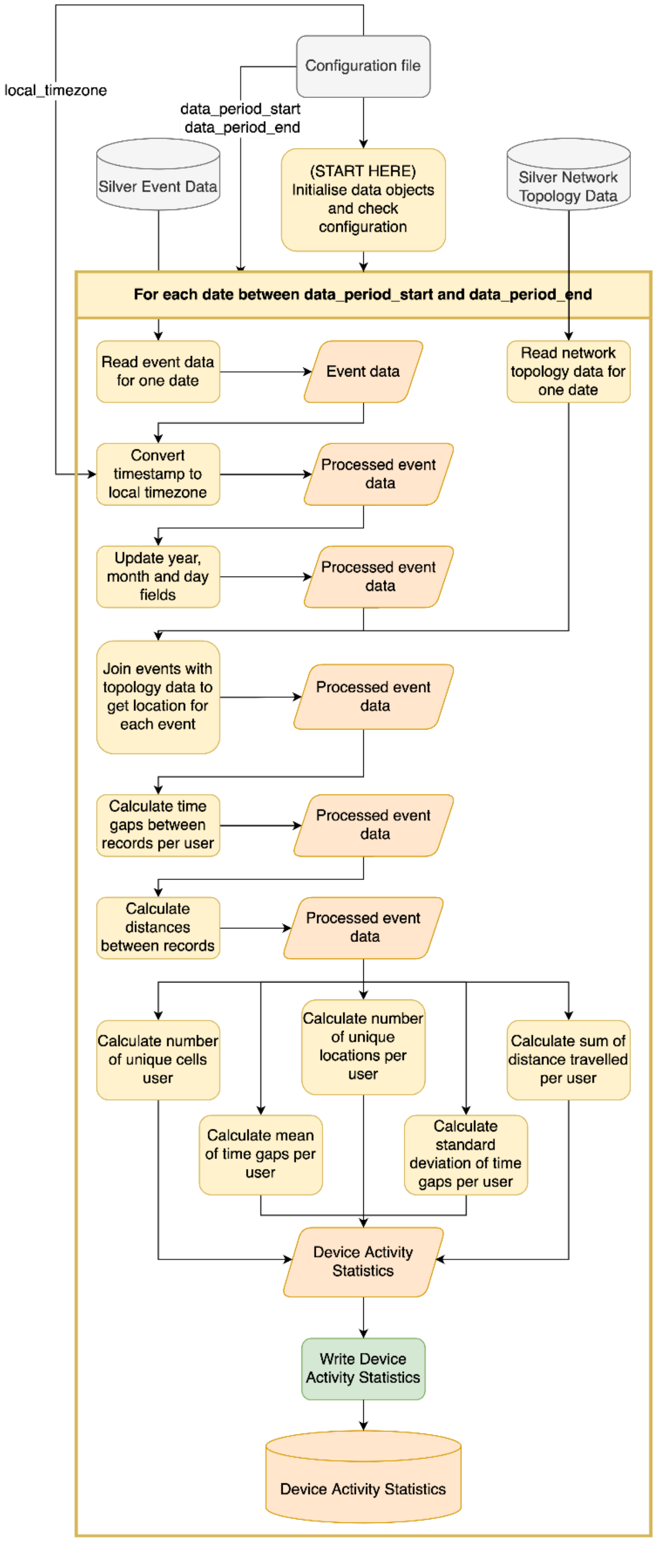
5.2.11.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
  The data is processed in one-day chunks in the local timezone. As the data is saved in UTC, the first

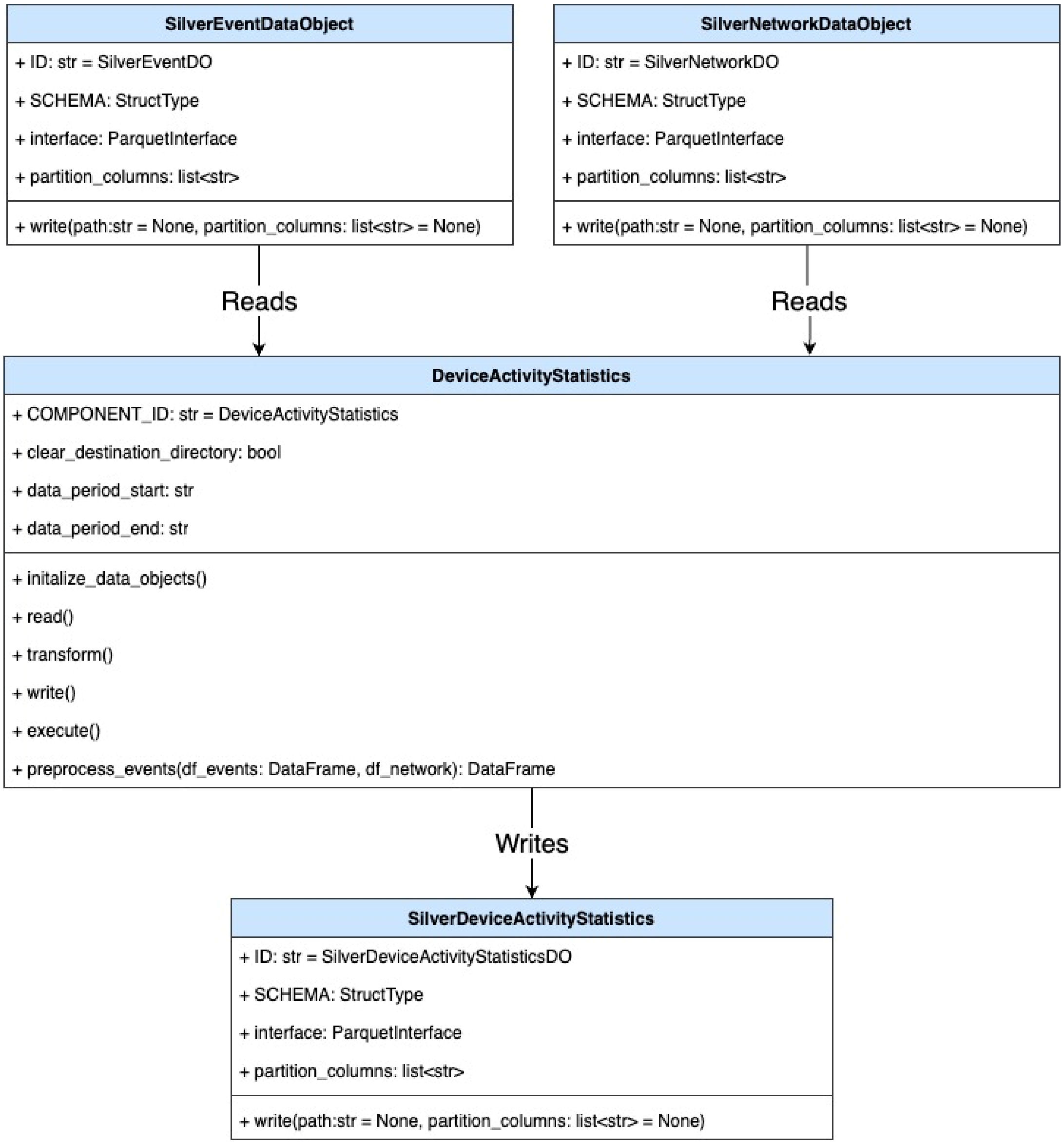
step is to calculate the times in UTC that need to be read. All of the following steps are run for each date in the data:

- Preprocess events data:
    - Timestamp is converted to local time.
    - Year, month and day fields are updated according to new timestamp.
    - Events data is joined with topology data to get locations of cells where the events happened.
    - The data is ordered by user and timestamp.
    - The time gaps between records are calculated per user.
    - Geometry (point) columns are created for the current cell location and location of next cell.
    - The distance between these two geometries is calculated.
- Calculate number of unique cells per user.
- Calculate number of unique locations per user (based on cell locations or lat/lon of event if cell_id not available).
- Calculate distance between records per user (based on cell locations or lat/lon of event if cell_id not available).
- Calculate number of unique hours present in data per user.
- Calculate mean and standard deviation of time gaps per user.

- **Data flow diagram:**

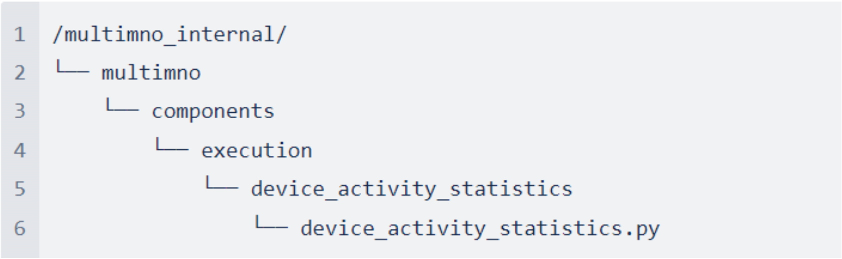

Configuration file
local_timezone
data_period_start
data_period_end
Silver Event Data
(START HERE) Initialise data objects and check configuration
Silver Network Topology Data
For each date between data_period_start and data_period_end
Read event data for one date
Event data
Read network topology data for one date
Convert timestamp to local timezone
Processed event data
Update year, month and day fields
Processed event data
Join events with topology data to get location for each event
Processed event data
Calculate time gaps between records per user
Processed event data
Calculate distances between records
Processed event data
Calculate number of unique cells user
Calculate number of unique locations per user
Calculate sum of distance travelled per user
Calculate mean of time gaps per user
Calculate standard deviation of time gaps per user
Device Activity Statistics
Write Device Activity Statistics
Device Activity Statistics

- **Class diagram:**


SilverEventDataObject
+ ID: str = SilverEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
SilverNetworkDataObject
+ ID: str = SilverNetworkDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
Reads
Reads
DeviceActivityStatistics
+ COMPONENT_ID: str = DeviceActivityStatistics
+ clear_destination_directory: bool
+ data_period_start: str
+ data_period_end: str
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
+ preprocess_events(df_events: DataFrame, df_network): DataFrame
Writes
SilverDeviceActivityStatistics
+ ID: str = SilverDeviceActivityStatisticsDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)

- **Code Structure:**
  The code structure follows the format set by the core package, and the general repository structure.

  The location of the module script in the repository is as follows:



```
/multimno_internal/
└── multimno
    └── components
        └── execution
            └── device_activity_statistics
                └── device_activity_statistics.py
```

device_activity_statistics.py contains one class named DeviceActivityStatistics which is a subclass of Component.

The EventCleaning class overwrites __init__, transform and execute in the Component class.

> __init__ method initialises the data objects and reads the necessary values from config file. transform performs all necessary transformations and calculation of activity statistics for daily data. transform contains calls to many other smaller functions that perform the actual data manipulation. execute is responsible for calling read, write and transform for each unique date in the dataset. The processing is done date-by-date. Only the data from one date is being processed at any given time.

5.2.12 CONTINUOUSTIMESEGMENTATION

5.2.12.1 MODULE DESCRIPTION

- **Module Name:** ContinuousTimeSegmentation
- **Objectives:** responsible for aggregating event data for each user into continuous time segments based on certain spatio-temporal conditions.
- **Functionality:** takes as input a configuration file, semantically cleaned event data, cell intersection groups and previously calculated time segments (only when available from executions from previous dates; if not available, the process calculates this information). It then processes user events for each date in the chosen data period and aggregates them into continuous time segments using cell intersection groups to determine events which are happening in nearby cells with overlapping coverage areas and so can belong to the same time segment. Segments are assigned with different states:
    - stay-the location of the device is known and the device is staying in one location for a certain period of time. Period of time is configuration parameter.
    - move - the device is moving from one location to the next; the location of the device is somewhere in between the two locations.
    - undetermined -the location of the device is known, but it is unclear whether or not the device is moving.
    - unknown the location of the device is unknown: there are no events for a certain (longer) period of time.

  Functionality is outlined in the software requirement specifications: 3.2.12 ContinuousTimeSegmentation
- **Data Inputs and Outputs:**
    - Inputs:
        - I.16 MNO Event Data – Semantically Cleaned
        - I.14 Cell Intersection Groups
        - I.20 Daily Continuous Time Segments (optional)
    - Outputs:
        - I.20 Daily Continuous Time Segments

5.2.12.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**

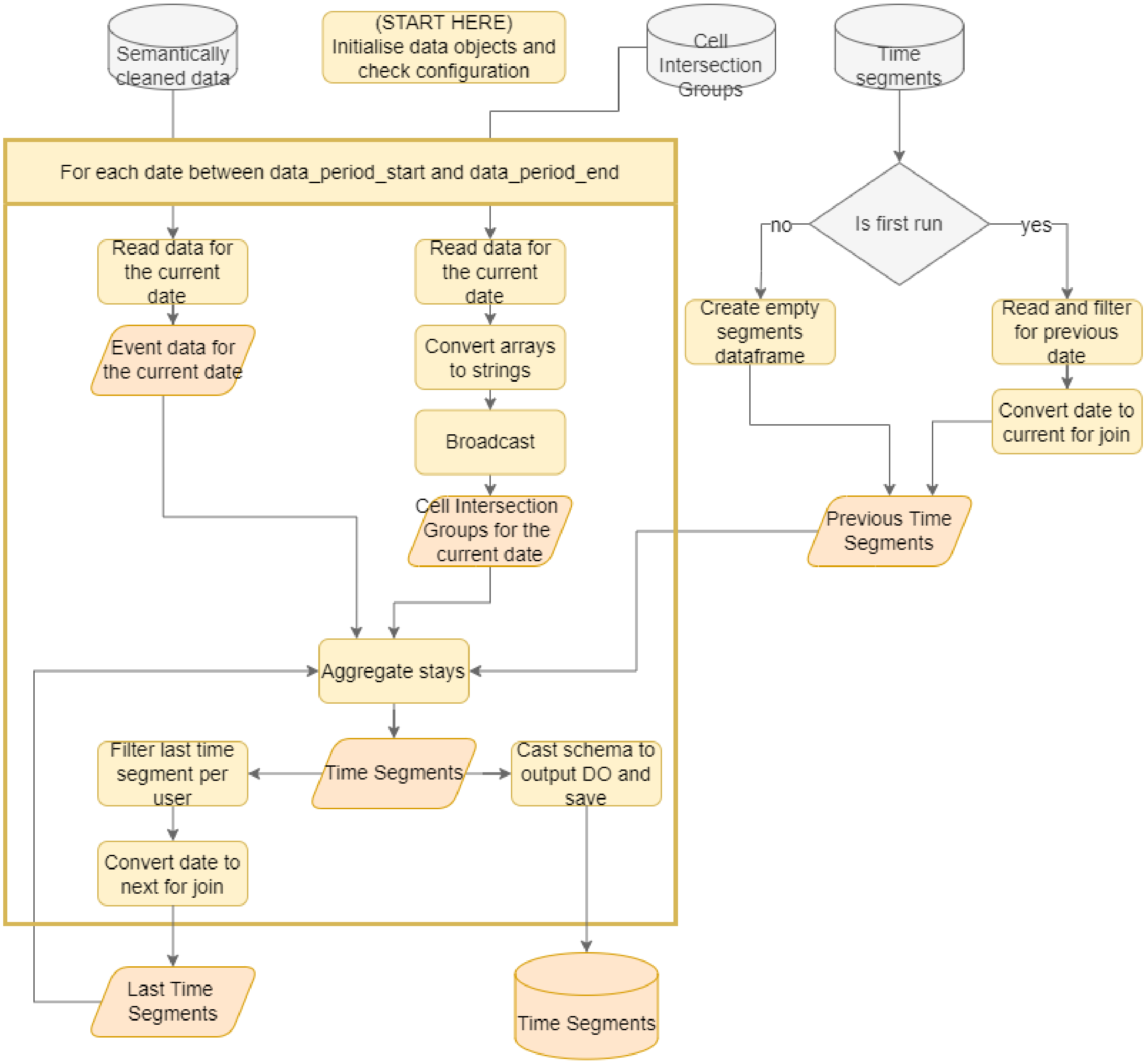
1. Initialisation. Read all necessary config parameters, check the availability of input data, read it in data objects. Check existence of input files for the dates within the date interval specified via configuration file, plus the previous (D-1) and posterior date (D+1) to each of these (D).
2. Prepare input datasets:
    1. If this is the first run of the component and no D-1 time segments are available, create an empty data frame with Time Segments Data Object schema. If CTS outputs are available for D-1, the last time segment is uploaded.
    2. If CTS outputs are available for D+1, the first segment is uploaded; otherwise, the first event for D+1 are considered together with 'D' events for the analysis of time segments.
    3. For previous time segment, convert year, month, date columns to the current processing date in order to properly join it with current date events dataset later.
    4. For previous time segment, convert user_id column to hex string from binary. This step is needed due to Pandas serialization/deserialization. Must be omitted when user_id will be stored as hex string instead of binary.
3. Start iterating over dates in date range between date period start and date period end, for each date do following input datasets preparations:
    1. Filter events dataset to include only events for current processing date and only for given semantic error flags. Default configuration includes only semantic error flag 0, so that only records without any semantic errors are included.
    2. Filter cell intersection groups to include only groups only for current processing date.
    3. In events dataset, convert user_id column to hex string from binary. This step is needed due to Pandas serialization/deserialization. Must be omitted when user_id will be stored as hex string instead of binary.
    4. Convert cells arrays column of cell intersection groups to string with comma separator for easier lookup and conversion to Pandas dataframe later
    5. Broadcast cell intersection groups dataframe to have it available on all worker nodes
4. Perform time segments aggregation. Group events dataframe by year, month, day, user_id_modulo partition key, user_id and cogroup it with last time segments dataframe also grouped by the same columns. Then apply Pandas UDF on cogroups of these datafames. Each cogroup is a Pandas dataframe with user events for the current processing date and last time segment for the same user in the previous date. The UDF function performs following steps:
    1. It initialises an empty segments list to store the time segments and a boolean is_first_ts to track if it's the first time segment.
    2. It calculates the start and end of the current and previous dates.
    3. It creates a Pandas dataframe intersection_pdf from the broadcasted intersection groups.
    4. It initializes the user info and the last time segment based on the presence of events and last segments for the current date:
        1. If events are present, but there is no last segment for the current user, creates time segment of type 'undetermined' for the whole previous date. Takes user_id, mcc, user_id_modulo columns from events dataframe
        2. If events and last segment for the current user are present, takes the last time segment as is. Takes user_id, mcc, user_id_modulo columns from events dataframe
        3. If there are no events for the current date, creates time segment of type 'undetermined' for the whole time period of the current date. Takes user_id, mcc, user_id_modulo columns from last time segment dataframe

5. If there are no events for the current date, append the only created segment to the final segments list.
6. If there are events for the current date, start iteration over events having the last time segment derived earlier as a current time segment. For each event:
    1. It initialises an empty dictionary to store the next time segment and an empty list to store time segments to be added to final time segments list.
    2. It extracts the timestamp and cell of the event.
    3. It handles first time segment. If it's the current time segment is the previous day's last time segment, It checks the state of this time segment and the time difference between the end of this current time segment and the first event in the current date:
        - If the state is 'undetermined' or 'stay' and the time difference is within the maximum missing stay time, or if the state is 'move' and the time difference is within the maximum missing move time, it creates a new time segment with the same cells, state. The start timestamp of the new time segment is the start of the current date, and the end timestamp is the timestamp of the first event.
        - If neither of these conditions are met, it creates a new time segment with an empty list of cells, state 'unknown'. The start timestamp of the new time segment is the start of the next date, and the end timestamp is the timestamp of the first event minus the padding time.
    4. It defines intersection status by checking if current segment cells are intersected with current even cell (their coverage areas overlaps).
    5. It processes current time segment with amending it or creating new time segments based on the following logic:
        - if there's an intersection and the time gap between the event timestamp and the end timestamp of the current time segment is within the acceptable range for a 'stay', it checks the state of the current time segment:
            - If the state is 'undetermined' or 'stay', it updates the current time segment with the new cell and the new end timestamp and sets the state to 'stay' if the time gap is more than the minimum time for a 'stay'.
            - If the state is 'move', it creates a new time segment with state 'undetermined' after the 'move' segment. If the time gap is big enough to assume that it's a 'stay', it changes the state to 'stay'. The current time segment and the new time segment are added to final time segments list.
        - If there's no intersection and the time gap is within the acceptable range for a 'move', it checks the state of the current time segment:
            - If the state is not 'unknown', it creates two new 'move' segments half of total time between time segment end timestamp and event timestamp each. It then adds first move time segment and the current time segment to the final list of time segments and sets the second move time segment as the current time segment.
        - If the state of the current time segment is 'unknown', it creates a new 'undetermined' time segment and adds the current time segment to the final time segments list.
        - Finally, if the time gap is too big, it creates new 'unknown' segment for the missing time and a new 'undetermined' time segment with start timestamp as event timestamp - pad time and event cell as segment cells. It adds current time segment and new 'unknown' time segment to the final time segments list and sets 'undetermined' time segment as current time segment.

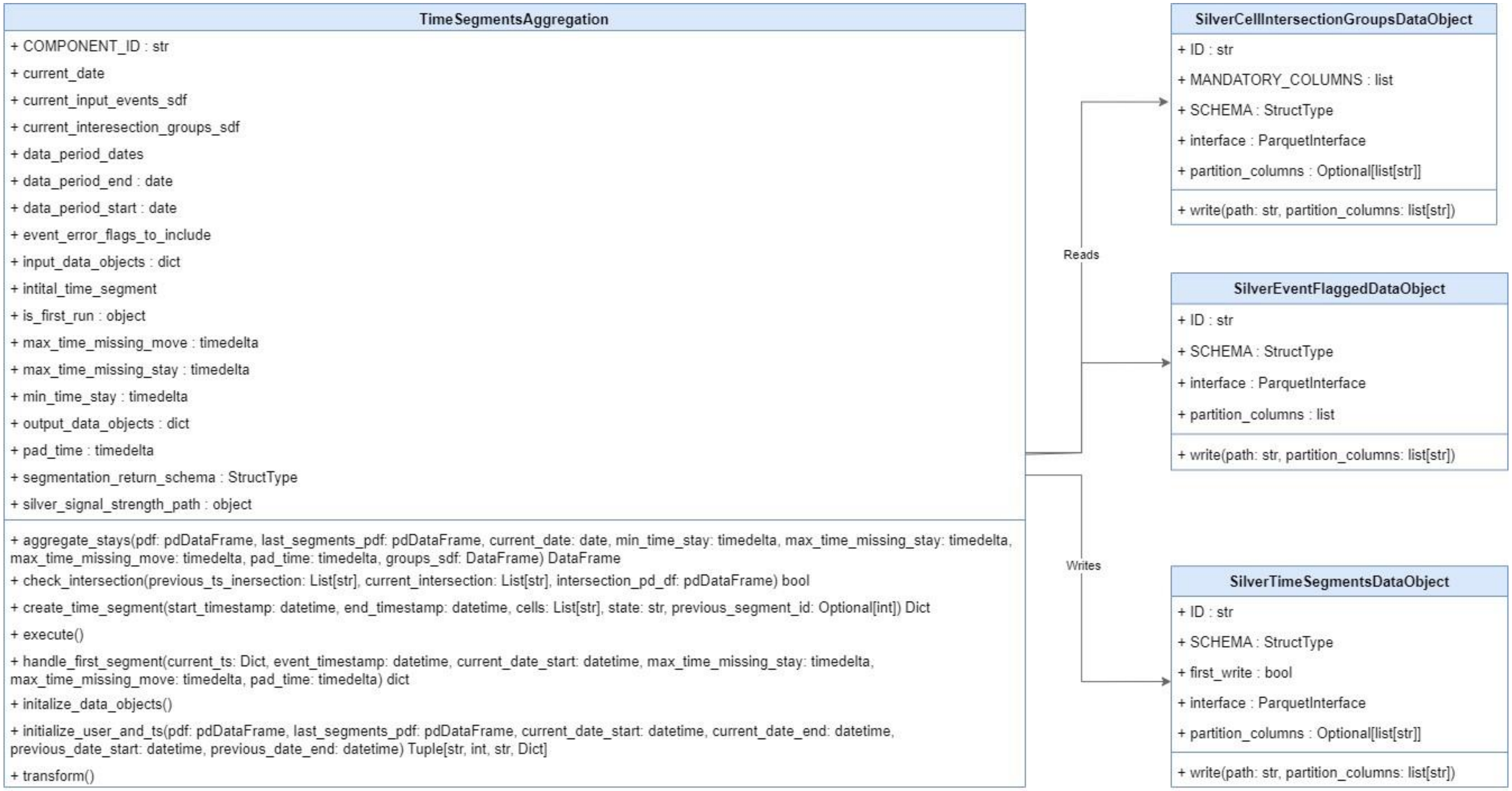
   6. After processing all events, it sets the is_last field of the last time segment to True and appends it to the final time segments list.
   7. Convert final time segments list to Pandas dataframe, assign user_id mcc, user_id_modulo columns and return.

5. Filter the last time segments for the current date from the time segments dataframe.
6. Convert year, month, date columns to the next processing date in order to properly join it with the next date events in the next iteration.
7. Extract year, month, day columns from the initial_timestamp of time segments.
8. Convert cell-grid dataset schema to match the output data object schema and save to storage partition by year, month, day, user_id_modulo.

- **Data flow diagram:**


Semantically cleaned data
(START HERE) Initialise data objects and check configuration
Cell Intersection Groups
Time segments
For each date between data_period_start and data_period_end
Read data for the current date
Read data for the current date
Is first run
no
yes
Event data for the current date
Convert arrays to strings
Create empty segments dataframe
Read and filter for previous date
Broadcast
Convert date to current for join
Cell Intersection Groups for the current date
Previous Time Segments
Aggregate stays
Filter last time segment per user
Time Segments
Cast schema to output DO and save
Convert date to next for join
Last Time Segments
Time Segments

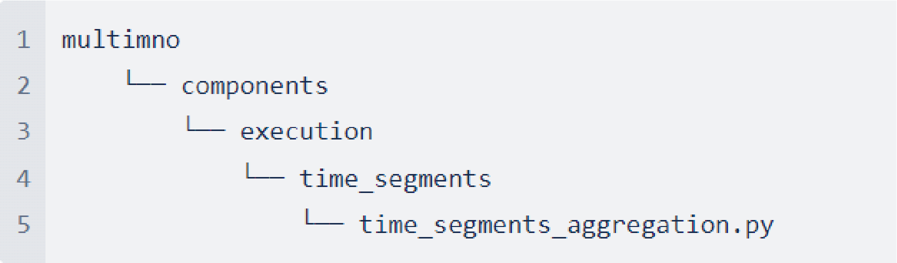
- **Class diagram:**


TimeSegmentsAggregation
+ COMPONENT_ID : str
+ current_date
+ current_input_events_sdf
+ current_interesection_groups_sdf
+ data_period_dates
+ data_period_end : date
+ data_period_start : date
+ event_error_flags_to_include
+ input_data_objects : dict
+ intital_time_segment
+ is_first_run : object
+ max_time_missing_move : timedelta
+ max_time_missing_stay : timedelta
+ min_time_stay : timedelta
+ output_data_objects : dict
+ pad_time : timedelta
+ segmentation_return_schema : StructType
+ silver_signal_strength_path : object
+ aggregate_stays(pdf: pdDataFrame, last_segments_pdf: pdDataFrame, current_date: date, min_time_stay: timedelta, max_time_missing_stay: timedelta, max_time_missing_move: timedelta, pad_time: timedelta, groups_sdf: DataFrame) DataFrame
+ check_intersection(previous_ts_inersection: List[str], current_intersection: List[str], intersection_pd_df: pdDataFrame) bool
+ create_time_segment(start_timestamp: datetime, end_timestamp: datetime, cells: List[str], state: str, previous_segment_id: Optional[int]) Dict
+ execute()
+ handle_first_segment(current_ts: Dict, event_timestamp: datetime, current_date_start: datetime, max_time_missing_stay: timedelta, max_time_missing_move: timedelta, pad_time: timedelta) dict
+ initalize_data_objects()
+ initialize_user_and_ts(pdf: pdDataFrame, last_segments_pdf: pdDataFrame, current_date_start: datetime, current_date_end: datetime, previous_date_start: datetime, previous_date_end: datetime) Tuple[str, int, str, Dict]
+ transform()
Reads
Writes
SilverCellIntersectionGroupsDataObject
+ ID : str
+ MANDATORY_COLUMNS : list
+ SCHEMA : StructType
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]
+ write(path: str, partition_columns: list[str])
SilverEventFlaggedDataObject
+ ID : str
+ SCHEMA : StructType
+ interface : ParquetInterface
+ partition_columns : list
+ write(path: str, partition_columns: list[str])
SilverTimeSegmentsDataObject
+ ID : str
+ SCHEMA : StructType
+ first_write : bool
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]
+ write(path: str, partition_columns: list[str])

- **Code Structure:**

The code structure follows the format set by the core package, and the general repository structure.

The location of the module script in the repository is as follows:



```
multimno
     └── components
          └── execution
               └── time_segments
                    └── time_segments_aggregation.py
```

time_segments_aggregation.py contains one class named ContinuousTimeSegmentation which is a subclass of Component.

The ContinuousTimeSegmentation class overrides transform and execute method of base Component class.

Execute method facilitates iteration over list of processing dates to manage all processing on daily batches. Transform method performs all necessary filtering and transformations of MNO event data to aggregate it to Time Segments by sequentially calling other methods that perform the actual data manipulation. The main method for Time Segments aggregation is aggregate_stays which is a Pandas UDF called on grouped Spark DataFrame.

5.2.13 DAILYPERMANENCESCORE

5.2.13.1 MODULE DESCRIPTION

- **Module Name:** DailyPermanenceScore
- **Objectives:** Given a definition of time intervals of the day, use events data to estimate each user's permanence time at each grid tile during each of the intervals.
- **Functionality:** needed functionalities are outlined in the software requirement specifications: 3.2.11 DailyPermanenceScore
- **Data Inputs and Outputs:**
    - Input:
        - I.13 Cell Footprints
        - I.16 MNO Event Data – Semantically Cleaned
    - Output:
        - I.21 Daily Permanence Score

5.2.13.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:** The raw data is processed for the set of dates within a date interval specified via configuration file. It is assumed that the raw input data is partitioned by year, month, day columns. The key processes are described below:

0) Check available data and load input data objects

- Check existence of input files for the dates within the date interval specified via configuration file, plus the previous (D-1) and posterior date (D+1) to each of these (D).
- Exit process if one or more input files are not found at their corresponding path.
- Iterate over dates within the date interval specified via configuration file. Each calculation after this will be performed once for each of the dates.
- Load Cell Footprint Data Objects and Semantically Cleaned Event Data Objects for the current date (D), for the immediately previous date (D-1) and for the date after (D+1).
- Remove unneeded columns from Semantically Cleaned Event Data Objects.
- Filter out events with semantic flag warnings from Semantically Cleaned Event Data Objects.
- Load user events:
    - reach the last event corresponding to one user for date D-1.
    - reach all events corresponding to the same user for date D.
    - reach the first event corresponding to the same user for date D+1.
    - combine all these, sorted by event time, into the same table.

1) Differentiate events associated to ‘permanence’ and events associated to ‘moves’

- Join user events table with cell footprint data for date D. Use key = cell_id. Generate new column ‘grid_ids’ in user events table.
- Search for ‘cell_id’ value from first user event from the user events table (which came from D-1) in cell footprint data for date D-1 and assign value to new column 'grid_ids' for first row.
- Search for ‘cell_id’ value from last user event from the user events table (which came from D+1) in cell footprint data for date D+1 and assign value to new column 'grid_ids' for last row.
- Add new ‘is_move’ boolean column to user events table. Initialise with False values.
- Generate 3-row window containing first 3 rows of the user events table.
- Calculate distance between cells in the 3-row window. Current distance algorithm computes all distances between cell 1 grids vertices and cell 2 grids vertices and finds minimum (final optimised algoritm for calculation of distance between grid tiles of cells to be defined).
- Calculate time difference (Δt) between 1st and last (3rd) event in the 3-row window.
- Calculate the maximum value of the distance from the 1st event’s cell and the 2nd event’s cell and the sumation of distances from 1st to 2nd and from 2nd to 3rd: d_max = max(d(1,3), d(1,2) + d(2,3))
- Calculate the speed (s) resulting from dividing d_max by Δt.
- If the speed (s) is higher than ‘max_vel_thresh', assign 'True’ value to the intermediate 3-row window event for the ‘is_move’ column.
- Move the 3-row window to the next position and repeat until the end of the user events table is reached.

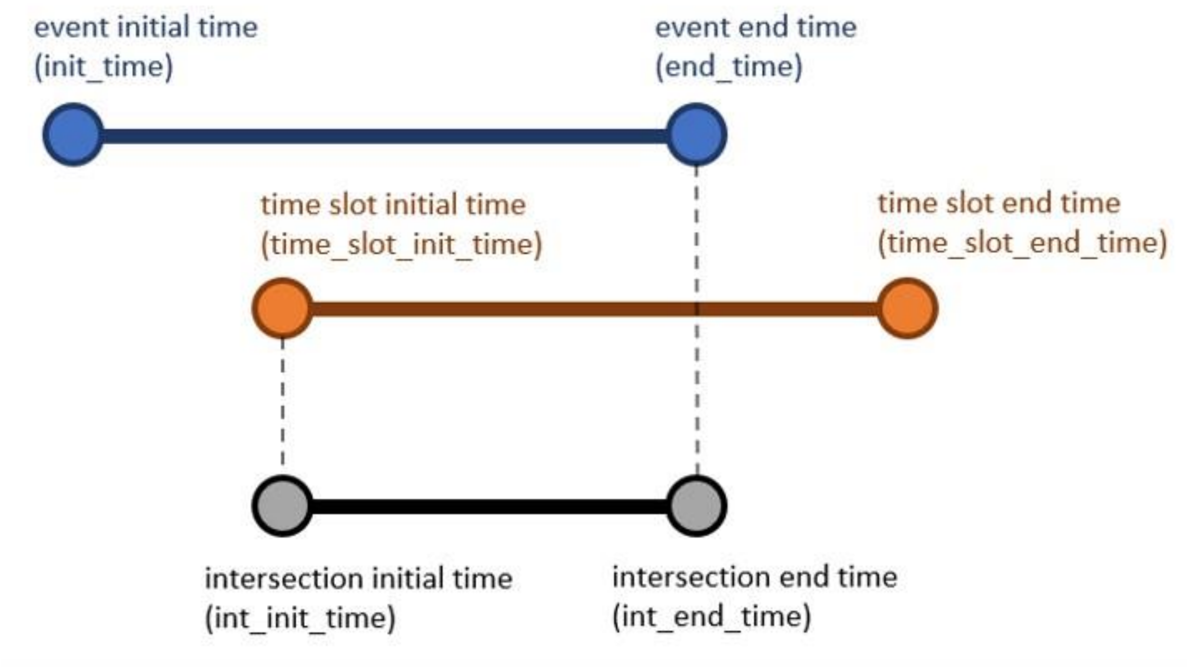
2) Assign initial and end times to each event classified as stay’

- Generate 3-row window containing the first 3 rows of the user events table.
- Add ‘init_time’ and ‘end_time’ columns (timestamp type) to user events table. Initialise with null values.
- Calculate time difference (Δt) between the event in the centre of the 3-row window and the previous event (with ‘timestamp’ column).
- If the previous event and the current event happen in a different cell ('cell_id' column has different values the respective rows), then check if the time difference (Δt) is higher than ‘max_time_thresh’:

    - If $\Delta t \leq$ ‘max_time_thresh’: ‘init_time’ value for the middle event is equal to the average between the ‘timestamp' value of the previous row and the 'timestamp’ value of the current row.
    - If $\Delta t >$ 'max_time_thresh': ‘init_time’ value for the middle event is equal to the ‘timestamp’ value of the current row minus 'max_time_thresh’ / 2.
- If the previous event and the current event happen in the same cell ('cell_id' column has the same value for both rows), then check if the time interval between the ‘timestamp’ of the previous event and the ‘timestamp’ of the current event intersects with the night interval. If the interval intersects with the night interval: ‘time_threshold’ = ‘max_time_thresh_night’. Else, ‘time_threshold’ = ‘max_time_thresh_day’:
    - If $\Delta t \leq$ ‘time_threshold’: ‘init_time’ value for the middle event is equal to the average between the ‘timestamp' value of the previous row and the 'timestamp’ value of the current row.
    - If $\Delta t >$ ‘time_threshold’: ‘init_time’ value for the middle event is equal to the ‘timestamp’ value of the current row minus 'time_threshold’ / 2.
- Calculate time difference (Δt) between the event in the centre of the 3-row window and the next event (with ‘timestamp’ column) and follow the analogous process to the one described just above for the previous event (in this case, filling in the ‘end_time’ values).
- Move the 3-row window to the next position and repeat until the end of the user events table is reached.

3) Intersect ‘permanence’ times with specified intervals

- Filter out moves: keep only those rows for which ‘is_move’ = False in the user events table.
- Define a list of equal intervals in which the day is split based on the ‘time_slot_number’ field from the configuration file. This number can only take values 24, 48 or 96, which result in equal intervals of 60, 30 and 15 minutes, respectively. Create a table with these time slots.
- Cross join user events table with time slots.
- For each row of the crossed table, calculate the maximum value of ‘init_time’ and ‘time_slot_init_time’ columns and the minimum value of ‘end_time’ and ‘time_slot_end_time’ columns. If the maximum init time is lower than the minimum end time, there is an intersection.
- Calculate subtraction to obtain duration and of the permanence.
- For each user and time slot, calculate the total time the user has performed a permanence. Subtract this value from the time length of the time slot to obtain the time duration during which the use has not been observed performing a permanence. Set this table aside for the moment, discarding rows where the ‘unknown’ duration is 0. This is the ‘unknown’ table.
- In the original cross table, discard the rest of the rows in which duration is 0.



4) Calculate 'permanence' times at each grid tile in each specified interval for each user

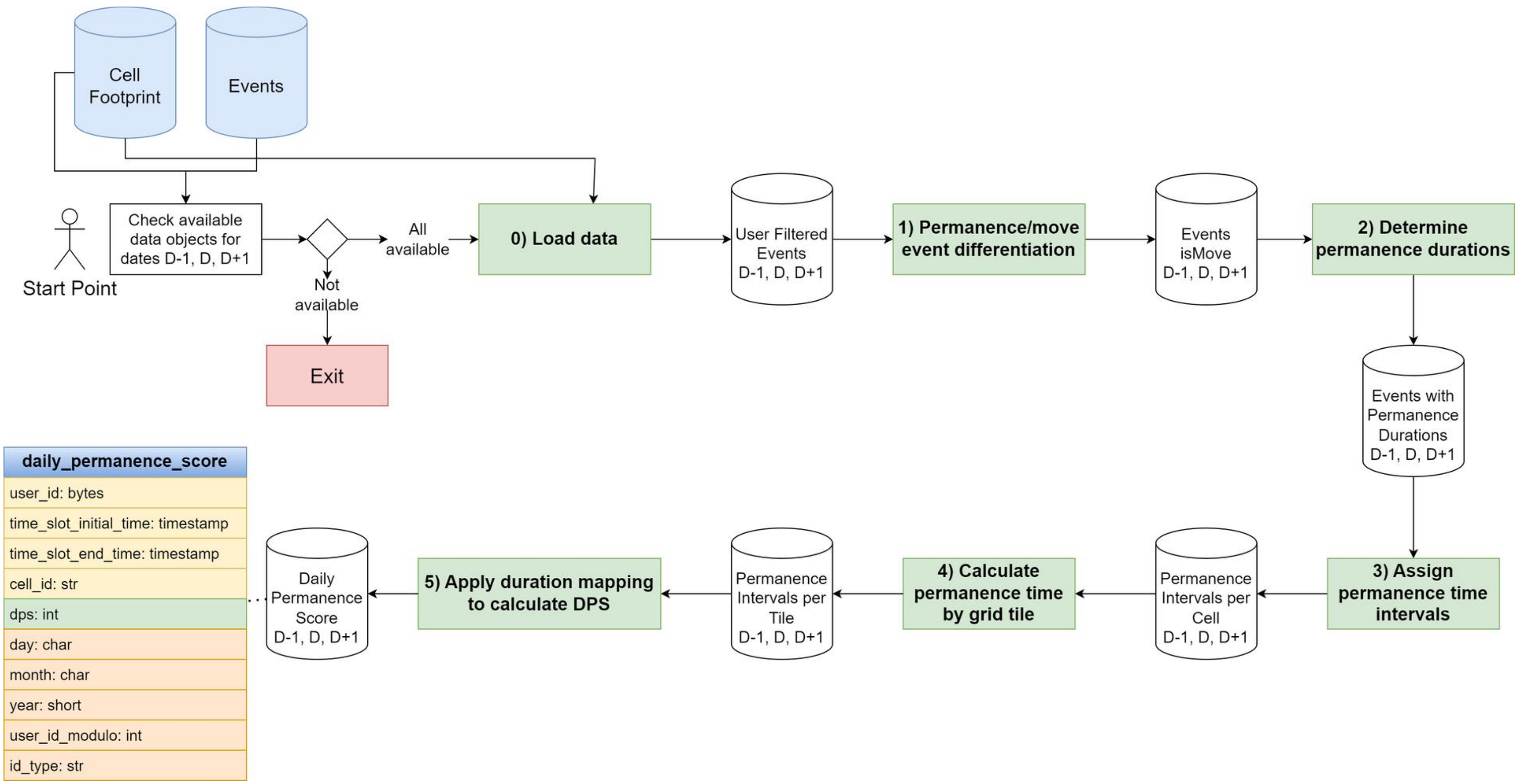
- Explode table: repeat each row as many times as tiles in the 'grid_ids' list.
- Group table by 'grid_id' and sum the 'int_duration' column values.
- Concatenate this new table with the 'unknown' duration table, by considering that the latter refers to a grid tile with its grid_id equal to 'unknown' and is equivalent to the permanence time in a grid tile with that ID.

5) Apply duration mapping to calculate DPS (Daily Permanence Score)

- From the permanence duration in a grid tile, create a new column 'dps' set to 0 if the user presents a permanence strictly less than half the duration of the time slot in that tile, or set it to 1 if the user presents a permanence at least half the duration of the time slot in that tile.

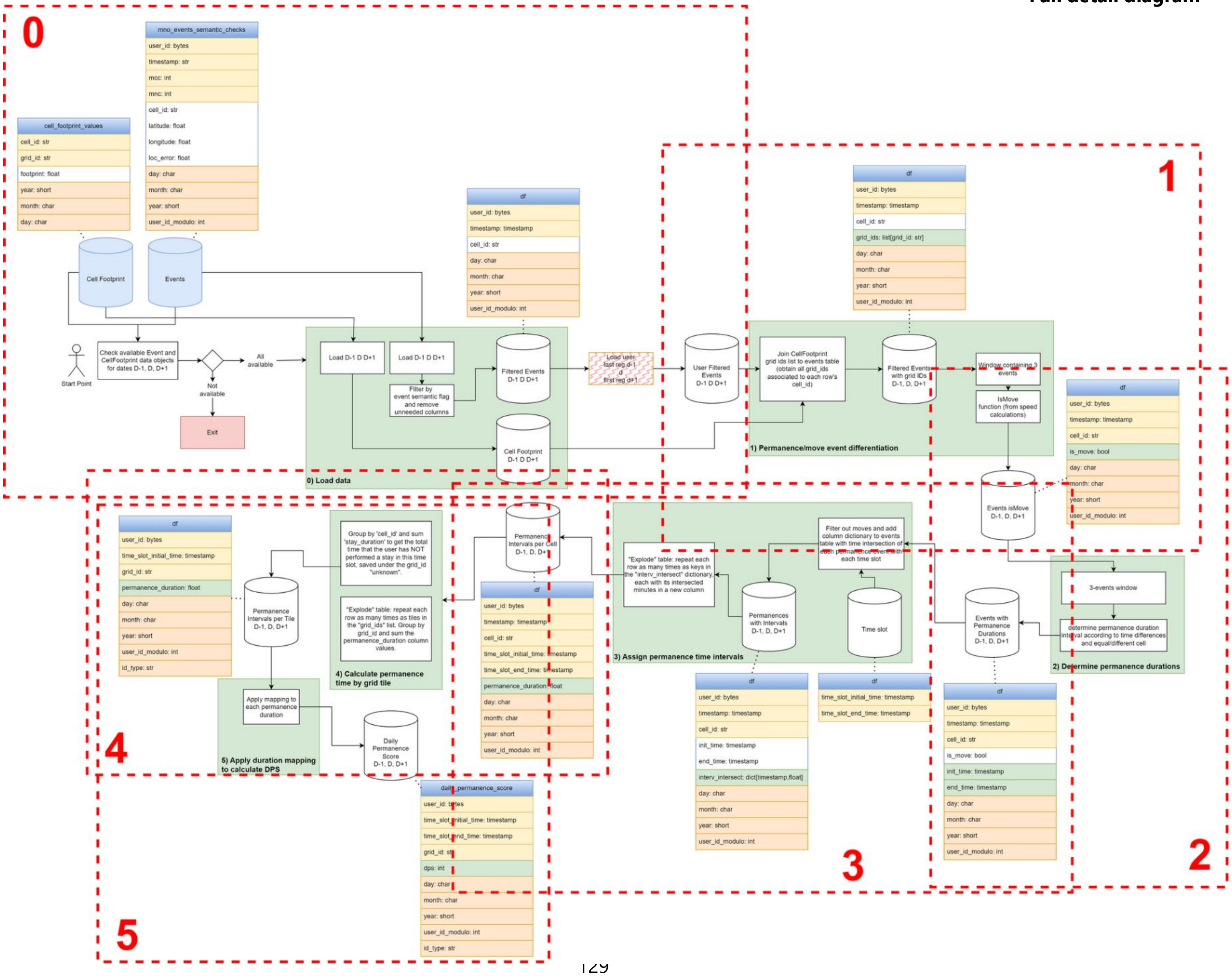
- **Data flow diagram:**

Simplified high-level diagram


Cell Footprint
Events
Start Point
Check available data objects for dates D-1, D, D+1
All available
Not available
Exit
0) Load data
User Filtered Events D-1, D, D+1
1) Permanence/move event differentiation
Events isMove D-1, D, D+1
2) Determine permanence durations
Events with Permanence Durations D-1, D, D+1
3) Assign permanence time intervals
Permanence Intervals per Cell D-1, D, D+1
4) Calculate permanence time by grid tile
Permanence Intervals per Tile D-1, D, D+1
5) Apply duration mapping to calculate DPS
Daily Permanence Score D-1, D, D+1

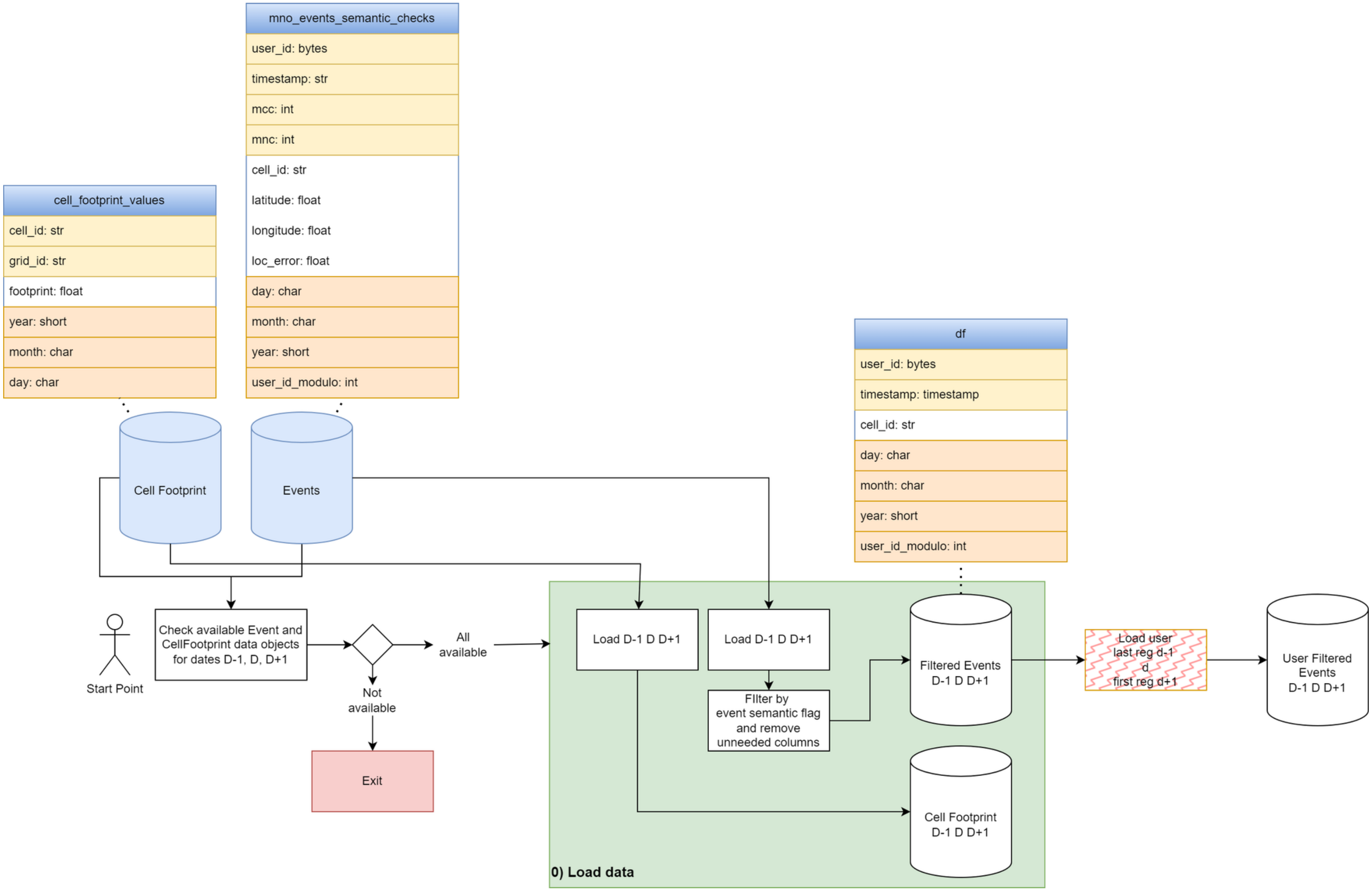
daily_permanence_score
user_id: bytes
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
cell_id: str
dps: int
day: char
month: char
year: short
user_id_modulo: int
id_type: str

Full detail diagram


0
mno_events_semantic_checks
user_id: bytes
timestamp: str
mcc: int
mnc: int
cell_id: str
latitude: float
longitude: float
loc_error: float
day: char
month: char
year: short
user_id_modulo: int
cell_footprint_values
cell_id: str
grid_id: str
footprint: float
year: short
month: char
day: char
Cell Footprint
Events
Start Point
Check available Event and CellFootprint data objects for dates D-1, D, D+1
All available
Not available
Exit
Load D-1 D D+1
Load D-1 D D+1
Filter by event semantic flag and remove unneeded columns
df
user_id: bytes
timestamp: timestamp
cell_id: str
day: char
month: char
year: short
user_id_modulo: int
Filtered Events D-1 D D+1
Cell Footprint D-1 D D+1
0) Load data
Load user last reg d-1 d first reg d+1
User Filtered Events D-1 D D+1
1
df
user_id: bytes
timestamp: timestamp
cell_id: str
grid_ids: list[grid_id: str]
day: char
month: char
year: short
user_id_modulo: int
Join CellFootprint grid ids list to events table (obtain all grid_ids associated to each row's cell_id)
Filtered Events with grid IDs D-1, D, D+1
Window containing 3 events
IsMove function (from speed calculations)
1) Permanence/move event differentiation
Events isMove D-1, D, D+1
df
user_id: bytes
timestamp: timestamp
cell_id: str
is_move: bool
day: char
month: char
year: short
user_id_modulo: int
3-events window
determine permanence duration interval according to time differences and equal/different cell
2) Determine permanence durations
Events with Permanence Durations D-1, D, D+1
df
user_id: bytes
timestamp: timestamp
cell_id: str
is_move: bool
init_time: timestamp
end_time: timestamp
day: char
month: char
year: short
user_id_modulo: int
2
Filter out moves and add column dictionary to events table with time intersection of each permanence event with each time slot
Time slot
df
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
Permanences with Intervals D-1, D, D+1
"Explode" table: repeat each row as many times as keys in the "interv_intersect" dictionary, each with its intersected minutes in a new column
3) Assign permanence time intervals
df
user_id: bytes
timestamp: timestamp
cell_id: str
init_time: timestamp
end_time: timestamp
interv_intersect: dict[timestamp,float]
day: char
month: char
year: short
user_id_modulo: int
3
Permanence Intervals per Cell D-1, D, D+1
df
user_id: bytes
timestamp: timestamp
cell_id: str
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
permanence_duration: float
day: char
month: char
year: short
user_id_modulo: int
Group by 'cell_id' and sum 'stay_duration' to get the total time that the user has NOT performed a stay in this time slot, saved under the grid_id "unknown".
"Explode" table: repeat each row as many times as tiles in the "grid_ids" list. Group by grid_id and sum the permanence_duration column values.
4) Calculate permanence time by grid tile
Permanence Intervals per Tile D-1, D, D+1
df
user_id: bytes
time_slot_initial_time: timestamp
grid_id: str
permanence_duration: float
day: char
month: char
year: short
user_id_modulo: int
id_type: str
Apply mapping to each permanence duration
5) Apply duration mapping to calculate DPS
Daily Permanence Score D-1, D, D+1
4
daily_permanence_score
user_id: bytes
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
grid_id: str
dps: int
day: char
month: char
year: short
user_id_modulo: int
id_type: str
5

Step-by-step diagrams:

STEP 0



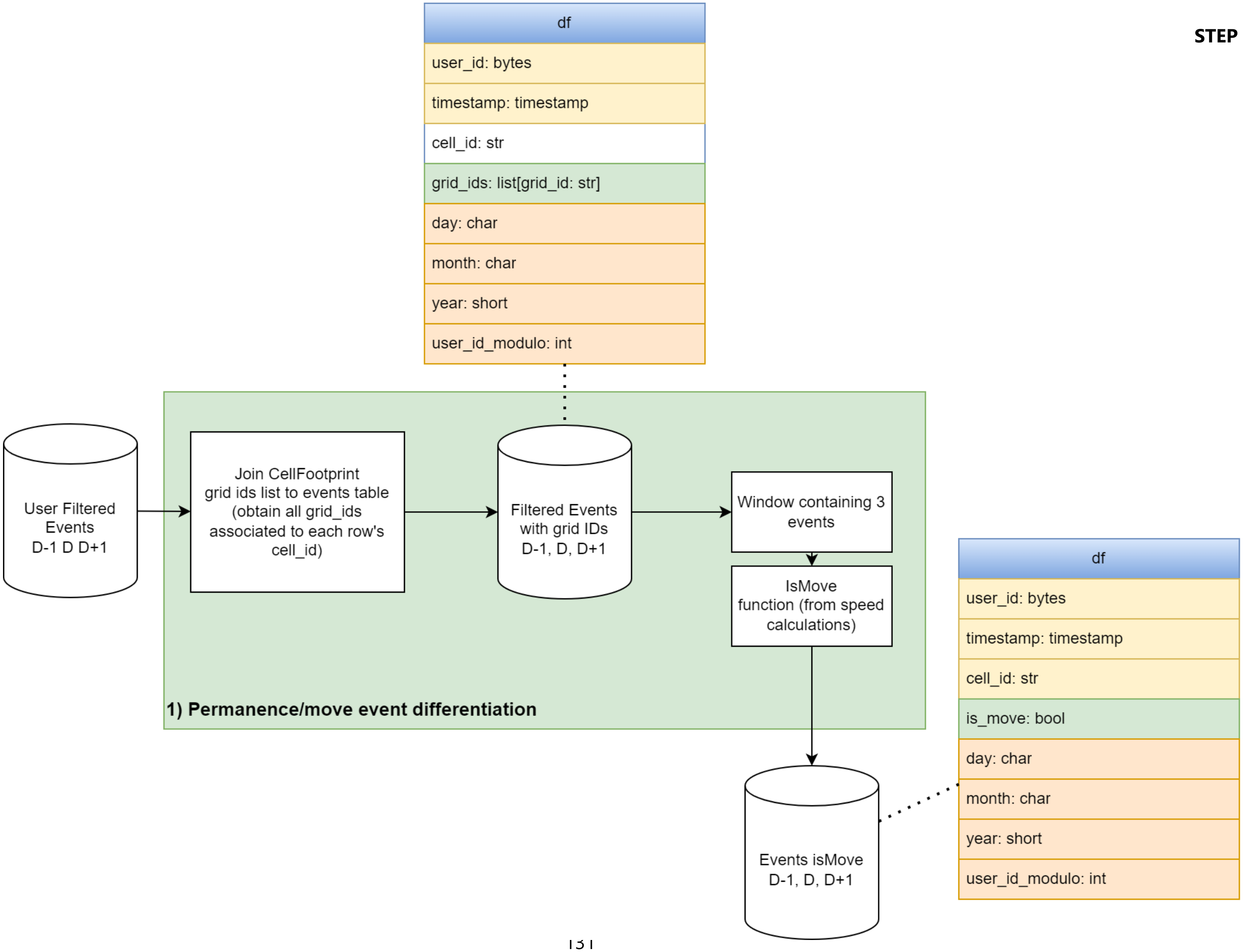
mno_events_semantic_checks
user_id: bytes
timestamp: str
mcc: int
mnc: int
cell_id: str
latitude: float
longitude: float
loc_error: float
day: char
month: char
year: short
user_id_modulo: int

cell_footprint_values
cell_id: str
grid_id: str
footprint: float
year: short
month: char
day: char

df
user_id: bytes
timestamp: timestamp
cell_id: str
day: char
month: char
year: short
user_id_modulo: int

Cell Footprint
Events
Start Point
Check available Event and CellFootprint data objects for dates D-1, D, D+1
All available
Not available
Exit
Load D-1 D D+1
Load D-1 D D+1
FIlter by event semantic flag and remove unneeded columns
Filtered Events D-1 D D+1
Cell Footprint D-1 D D+1
0) Load data
Load user last reg d-1 d first reg d+1
User Filtered Events D-1 D D+1

STEP 1

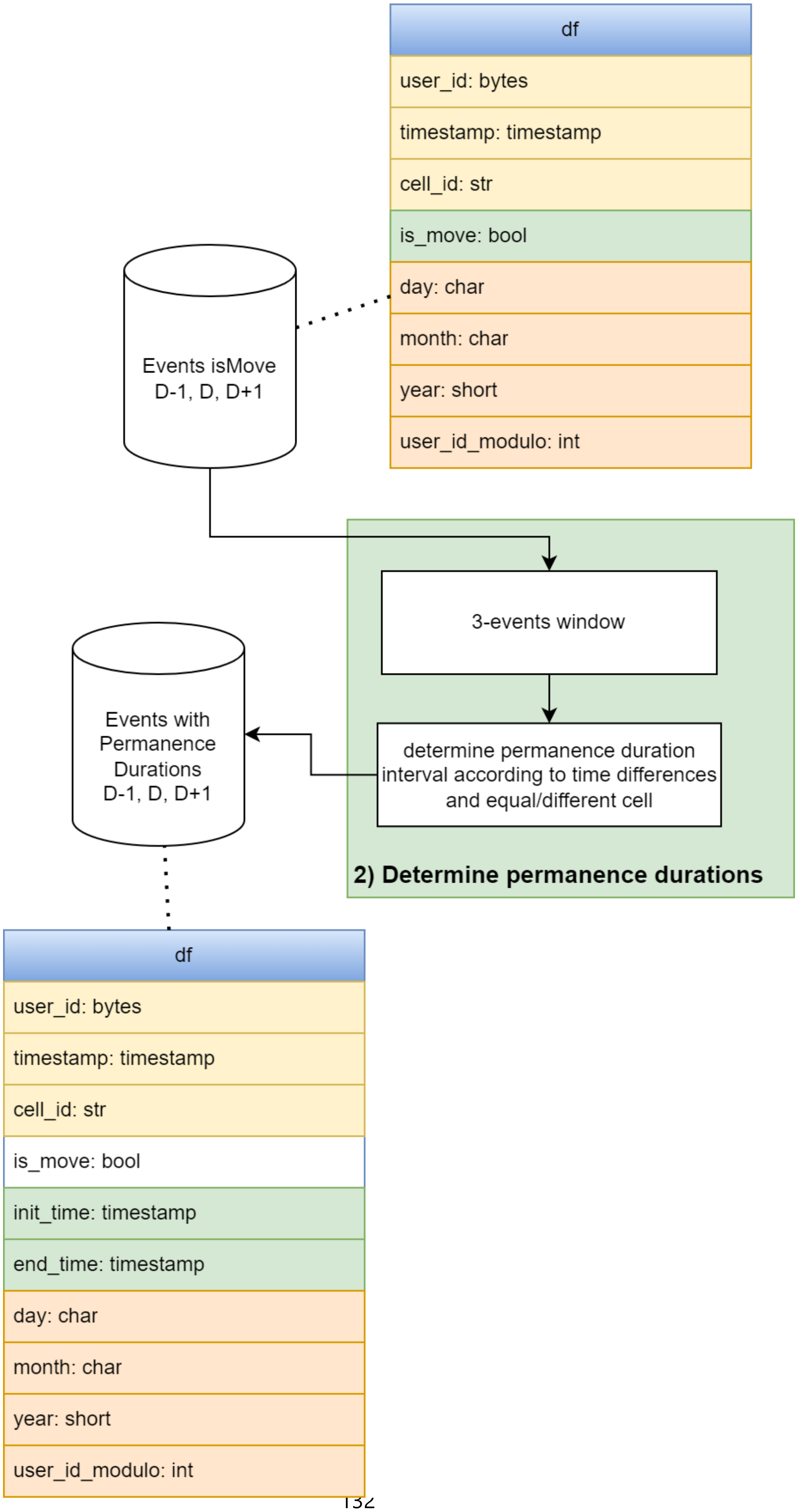


df
user_id: bytes
timestamp: timestamp
cell_id: str
grid_ids: list[grid_id: str]
day: char
month: char
year: short
user_id_modulo: int

User Filtered Events
D-1 D D+1
Join CellFootprint grid ids list to events table (obtain all grid_ids associated to each row's cell_id)
Filtered Events with grid IDs
D-1, D, D+1
Window containing 3 events
IsMove function (from speed calculations)
1) Permanence/move event differentiation
Events isMove
D-1, D, D+1

df
user_id: bytes
timestamp: timestamp
cell_id: str
is_move: bool
day: char
month: char
year: short
user_id_modulo: int

STEP 2



df
user_id: bytes
timestamp: timestamp
cell_id: str
is_move: bool
day: char
month: char
year: short
user_id_modulo: int

Events isMove
D-1, D, D+1

3-events window

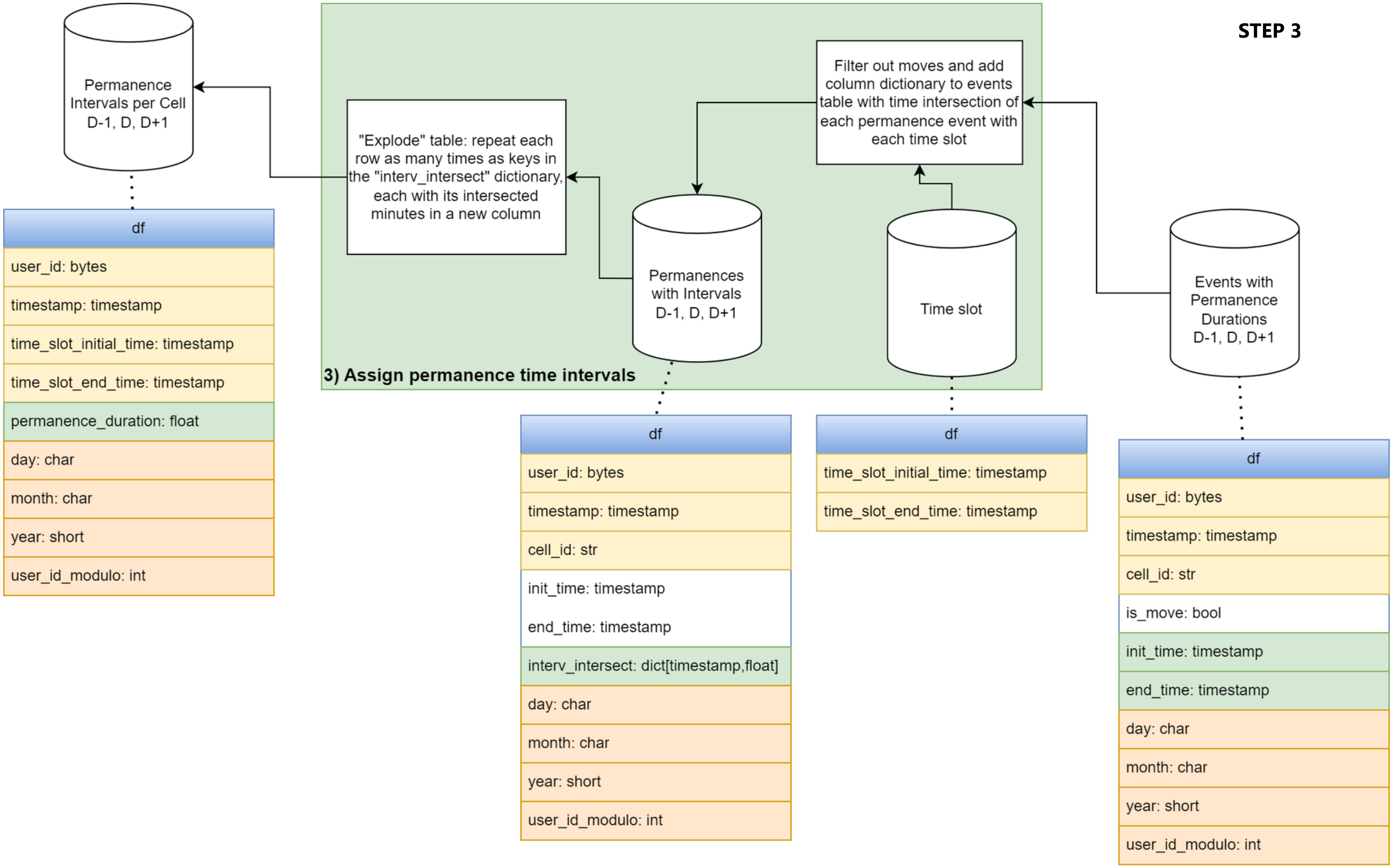
determine permanence duration interval according to time differences and equal/different cell

2) Determine permanence durations

Events with Permanence Durations
D-1, D, D+1

df
user_id: bytes
timestamp: timestamp
cell_id: str
is_move: bool
init_time: timestamp
end_time: timestamp
day: char
month: char
year: short
user_id_modulo: int

STEP 3



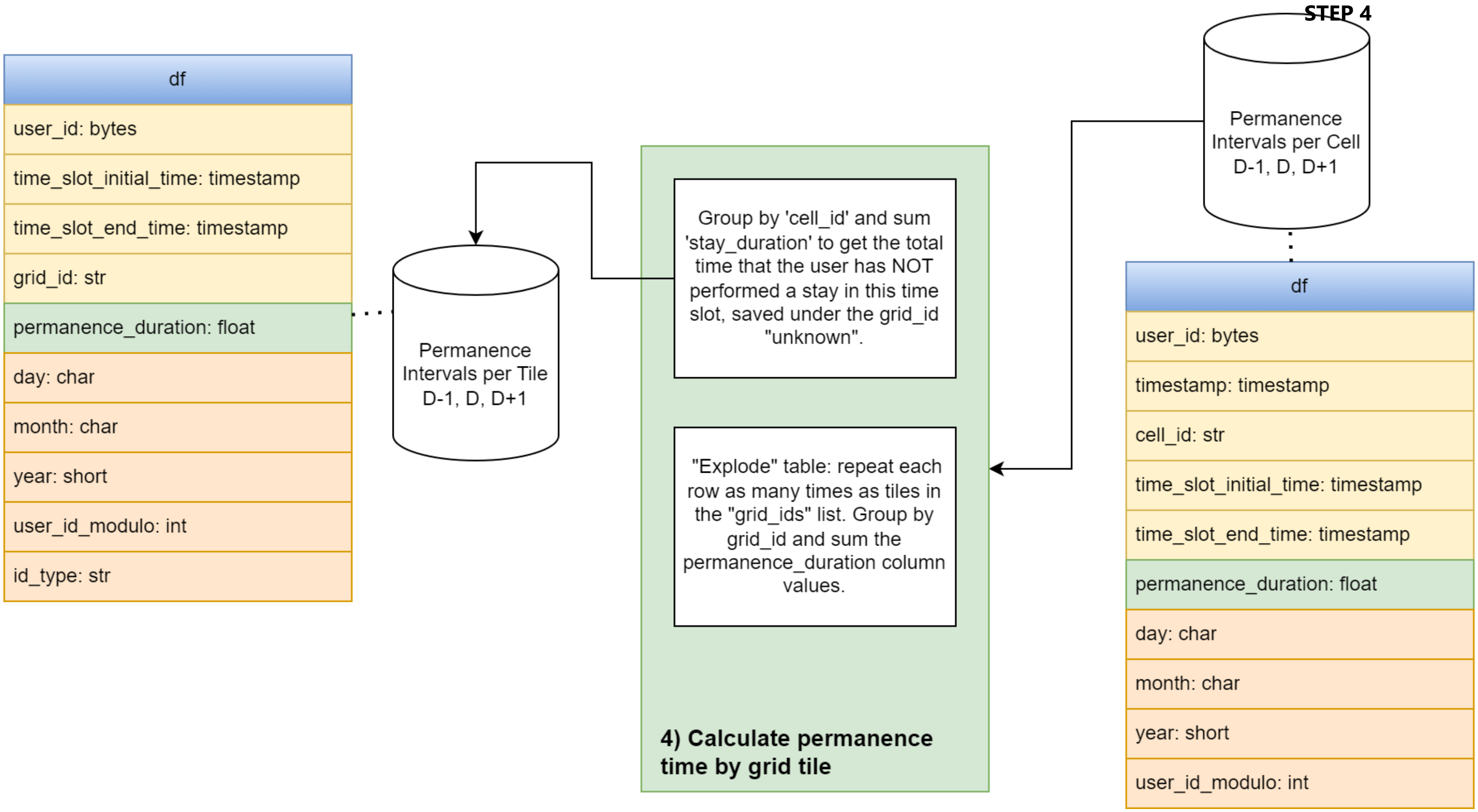
Permanence Intervals per Cell D-1, D, D+1
"Explode" table: repeat each row as many times as keys in the "interv_intersect" dictionary, each with its intersected minutes in a new column
Filter out moves and add column dictionary to events table with time intersection of each permanence event with each time slot
Permanences with Intervals D-1, D, D+1
Time slot
Events with Permanence Durations D-1, D, D+1
3) Assign permanence time intervals

df
user_id: bytes
timestamp: timestamp
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
permanence_duration: float
day: char
month: char
year: short
user_id_modulo: int

df
user_id: bytes
timestamp: timestamp
cell_id: str
init_time: timestamp
end_time: timestamp
interv_intersect: dict[timestamp,float]
day: char
month: char
year: short
user_id_modulo: int

df
time_slot_initial_time: timestamp
time_slot_end_time: timestamp

df
user_id: bytes
timestamp: timestamp
cell_id: str
is_move: bool
init_time: timestamp
end_time: timestamp
day: char
month: char
year: short
user_id_modulo: int



STEP 4

Permanence Intervals per Cell D-1, D, D+1

df
user_id: bytes
timestamp: timestamp
cell_id: str
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
permanence_duration: float
day: char
month: char
year: short
user_id_modulo: int

"Explode" table: repeat each row as many times as tiles in the "grid_ids" list. Group by grid_id and sum the permanence_duration column values.

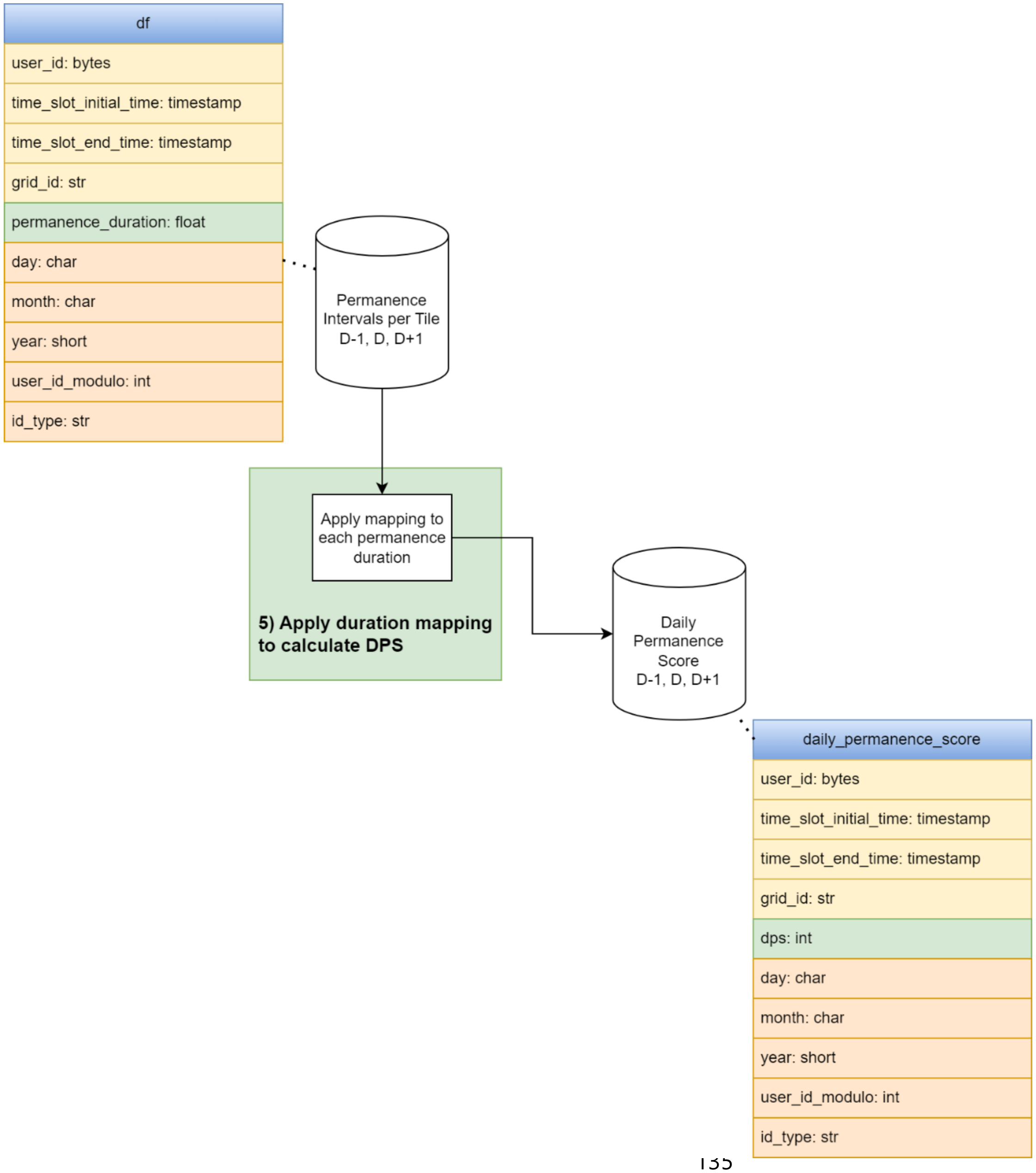
Group by 'cell_id' and sum 'stay_duration' to get the total time that the user has NOT performed a stay in this time slot, saved under the grid_id "unknown".

4) Calculate permanence time by grid tile

Permanence Intervals per Tile D-1, D, D+1

df
user_id: bytes
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
grid_id: str
permanence_duration: float
day: char
month: char
year: short
user_id_modulo: int
id_type: str

STEP 5



df
user_id: bytes
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
grid_id: str
permanence_duration: float
day: char
month: char
year: short
user_id_modulo: int
id_type: str

Permanence Intervals per Tile D-1, D, D+1

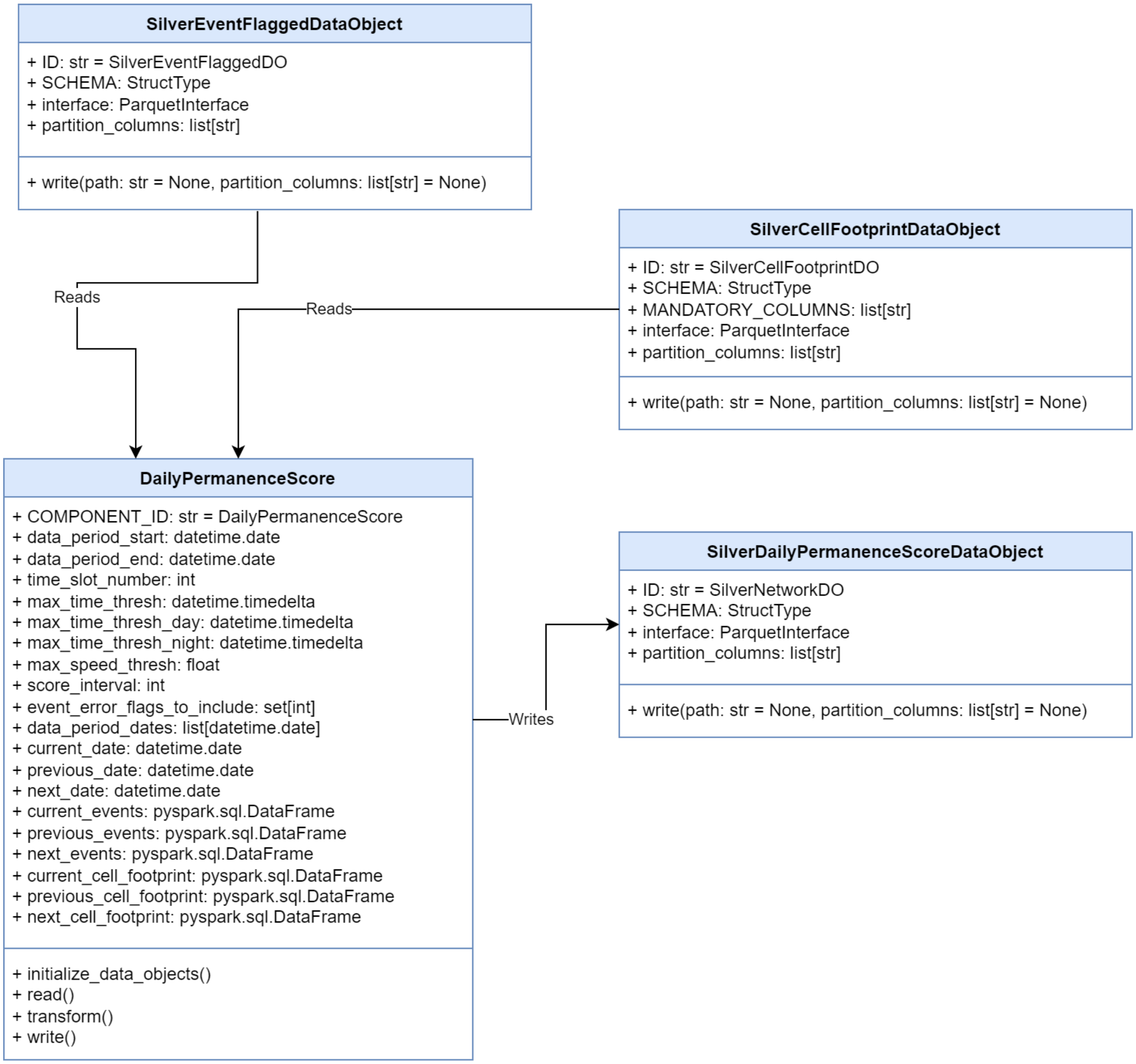
Apply mapping to each permanence duration

5) Apply duration mapping to calculate DPS

Daily Permanence Score D-1, D, D+1

daily_permanence_score
user_id: bytes
time_slot_initial_time: timestamp
time_slot_end_time: timestamp
grid_id: str
dps: int
day: char
month: char
year: short
user_id_modulo: int
id_type: str

- **Class diagram:**



SilverEventFlaggedDataObject

+ ID: str = SilverEventFlaggedDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]

+ write(path: str = None, partition_columns: list[str] = None)

SilverCellFootprintDataObject

+ ID: str = SilverCellFootprintDO
+ SCHEMA: StructType
+ MANDATORY_COLUMNS: list[str]
+ interface: ParquetInterface
+ partition_columns: list[str]

+ write(path: str = None, partition_columns: list[str] = None)

DailyPermanenceScore

+ COMPONENT_ID: str = DailyPermanenceScore
+ data_period_start: datetime.date
+ data_period_end: datetime.date
+ time_slot_number: int
+ max_time_thresh: datetime.timedelta
+ max_time_thresh_day: datetime.timedelta
+ max_time_thresh_night: datetime.timedelta
+ max_speed_thresh: float
+ score_interval: int
+ event_error_flags_to_include: set[int]
+ data_period_dates: list[datetime.date]
+ current_date: datetime.date
+ previous_date: datetime.date
+ next_date: datetime.date
+ current_events: pyspark.sql.DataFrame
+ previous_events: pyspark.sql.DataFrame
+ next_events: pyspark.sql.DataFrame
+ current_cell_footprint: pyspark.sql.DataFrame
+ previous_cell_footprint: pyspark.sql.DataFrame
+ next_cell_footprint: pyspark.sql.DataFrame

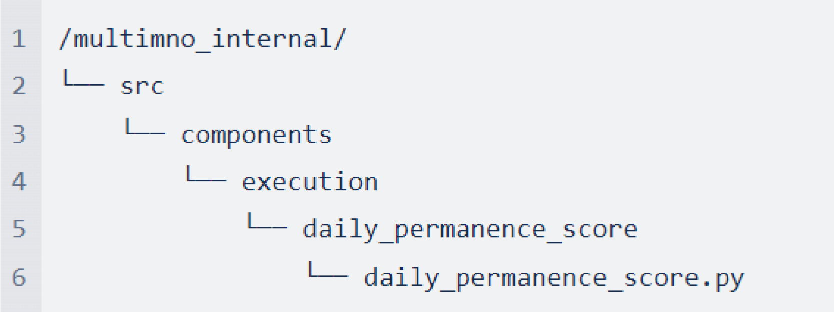
+ initialize_data_objects()
+ read()
+ transform()
+ write()

SilverDailyPermanenceScoreDataObject

+ ID: str = SilverNetworkDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]

+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:



```
/multimno_internal/
└── src
    └── components
        └── execution
            └── daily_permanence_score
                └── daily_permanence_score.py
```

daily_permanence_score.py contains one class named DailyPermanenceScore which is a subclass of Component. The DailyPermanenceScore class overrides some of the methods of Component:

The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.

The transform method performs all necessary filtering and transformations pertaining to the daily permanence score calculation.

5.2.14 INSPIREReferenceGridGeneration

5.2.14.1 MODULE DESCRIPTION

- **Module Name:** InspireGridGeneration
- **Objectives:** Create Reference Grid in INSPIRE format.
- **Functionality:** Generates 100 by 100 meters rectangular grid following INSPIRE specification for the given extent or given country polygon. For a country polygon, a buffer distance can be defined to extend grid beyond the country polygon borders.
  Functionality specification:
  - 3.2.13 INSPIRE Grid Generation
- **Data Inputs and Outputs:**
  - Inputs:
    - I.29 Countries
  - Outputs:
    - I.11 Reference Grid

5.2.14.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**

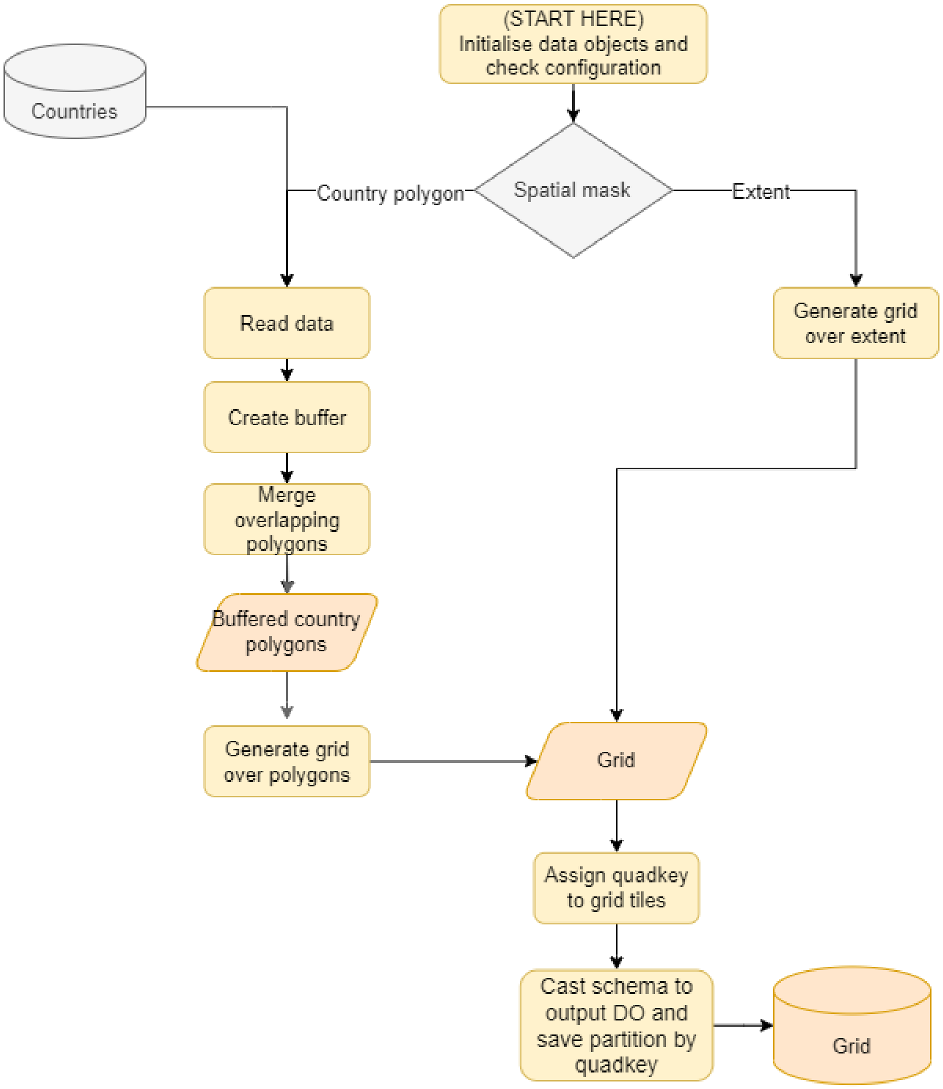
Initialisation:

1. Read configurations parameters: type of grid mask, extent, reference country, buffer around country borders, quadkey level for spatial partitioning.
2. Clear the destination directory if configured.
3. Load input data objects for grid, transportation, and land use data.
4. Initialize the output data object for the enriched grid.

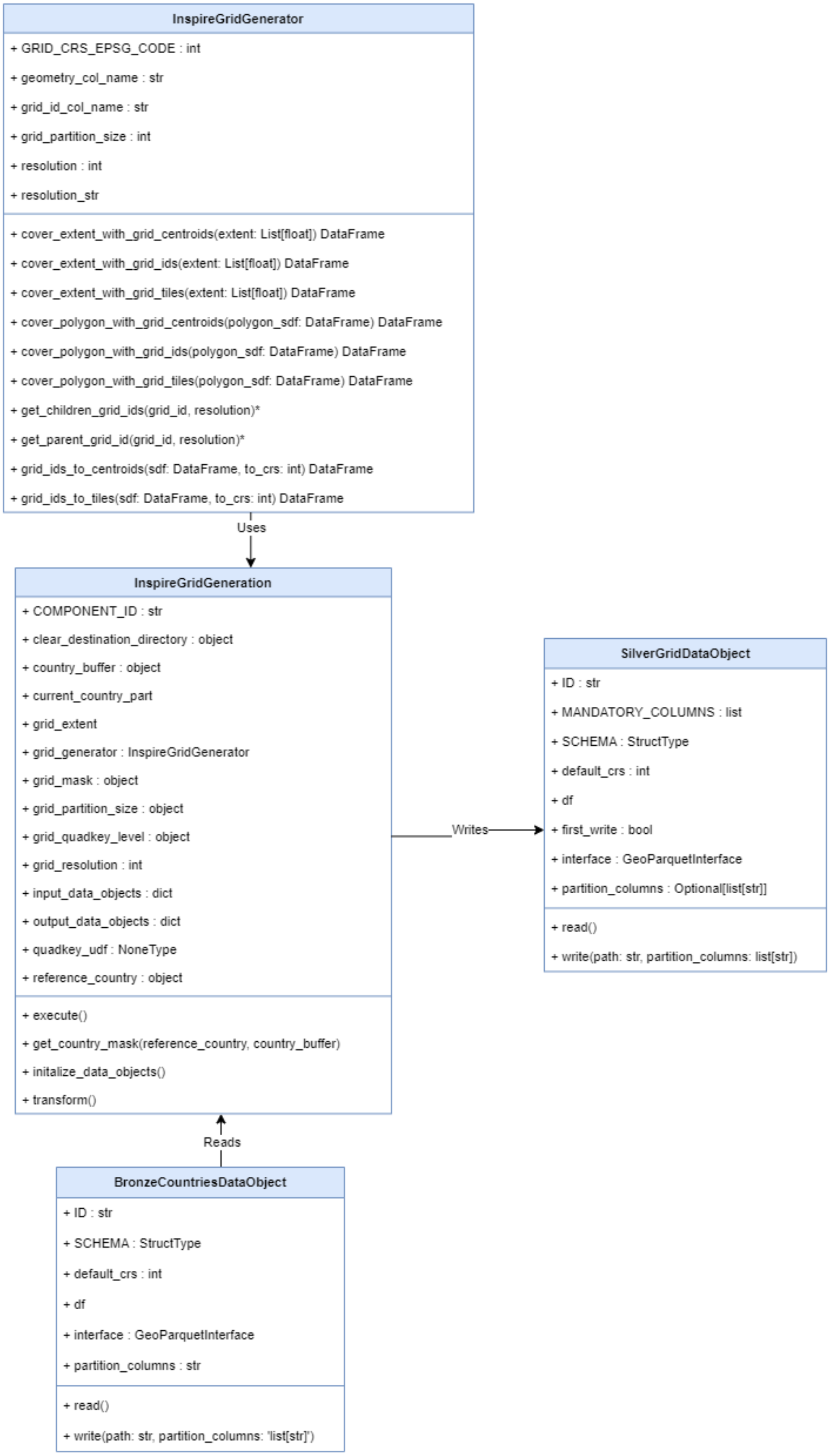
Processing:

1. Generate INSPIRE grid centroids.
    1. If grid mask parameter is 'extent', generate grid for given extent.
    2. If grid mask parameter 'polygon':
        1. Filter reference country polygon from countries dataset using given iso2 code.
        2. Create a buffer of a given distance around country polygons.
        3. Country may consist of multiple polygons, so it is necessary to merge overlapping resulted polygons together.
        4. Generate grid for each polygon.
2. Assign quadkey of a given level for each grid tile based on tile's centroid latitude and longitude.
3. Convert dataset schema to match the output data object schema and save to storage partitioned by quadkey.

- **Data flow diagram:**

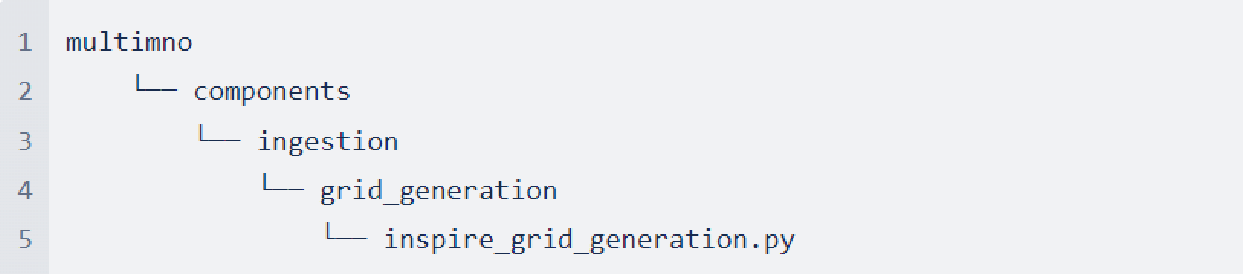

(START HERE)
Initialise data objects and check configuration
Countries
Spatial mask
Country polygon
Extent
Read data
Create buffer
Merge overlapping polygons
Buffered country polygons
Generate grid over polygons
Generate grid over extent
Grid
Assign quadkey to grid tiles
Cast schema to output DO and save partition by quadkey
Grid

- **Class diagram:**


InspireGridGenerator
+ GRID_CRS_EPSG_CODE : int
+ geometry_col_name : str
+ grid_id_col_name : str
+ grid_partition_size : int
+ resolution : int
+ resolution_str
+ cover_extent_with_grid_centroids(extent: List[float]) DataFrame
+ cover_extent_with_grid_ids(extent: List[float]) DataFrame
+ cover_extent_with_grid_tiles(extent: List[float]) DataFrame
+ cover_polygon_with_grid_centroids(polygon_sdf: DataFrame) DataFrame
+ cover_polygon_with_grid_ids(polygon_sdf: DataFrame) DataFrame
+ cover_polygon_with_grid_tiles(polygon_sdf: DataFrame) DataFrame
+ get_children_grid_ids(grid_id, resolution)*
+ get_parent_grid_id(grid_id, resolution)*
+ grid_ids_to_centroids(sdf: DataFrame, to_crs: int) DataFrame
+ grid_ids_to_tiles(sdf: DataFrame, to_crs: int) DataFrame
Uses
InspireGridGeneration
+ COMPONENT_ID : str
+ clear_destination_directory : object
+ country_buffer : object
+ current_country_part
+ grid_extent
+ grid_generator : InspireGridGenerator
+ grid_mask : object
+ grid_partition_size : object
+ grid_quadkey_level : object
+ grid_resolution : int
+ input_data_objects : dict
+ output_data_objects : dict
+ quadkey_udf : NoneType
+ reference_country : object
+ execute()
+ get_country_mask(reference_country, country_buffer)
+ initalize_data_objects()
+ transform()
Writes
SilverGridDataObject
+ ID : str
+ MANDATORY_COLUMNS : list
+ SCHEMA : StructType
+ default_crs : int
+ df
+ first_write : bool
+ interface : GeoParquetInterface
+ partition_columns : Optional[list[str]]
+ read()
+ write(path: str, partition_columns: list[str])
Reads
BronzeCountriesDataObject
+ ID : str
+ SCHEMA : StructType
+ default_crs : int
+ df
+ interface : GeoParquetInterface
+ partition_columns : str
+ read()
+ write(path: str, partition_columns: 'list[str]')

- **Code Structure:**

The code structure follows the format set by the core package, and the general repository structure.
The location of the module script in the repository is as follows:



```
multimno
    └── components
        └── ingestion
            └── grid_generation
                └── inspire_grid_generation.py
```

inspire_grid_generation.py contains one class named InspireGridGeneration which is a subclass of Component.
The InspireGridGeneration class is rely on InspireGridGenerator utility class to perform actual grid generation.
The InspireGridGeneration class overrides transform method of base Component class.
transform method instantiates InspireGridGenerator class and uses its method to generate INPSIRE grid centroids for the given spatial extent or country polygon.

5.2.15 SYNTHETICDIARIES

5.2.15.1 MODULE DESCRIPTION

- **Module Name:** SyntheticDiaries
- **Objectives:** the objective of this module is to generate a given number of synthetic user activity-trip diaries.
- **Functionality:** the module includes the following functionalities:
    - Synthetically generating users with random but compatible home and work locations.
    - Synthetically generating compatible sequences of stays for each user (e.g. home-work-other-home).
    - Synthetically assigning a location (exact coordinates) to each of the activities of the user.
    - Synthetically assigning a start time and an end time to each of the activities of the user.
- **Data Inputs and Outputs:**
    - Input:
        - No input datasets are used by this method
    - Output:
        - I.30 Synthetic Diaries

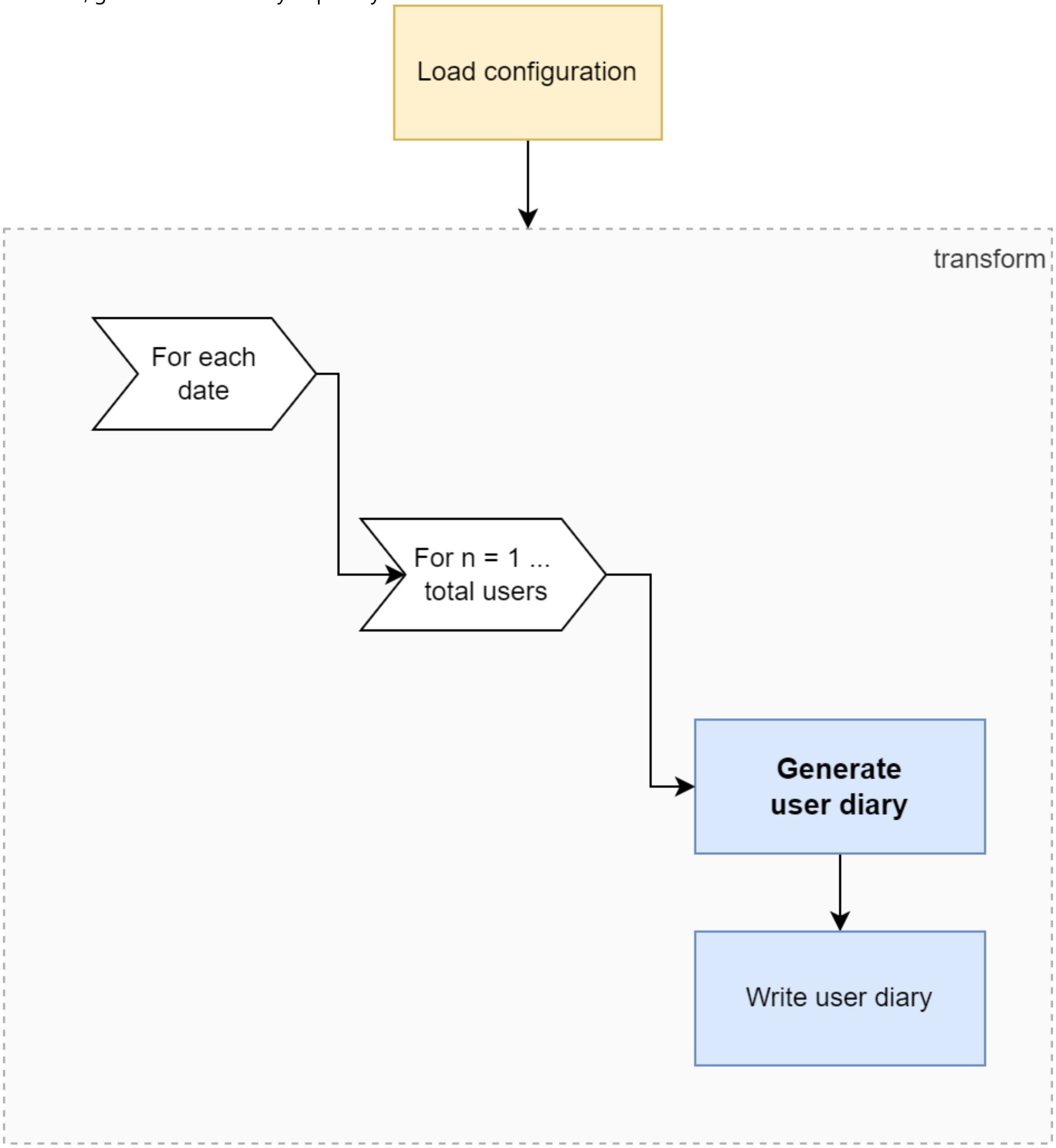
5.2.15.2 DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
    - Read from the configuration file the number of users for which to generate diaries.
    - Read from the configuration file the date format for the output diaries.
    - Read from the configuration file the initial date for the diary generation.
    - Read from the configuration file the number of dates for the diary generation.
    - Read from the configuration file the maximum longitude and latitude for activity generation (bounding box top right corner limit).
    - Read from the configuration file the minimum longitude and latitude for activity generation (bounding box bottom left corner limit).

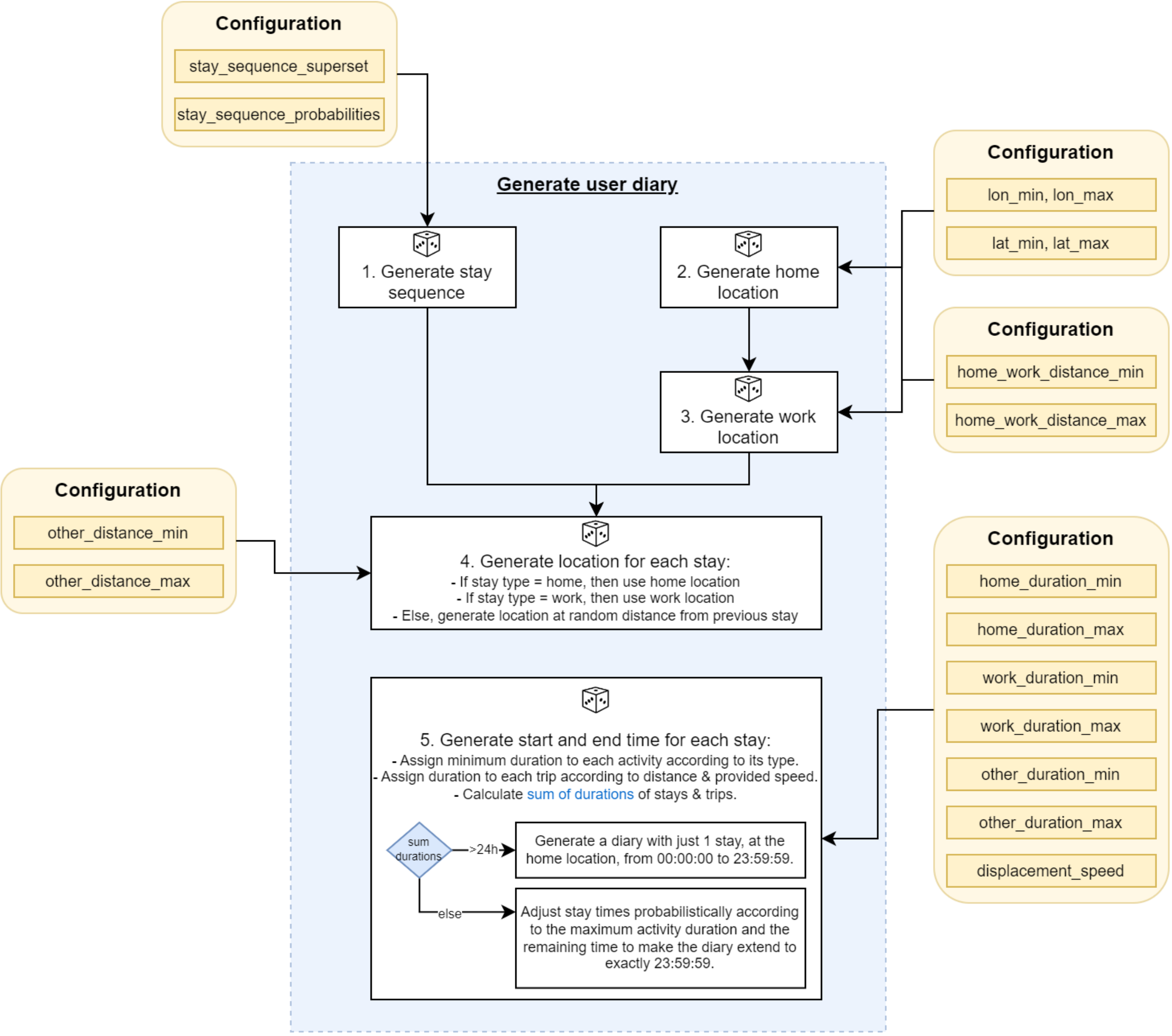
- Read from the configuration file the minimum and maximum distance between home and work for synthetic generation.
- Read from the configuration file the minimum and maximum distance for the assignment of the location of other activities with respect to the location of the previous activity.
- Read from the configuration file the minimum and maximum home activity duration.
- Read from the configuration file the minimum and maximum work activity duration.
- Read from the configuration file the minimum and maximum other activity duration.
- Read from the configuration file the displacement speed which will be considered for assignment of the start activity time of the next activity.
- Read from the configuration file the stay sequence superset in which all synthetically generated diaries will be based.
- Read from the configuration file the sequence of probabilities of the stays in the stay sequence superset of being generated in each synthetically generated diary.
- For each date, starting in the initial date and ending in initial date + number of dates:
    - Create one agent, from 0 to the provided number of users, and for each agent:
        1. Generate a stay sequence for an agent probabilistically based on the provided stay sequence and weights (e.g. home-work-other-other-home).
        2. Generate home location coordinates for the agent based on the bounding limits.
        3. Generate work location coordinates for the agent based on the bounding limits and minimum and maximum distance to home.
        4. Generate activity locations for each of the activities in the generated stay sequence for this agent:
            1. For 'home' activities, assign home location of the agent.
            2. For 'work' activities, assign work location of the agent.
            3. For 'other', reach previous activity, and assign a random location that is at a distance to the previous activity location that is between the provided thresholds.
        5. Generate activity times according to generated stay sequence for this agent:
            1. Firstly, assign to each of these activities the minimum duration considered for that activity type. Trip times are based on Pythagorean distance and a specified average speed.
                1. If the sum of all minimum duration of the activities and the duration of the trips is higher than the 24h of the day, then assign just one "home" activity to the agent from 00:00:00 to 23:59:59.
                2. Else, there will be a remaining time. E.g., the diary of an agent, after adding up all trip durations and minimum activity durations may end at 20:34:57. There is a remaining time to complete the full diary (23:59:59 - 20:34:57). Adjust activity times probabilistically according to the maximum activity duration and this remaining time, making the diary end at exactly 23:59:59.
        6. Write diaries to output file.

- **Data flow diagram:**
  Initialising the SyntheticDiaries component launches a process that loads all necessary parameters from the configuration path.

Then, the transform method is applied, triggering the processing of the rest of the method, which, for each date and user, generates an activity-trip diary:

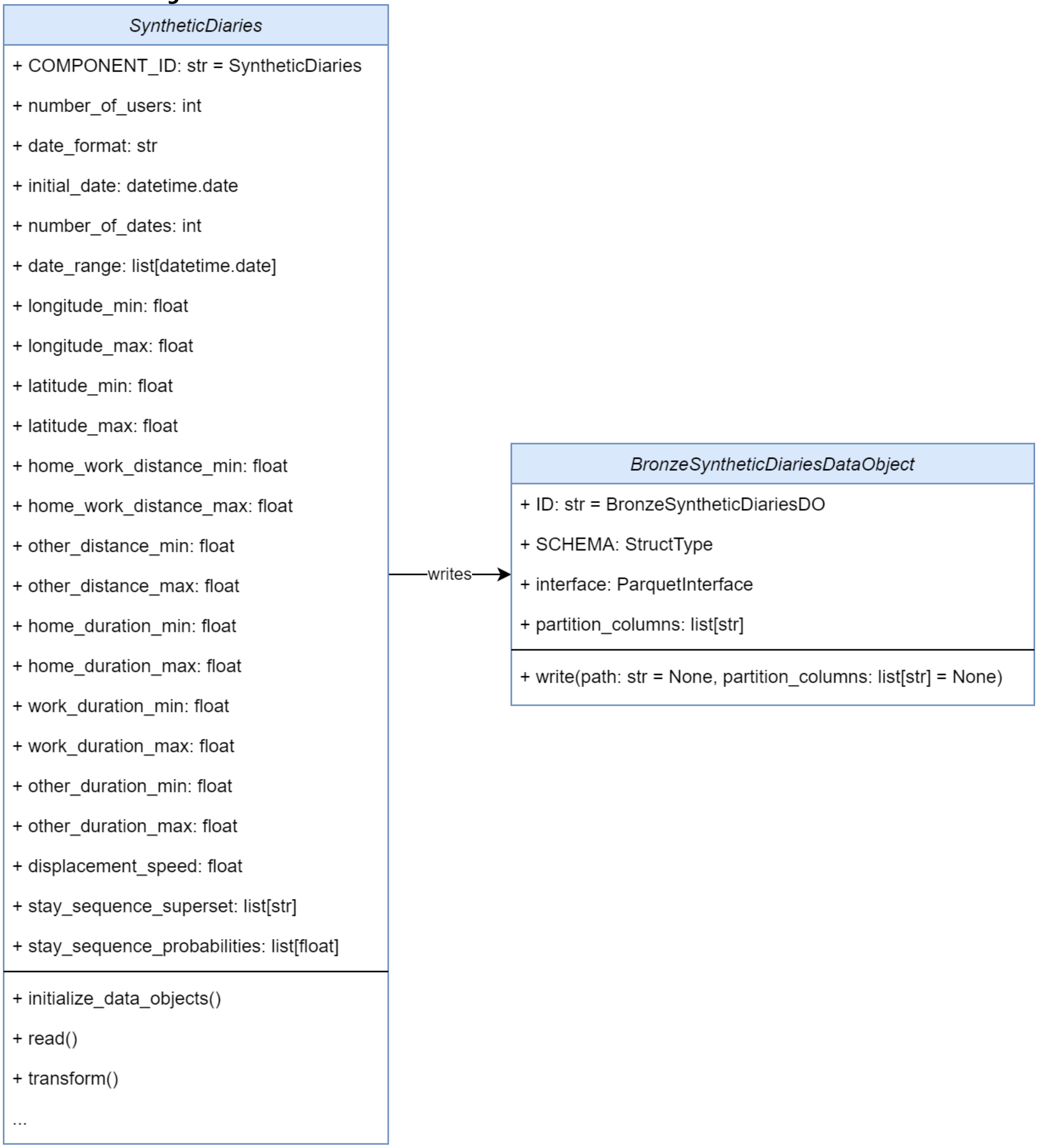


The generation of each user's diary is described in the graph below:



Configuration
stay_sequence_superset
stay_sequence_probabilities
Generate user diary
1. Generate stay sequence
2. Generate home location
3. Generate work location
Configuration
lon_min, lon_max
lat_min, lat_max
Configuration
home_work_distance_min
home_work_distance_max
Configuration
other_distance_min
other_distance_max
4. Generate location for each stay:
- If stay type = home, then use home location
- If stay type = work, then use work location
- Else, generate location at random distance from previous stay
5. Generate start and end time for each stay:
- Assign minimum duration to each activity according to its type.
- Assign duration to each trip according to distance & provided speed.
- Calculate sum of durations of stays & trips.
sum durations
>24h
Generate a diary with just 1 stay, at the home location, from 00:00:00 to 23:59:59.
else
Adjust stay times probabilistically according to the maximum activity duration and the remaining time to make the diary extend to exactly 23:59:59.
Configuration
home_duration_min
home_duration_max
work_duration_min
work_duration_max
other_duration_min
other_duration_max
displacement_speed

- **Class diagram:**


SyntheticDiaries
+ COMPONENT_ID: str = SyntheticDiaries
+ number_of_users: int
+ date_format: str
+ initial_date: datetime.date
+ number_of_dates: int
+ date_range: list[datetime.date]
+ longitude_min: float
+ longitude_max: float
+ latitude_min: float
+ latitude_max: float
+ home_work_distance_min: float
+ home_work_distance_max: float
+ other_distance_min: float
+ other_distance_max: float
+ home_duration_min: float
+ home_duration_max: float
+ work_duration_min: float
+ work_duration_max: float
+ other_duration_min: float
+ other_duration_max: float
+ displacement_speed: float
+ stay_sequence_superset: list[str]
+ stay_sequence_probabilities: list[float]
+ initialize_data_objects()
+ read()
+ transform()
...
writes
BronzeSyntheticDiariesDataObject
+ ID: str = BronzeSyntheticDiariesDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

    -
      ```
      /multimno/
      └── src
          └── components
              └── ingestion
                  └── synthetic
                      └── synthetic_diaries.py
      ```
    - synthetic_diaries.py contains one class named SyntheticDiaries which is a subclass of Component. It overrides the following methods:
        - The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
        - The transform method handles all the logic behind the component.
    - The SyntheticDiaries component also has the following methods:
        - initalize_data_objects loads the output data object schema.
        - haversine calculates haversine distance between 2 points in lat-lon.
        - random_seed_number_generator generates random seed based on several arguments.
        - calculate_trip_time calculates trip time given an origin location and a destination location, according to the specified trip speed.
        - calculate_trip_final_time calculates end time of a trip given an origin time, an origin location, a destination location and a speed.
        - generate_stay_location generates a random activity location within the bounding box limits based on the activity type and previous activity locations.
        - create_agent_activities_min_duration generates activities of the minimum duration following the specified agent activity sequence for this agent and date.
        - adjust_activity_times modifies the "date_activities" list, changing the initial and final timestamps of both stays and moves probablilistically in order to generate stay durations different from the minimum and adjust the durations of the activities to the 24h of the day.
        - add_agent_date_activities for a specific date and user, generate a sequence of activities probabilistically according to the specified activity superset and the activity probabilities. Firstly, assign to each of these activities the minimum duration considered for that activity type. Trip times are based on Pythagorean distance and a specified average speed. If the sum of all minimum duration of the activities and the duration of the trips is higher than the 24h of the day, then assign just one "home" activity to the agent from 00:00:00 to 23:59:59. Else, there will be a remaining time. E.g., the diary of an agent, after adding up all trip durations and minimum activity durations may end at 20:34:57. There is a remaining time to complete the full diary (23:59:59 - 20:34:57). Adjust activity times probabilistically according to the maximum activity duration and this remaining time, making the diary end at exactly 23:59:59.
        - add_date_activities generates activity (stays and moves) rows for a specific date according to parameters.
        - generate_activities generates activity and trip rows according to parameters.
        - generate_lonlat_at_distance given a point (lon, lat) and a distance, in meters, calculates a new random point that is exactly at the specified distance of the provided lon, lat.
        - generate_home_location generates a random home location based on bounding box limits.

        - generate_work_location generates random work location based on home location and maximum distance to home. If the work location falls outside of bounding box limits, try again.
        - generate_other_location generates other activity location based on previous location and maximum distance to previous location. If there is no previous location (this is the first activity of the day), then the home location is considered as previous location. If the location falls outside of bounding box limits, try again.
        - generate_stay_duration generates stay duration probabilistically based on activity type and remaining time.
        - generate_min_stay_duration generates minimum stay duration based on stay type specifications.
        - remove_consecutive_stay_types generates new list replacing consecutive stays of the same type by a unique stay as long as the stay type is contained in the "stay_types_to_group" list.
        - generate_stay_type_sequence generates the sequence of stay types for an agent for a specific date probabilistically based on the superset sequence and specified probabilities. Replace 'home'-'home' and 'work'-'work' sequences by just 'home

5.2.16 SYNTHETICNETWORK

5.2.16.1 MODULE DESCRIPTION

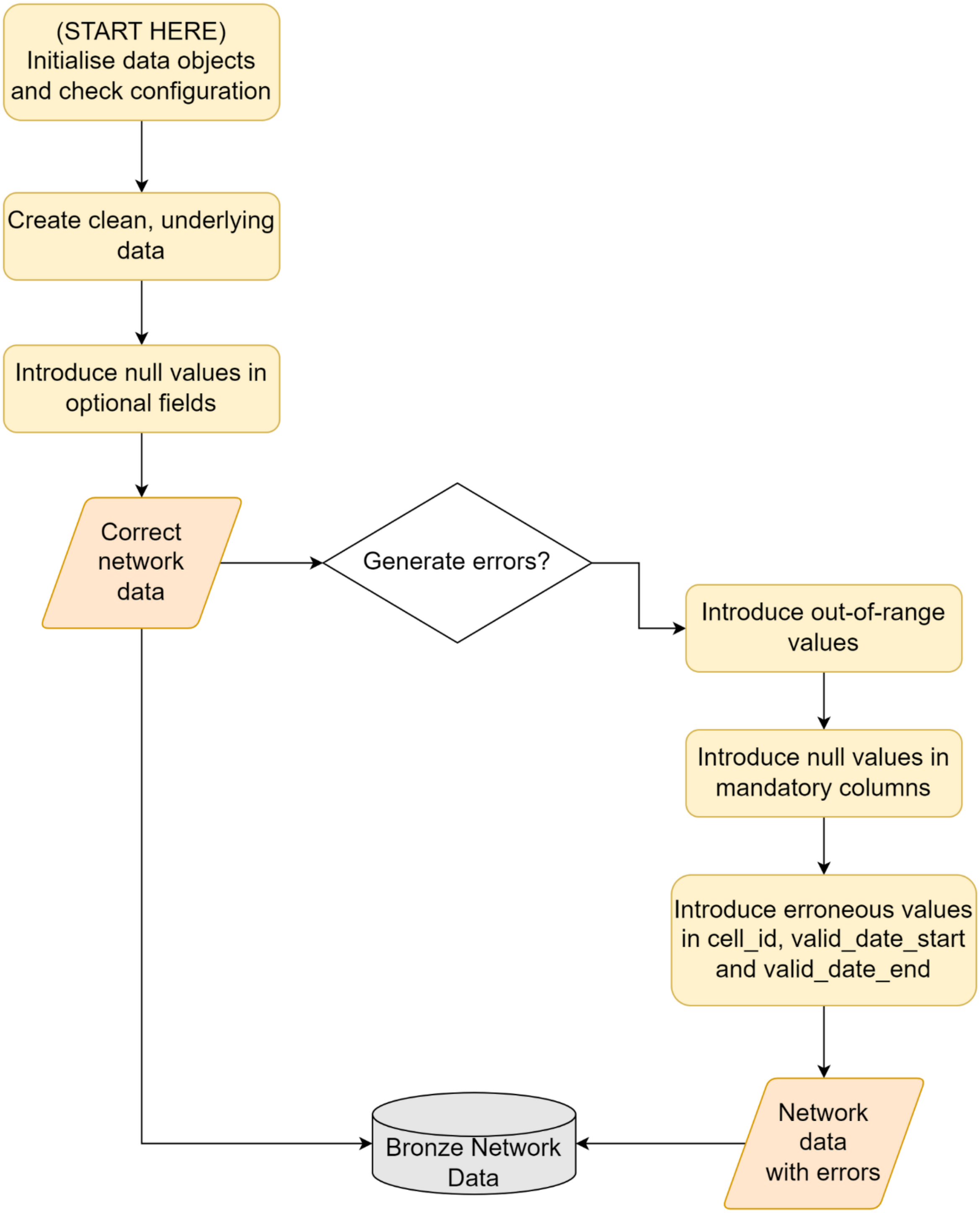
- **Module Name:** SyntheticNetwork.
- **Objectives:** The main goal of the service is to generate synthetic MNO network topology data to simulate real network data provided by the MNO, and allowing the testing and execution of the pipeline. The development of this service will be incremental, iteratively adding more features and characteristics of the real data as the different steps of the pipeline will require them.
- **Functionality:** The SRS documentation sums up the functionality of this service: 3.2.15 SyntheticNetwork.
- **Data Inputs and Outputs:**
    - There are no input objects.
    - The current output data object is I.7 Cell Locations with Physical Properties - Raw.

5.2.16.2 DEVELOPMENT DESIGN

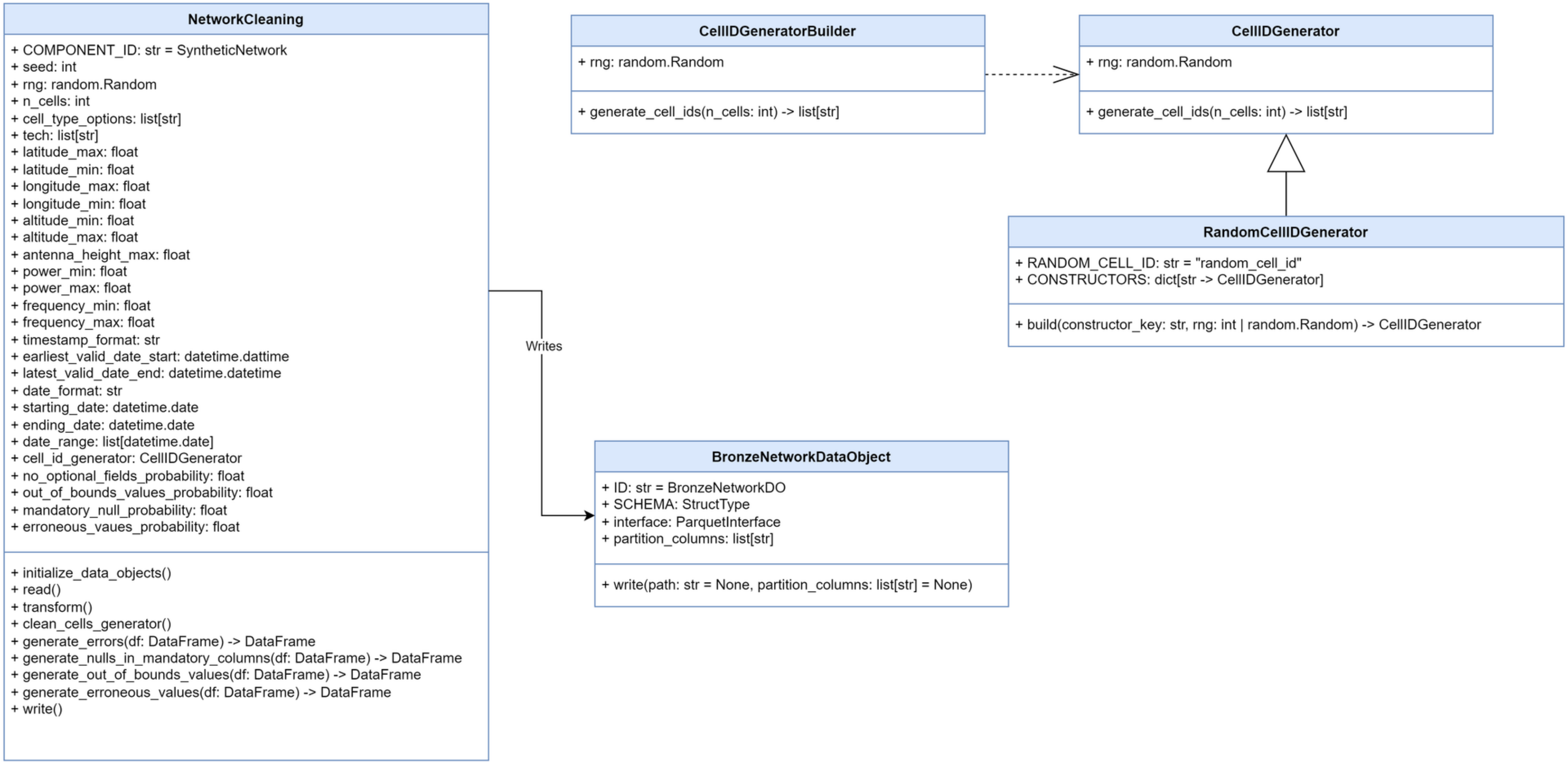
- **Key Algorithms/Processes:** an underlying set of all cells is initially generated. In the clean, no error version, each cell will have a constant value of each of its properties across all dates (e.g., the altitude will be the same for every day). It is supposed that cell data will be available at a daily rate, thus, a parquet partition will be created for every day between starting_date and ending_date.
    - Clean, underlying data generation:
        - The ID of the cell is generated as a 14- or 15-digit string (for now, not following CGI/eCGI standards).
        - The latitude and longitude of the cell are randomly generated with uniform probability in the rectangle defined by latitude_min, latitude_max, longitude_min and longitude_max.
        - The altitude is randomly sampled with uniform probability in the interval defined by altitude_min and altitude_max.
        - The antenna height is randomly sampled with uniform probability in the interval defined by 0and antenna_height_max.

  - The directionality is randomly sampled from the set *{0, 1}* with uniform probability.
  - The azimuth angle is equal to None if the directionality is equal to 0, or is randomly sampled with uniform probability in the interval *[0, 360]*.
  - The elevation angle is randomly sampled with uniform probability in the interval *[-90, 90]*.
  - The horizontal and vertical beam widths are each randomly sampled with uniform probability in the interval *[0, 360]*.
  - The power is randomly sampled with uniform probability in the interval defined by power_minand power_max.
  - The range is randomly sampled with uniform probability in the interval defined by range_min and range_max.
  - The power is randomly sampled with uniform probability in the interval defined by frequency_min and frequency_max.
  - The technology is randomly sampled with uniform probability from the four options 5G, LTE, UMTS, GSM.
  - The valid date start is set to earlist_valid_date_start.
  - The valid date end is set to latest_valid_date_end.
  - The cell type is randomly sampled with uniform probability from a set of options defined via configuration in cell_type_options. Example: macrocell, microcell, picocell, femtocell.
- Null values: with a probability specified via configuration, all optional fields of a row are set to null.
- If the user decides to generate synthetic data with errors, then the following steps are followed:
  - Out of bound values: with a probability specified via configuration, for each appropriate field a subset of records is selected and that field's values are changed by value outside the admitted range of values.
  - Nulls in mandatory columns: with a probability specified via configuration, for each mandatory field a subset of records is selected and that field's values are changed to null.
  - Erroneous values:
    - With a probability specified via configuration, a subset of records is selected and the cell_id value is changed by an erroneous one.
    - With a probability specified via configuration, a subset of records is selected: the valid_date_start and valid_date_end are swapped for one half, and for the other half the timestamps are changed for invalid ones.

- **Data flow diagram:**


(START HERE)
Initialise data objects and check configuration
Create clean, underlying data
Introduce null values in optional fields
Correct network data
Generate errors?
Introduce out-of-range values
Introduce null values in mandatory columns
Introduce erroneous values in cell_id, valid_date_start and valid_date_end
Network data with errors
Bronze Network Data

- **Class diagram:**


NetworkCleaning
+ COMPONENT_ID: str = SyntheticNetwork
+ seed: int
+ rng: random.Random
+ n_cells: int
+ cell_type_options: list[str]
+ tech: list[str]
+ latitude_max: float
+ latitude_min: float
+ longitude_max: float
+ longitude_min: float
+ altitude_min: float
+ altitude_max: float
+ antenna_height_max: float
+ power_min: float
+ power_max: float
+ frequency_min: float
+ frequency_max: float
+ timestamp_format: str
+ earliest_valid_date_start: datetime.dattime
+ latest_valid_date_end: datetime.datetime
+ date_format: str
+ starting_date: datetime.date
+ ending_date: datetime.date
+ date_range: list[datetime.date]
+ cell_id_generator: CellIDGenerator
+ no_optional_fields_probability: float
+ out_of_bounds_values_probability: float
+ mandatory_null_probability: float
+ erroneous_vaues_probability: float
+ initialize_data_objects()
+ read()
+ transform()
+ clean_cells_generator()
+ generate_errors(df: DataFrame) -> DataFrame
+ generate_nulls_in_mandatory_columns(df: DataFrame) -> DataFrame
+ generate_out_of_bounds_values(df: DataFrame) -> DataFrame
+ generate_erroneous_values(df: DataFrame) -> DataFrame
+ write()
Writes
CellIDGeneratorBuilder
+ rng: random.Random
+ generate_cell_ids(n_cells: int) -> list[str]
CellIDGenerator
+ rng: random.Random
+ generate_cell_ids(n_cells: int) -> list[str]
RandomCellIDGenerator
+ RANDOM_CELL_ID: str = "random_cell_id"
+ CONSTRUCTORS: dict[str -> CellIDGenerator]
+ build(constructor_key: str, rng: int | random.Random) -> CellIDGenerator
BronzeNetworkDataObject
+ ID: str = BronzeNetworkDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
/multimno/
└── components
    └── ingestion
        └── synthetic
            └── synthetic_network.py
```

  - synthetic_network.py contains one class named SyntheticNetwork which is a subclass of Component. The SyntheticNetwork class overrides some of the methods of Component:
    - The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - transform performs handles the main logic of the execution.
  - The SyntheticNetwork class also has the following methods:
    - clean_cells_generator creates the clean synthetic network topology data according to the different parameters specified via configuration.
    - generate_errors handles the error generation logic whenever any of the probabilities of generating null values in mandatory columns, creating out-of-bound values or other erroneous values is greater than zero.
    - generate_nulls_in_mandatory_columns handles the changing of valid values in mandatory columns by null values.
    - generate_out_of_bounds_values handles the logic of creating values outside accepted ranges for each applicable field.
    - generate_erroneous_values is handles the creation of invalid cell_id values, swapping valid_date_start and valid_date_end values, and creating invalid timestamps for these two fields as well.

5.2.17 SYNTHETICEVENTS

5.2.17.1 MODULE DESCRIPTION

- **Module Name:** SyntheticEvents
- **Objectives:** the objective of this module is to generate synthetic data on the event level.
- **Functionality:** the module includes the following functionalities:
  - Generating event data that corresponds to stay and move information from synthetic diaries data object and and cell_ids and their locations from the synthetic network data object.
  - Generating event data given with a given set of stay and move frequency parameters and distance measure parameters.
  - Generating location errors and records with nonexistent cell_ids (cell_ids that are not present in synthetic network).
- **Data Inputs and Outputs:**
  - Input:
    - I.30 Synthetic Diaries
    - I.7 Cell Locations with Physical Properties - Raw
  - Output:
    - I.1 MNO Event Data – Raw

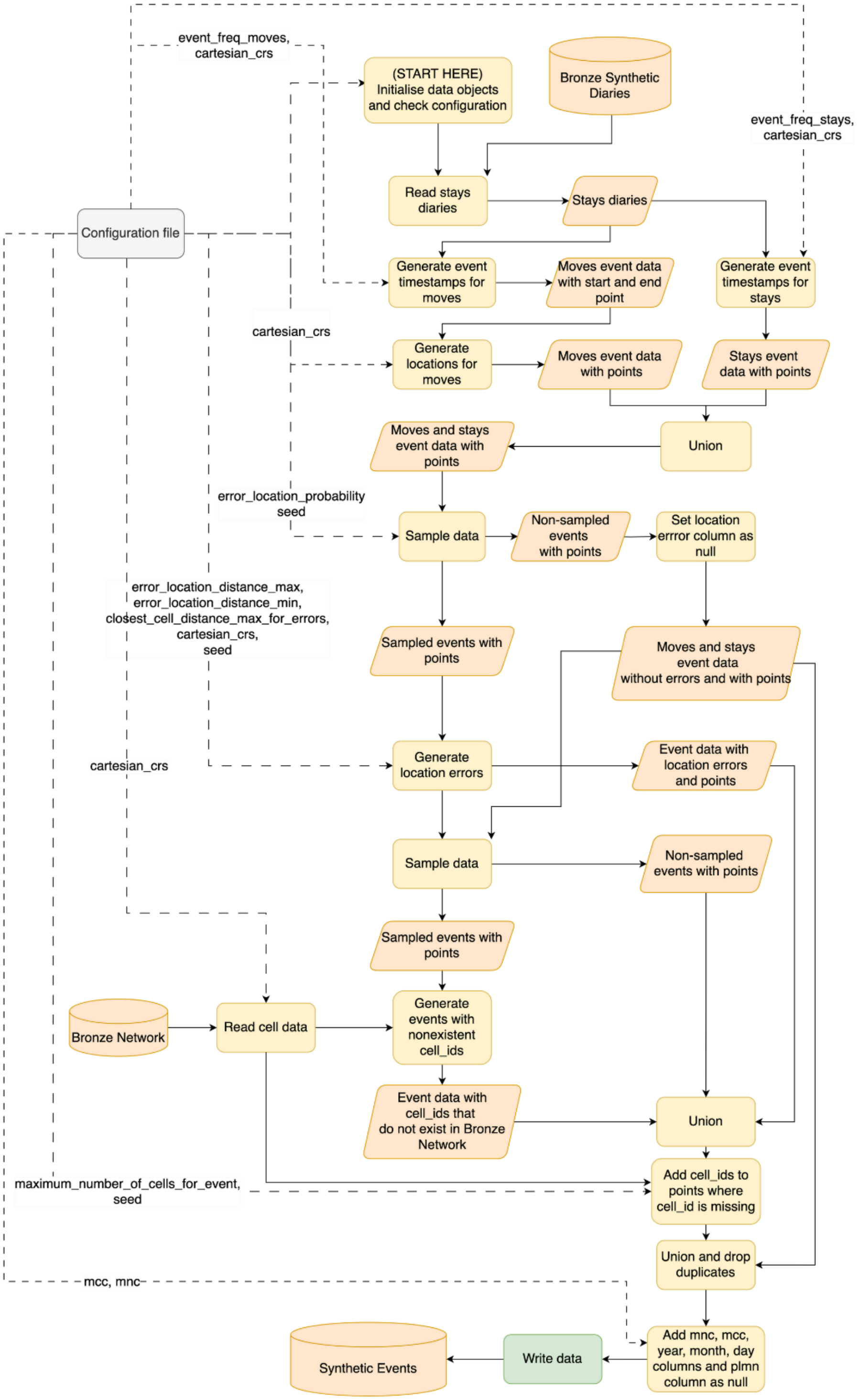
5.2.17.2DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
    - Create the data objects: diaries, network and event data.
    - Read from the configuration file the frequency of events to be generated for all stays.
    - Read from the configuration file the frequency of events to be generated for all moves.
    - Read from the configuration file the maximum distance for a cell to be allowed to be linked to generated point.
    - Read from the configuration file the maximum distance for a cell to be considered as the closest cell when generating events that have erroneous locations.
    - Read from the configuration file the ratio of generated stays and moves to sample, for the generating events that have nonexistent cell ids.
    - Read from the configuration file, the maximum number of cells to consider when generating an event.
    - Generate event timestamps for moves. From synthetic diaries, read in stays, and proceed as follows:
        1. Define window according to initial timestamp
        2. Define geometry column based on the longitude and latitude of next stay
        3. Calculate the time difference between the initial timestamp of next stay and the final timestamp of the current stay
        4. Calculate the total amount of events to be generated for a move event as time_difference_in_seconds / event_freq_moves.
        5. Generate as many random random values in the range of 0 and 1 as was the result of (4) on seperate rows - i.e., explode the dataframe so each random value is assigned to a separate row for a given combination of the timestamp column values
        6. Calculate the offset in seconds as random_float*time_difference_in_seconds
        7. Add the offset in seconds to the final timestamp, which results in random event timestamps between the two stays, and label these rows with the activity type “move”
    - Generate event timestamps for stays. Follow the same logic as for moves generation, but instead use the final and initial timestamp columns for the same row to calculate time difference. Then follow the same sampling idea and generate (or explode to) as many rows as time_difference_in_seconds / event_freq_moves results in.
    - Generate locations for moves. Calculate a line between the geometry of column of current stay, and the next stay. Interpolate on the point, random float values that have been calculated in step 5 of event timestamp generation for moves.
    - Sampling records for which to generate location errors. The parameter “error_location_probability” determines how many rows are sampled, according to the seed value given in the configuration.
    - Generating location errors for sampled records. Location errors are generated using the following steps:
        1. Generated longitude and latitude values from the sample are projected from EPSG:4326 to a configured coordinate reference system.
        2. The location error column (loc_error) is calculated as
           (random_float*(error_location_distance_max - error_location_distance_min))
           + error_location_distance_min
           where random_float is generated from a uniform distribution, with paramers (0, 1), using the seed value from configuration. The result is a random value within the range of configured parameters, that is be used to offset x and y values (the result of projection in step I).

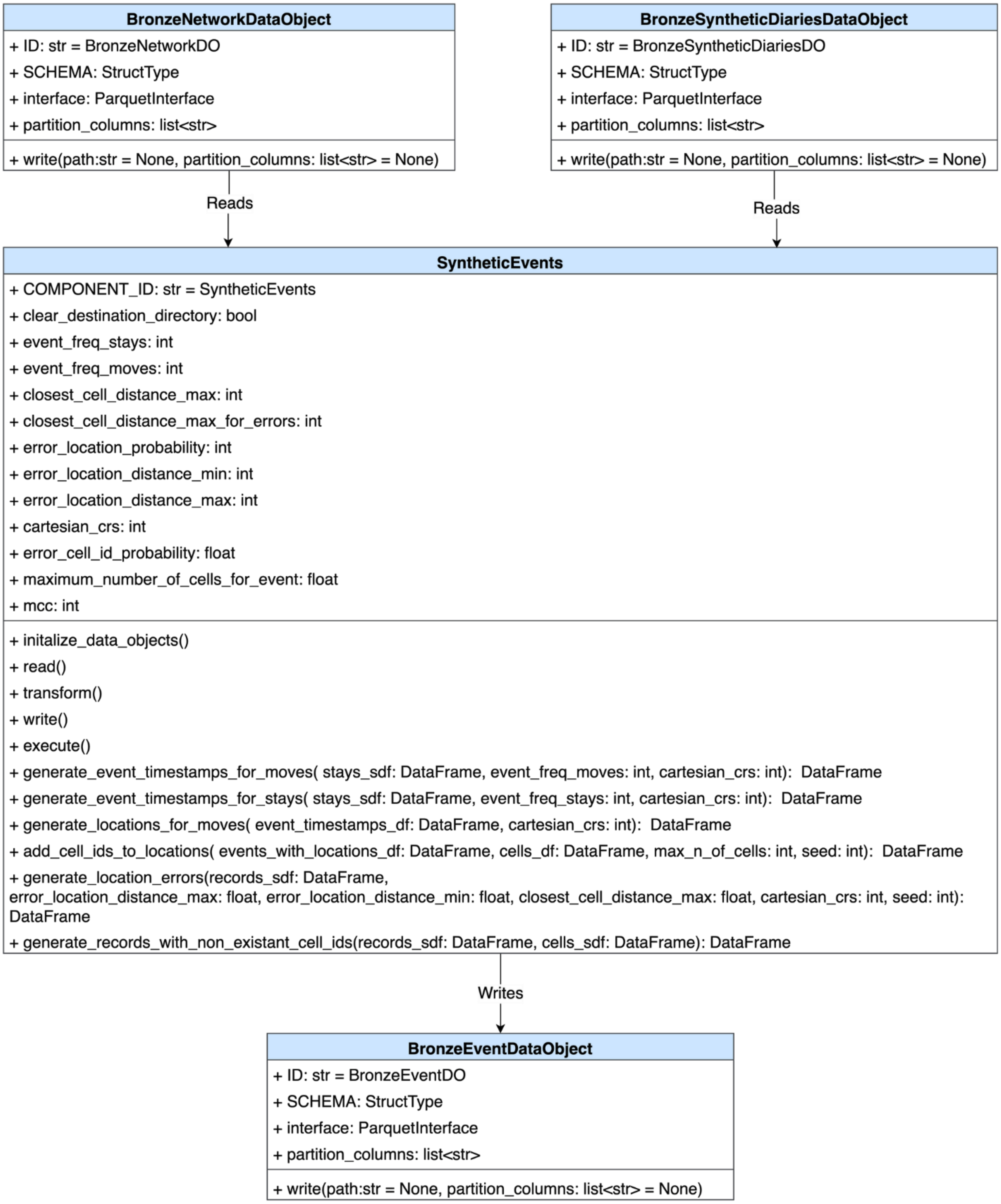
3. For each sampled x and y coordinate, randomly determine if the offset is applied by summing loc_error or subtracting it. The probability for either case is equal (0.5).
4. Existing generated points are replaced with the new points, that have been applied the offset value in the loc_error column with the sign generated in step III.

- Sampling records for which to generate nonexistent cell ids. Nonexistent here refers to cell ids that are not present in the input network data object. This is done by using the parameter error_cell_id_probability, which determines how many rows are sampled, according to the seed value given in the configuration.
- Generating records with nonexistent cell ids for sampled records. This is done by:
    1. Generate random values to match the cell_id field length requirements
    2. Using a left anti join, select from generated cell ids, only those that are not present in the existing network data.
    3. Join those selected in II to the sampled records, using a generated id. The generated id is equal to the row number in the result of II, whereas in the generated records, the id is achieved as row_number() % number_of_unique_cell_ids_in_network_data_object.
- Add cell ids to all previously generated latitude and longitude values. This is done by:
    1. Creating a buffer around each event location and finding cells that intersect with this buffer
    2. Calculating the distance from each event location to the cell and ranking the cells based on this distance.
    3. Keeping only the top 'max_n_of_cells' closest cells for each event
- Add the mcc column value for all users, as the value given in the configuration parameter "mcc"
- Add the mnc column value for all users, as the value given in the configuration parameter "mnc"
- Add the plmn column value for all users as null.
- Add year, month and day columns based on generated timestamps

- **Data flow diagram:**



- **Class diagram:**



BronzeNetworkDataObject

+ ID: str = BronzeNetworkDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>

+ write(path:str = None, partition_columns: list<str> = None)

BronzeSyntheticDiariesDataObject

+ ID: str = BronzeSyntheticDiariesDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>

+ write(path:str = None, partition_columns: list<str> = None)

SyntheticEvents

+ COMPONENT_ID: str = SyntheticEvents
+ clear_destination_directory: bool
+ event_freq_stays: int
+ event_freq_moves: int
+ closest_cell_distance_max: int
+ closest_cell_distance_max_for_errors: int
+ error_location_probability: int
+ error_location_distance_min: int
+ error_location_distance_max: int
+ cartesian_crs: int
+ error_cell_id_probability: float
+ maximum_number_of_cells_for_event: float
+ mcc: int

+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
+ generate_event_timestamps_for_moves(stays_sdf: DataFrame, event_freq_moves: int, cartesian_crs: int): DataFrame
+ generate_event_timestamps_for_stays(stays_sdf: DataFrame, event_freq_stays: int, cartesian_crs: int): DataFrame
+ generate_locations_for_moves(event_timestamps_df: DataFrame, cartesian_crs: int): DataFrame
+ add_cell_ids_to_locations(events_with_locations_df: DataFrame, cells_df: DataFrame, max_n_of_cells: int, seed: int): DataFrame
+ generate_location_errors(records_sdf: DataFrame, error_location_distance_max: float, error_location_distance_min: float, closest_cell_distance_max: float, cartesian_crs: int, seed: int): DataFrame
+ generate_records_with_non_existant_cell_ids(records_sdf: DataFrame, cells_sdf: DataFrame): DataFrame

BronzeEventDataObject

+ ID: str = BronzeEventDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>

+ write(path:str = None, partition_columns: list<str> = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
/multimno/
└── src
    └── components
        └── ingestion
            └── synthetic
                └── synthetic_events.py
```

  - synthetic_events.py contains one class named SyntheticEvents which is a subclass of Component. It overrides the following methods:
    - The __init__ method first call its parent’s __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - The transform method handles all the logic behind the component.
  - The SyntheticEvents component also has the following methods:
    - generate_event_timestamps_for_moves handles generation of event timestamps for moves.
    - generate_event_timestamps_for_stays handles generation of event timestamps for stays
    - generate_locations_for_moves handles generation of point geometry for moves.
    - add_cell_ids_to_locations handles joining cell ids to generated longitude and latitude values, given the parameter closest_cell_distance_max .
    - generate_location_errors handles the generation of point geometry, using the given seed parameter for randomized processes (such as exact range of distance, and direction of offset), and the parameters error_location_distance_max and error_location_distance_min .
    - generate_records_with_non_existant_cell_ids handles the generation of records with cell_ids that do not exist in the synthetic network data object, yet follow the format of a cell id syntactically.

5.2.18 GRIDENRICHMENT

5.2.18.1 MODULE DESCRIPTION

- **Module Name:** GridEnrichment
- **Objectives:** Add additional attributes to reference grid.
- **Functionality:** The component uses landuse and transportation data objects to calculate following metrics for each grid tile:
  - Landuse prior probabilities of the distribution of mobile devices.
  - Environment coefficient for dynamic path loss exponent calculation for cell signal strength modeling.

  Functionality specification: 3.2.18 Grid Enrichment
- **Data Inputs and Outputs:**
  - Inputs:
    - I.32 Landuse
    - I.33 Transportation
    - I.11 Reference Grid
  - Outputs:
    - I.31 Enriched Grid

5.2.18.2DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**

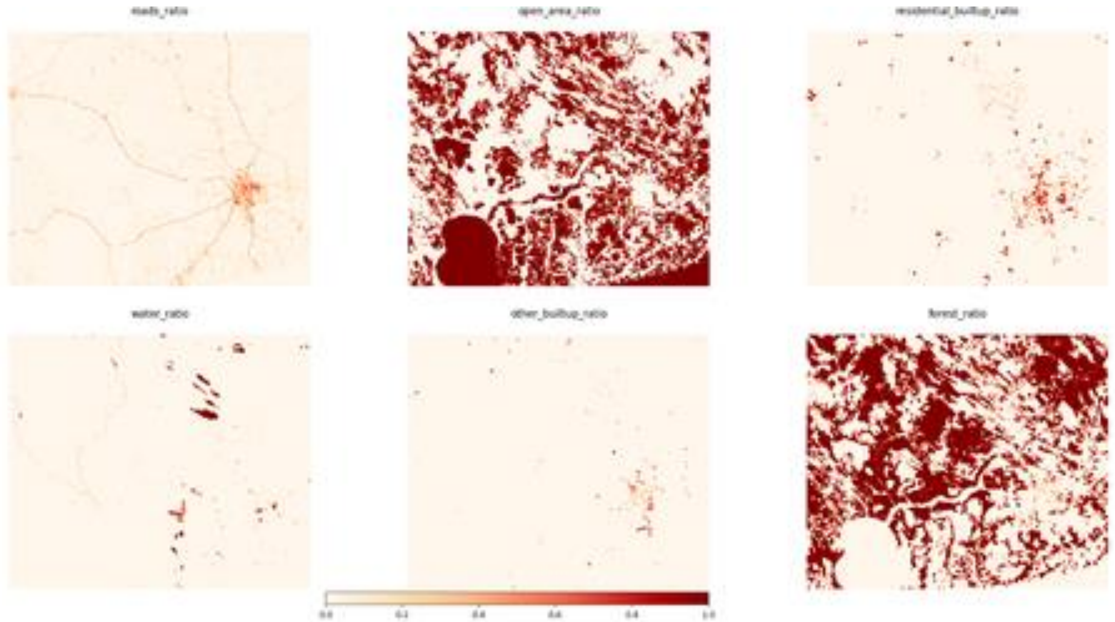
Initialisation:

1. Read configurations parameters for performing land cover enrichment, transportation category buffers, landuse types weights for prior calculation, landuse types weights for environment coefficient calculation and Spark checkpoint directory.
2. Clear the destination directory if configured.
3. Load input data objects for grid, transportation, and land use data.
4. Initialise the output data object for the enriched grid.

Processing:

For each quadkey partition in input grid data object:

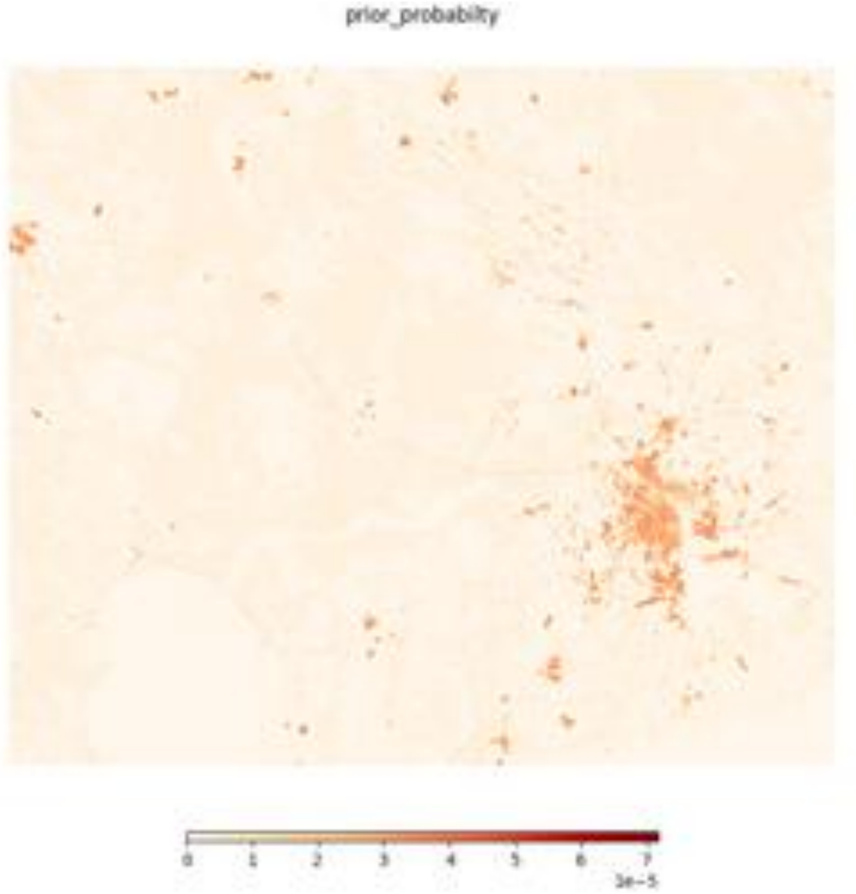
1. Prepare input data.
    1. Filter transportation data to the extent of the current quadkey.
    2. Perform buffer operation using different buffer distances for different road types based on configuration to convert transportation lines to polygons.
    3. Filter landuse data to the extent of the current quadkey.
    4. Cut landuse polygons with transportation polygons and merge them together so that transportation polygons do not overlap landuse polygons.
2. Intersect grid tiles geometry with combined landuse polygons and calculate ratios of landuse classes which are intersected with a grid tile to the total area of a grid tile.



    a. Assign weights for prior probability to each landuse category based on configuration parameter and calculate weighed sum of landuse categories per grid tile.
    b. Assign weights for path loss exponent coefficient to each landuse category based on configuration parameter an calculate weighed sums of landuse category ratios per grid tile to get environment Path Loss Exponent coefficient
    c. Persists the current results with a checkpoint and clear cache to reduce memory footprint.

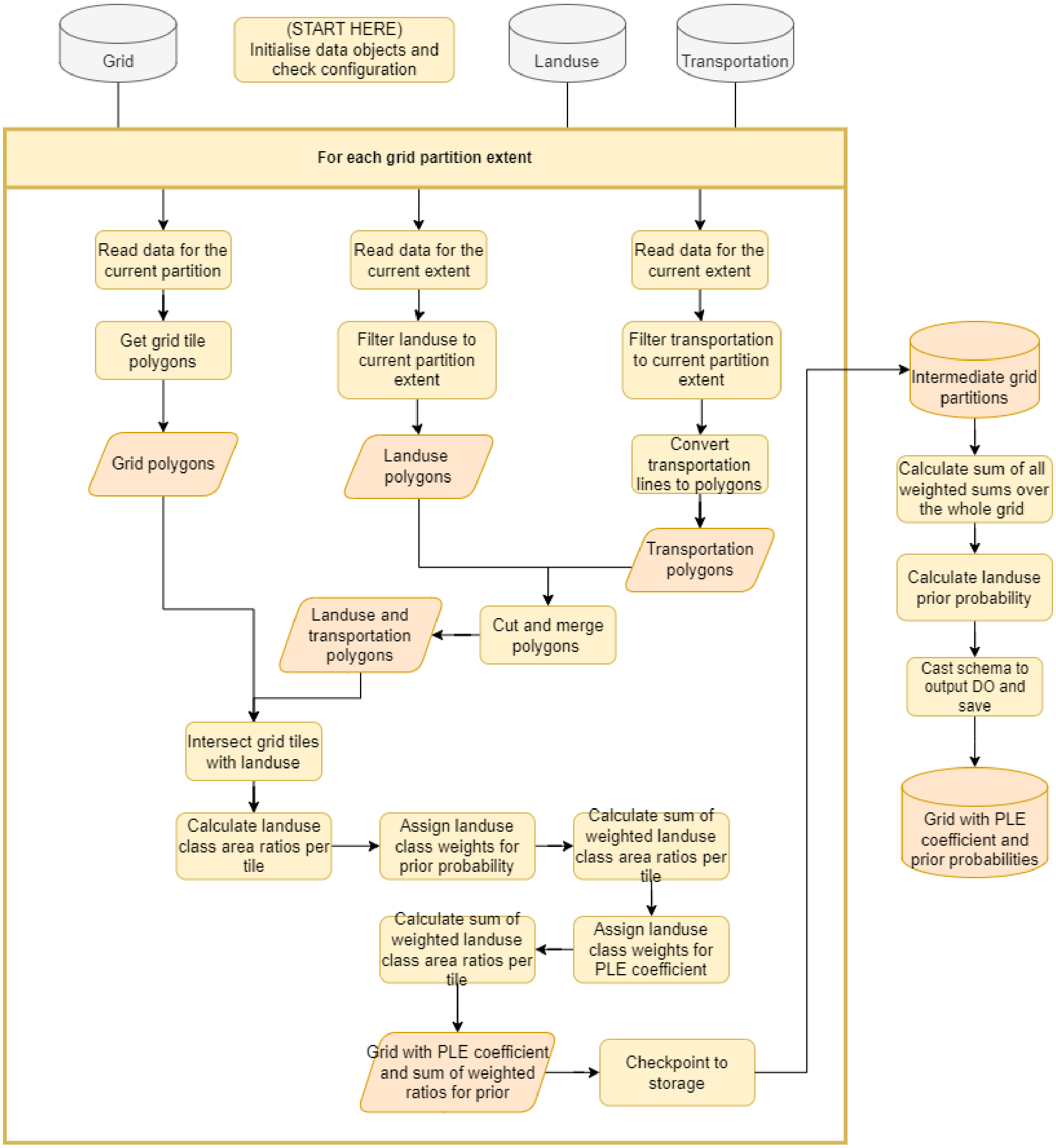
3. Combine all persisted result parts and remove potential duplicates.
4. Calculate total sum of weighed sums of landuse categories over all grid tiles.

5. Divide weighed sums of landuse categories in grid tiles by total sum to get landuse prior probability values

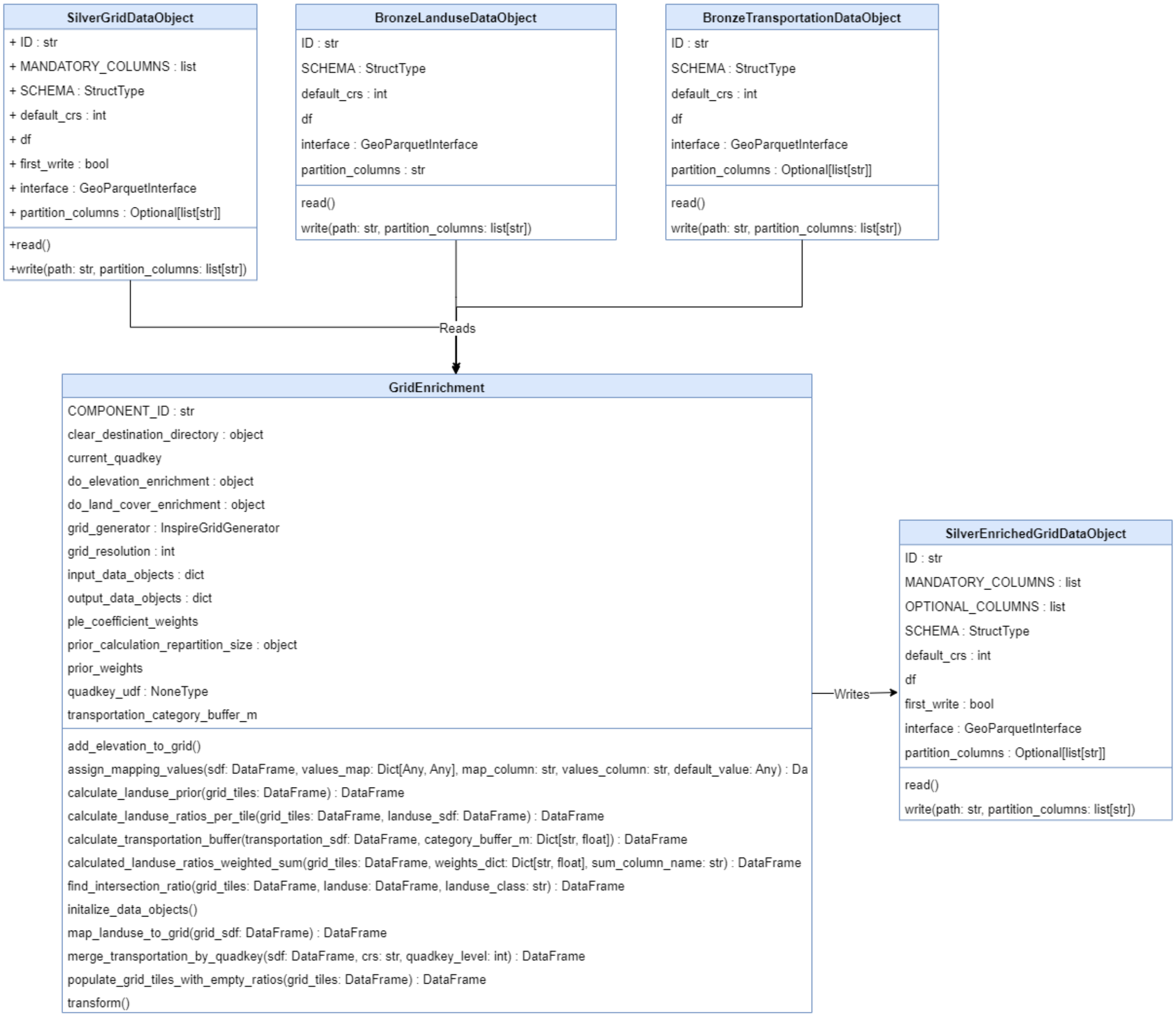


6. Order and repartition resulted grid by quadkey.
7. Apply schema casting and add missing columns to match the output data object's schema.
8. Save to storage partition by quadkey.

- **Data flow diagram:**


Grid
(START HERE) Initialise data objects and check configuration
Landuse
Transportation
For each grid partition extent
Read data for the current partition
Read data for the current extent
Read data for the current extent
Get grid tile polygons
Filter landuse to current partition extent
Filter transportation to current partition extent
Grid polygons
Landuse polygons
Convert transportation lines to polygons
Transportation polygons
Landuse and transportation polygons
Cut and merge polygons
Intersect grid tiles with landuse
Calculate landuse class area ratios per tile
Assign landuse class weights for prior probability
Calculate sum of weighted landuse class area ratios per tile
Assign landuse class weights for PLE coefficient
Calculate sum of weighted landuse class area ratios per tile
Grid with PLE coefficient and sum of weighted ratios for prior
Checkpoint to storage
Intermediate grid partitions
Calculate sum of all weighted sums over the whole grid
Calculate landuse prior probability
Cast schema to output DO and save
Grid with PLE coefficient and prior probabilities

- **Class diagram:**


SilverGridDataObject
+ ID : str
+ MANDATORY_COLUMNS : list
+ SCHEMA : StructType
+ default_crs : int
+ df
+ first_write : bool
+ interface : GeoParquetInterface
+ partition_columns : Optional[list[str]]
+read()
+write(path: str, partition_columns: list[str])
BronzeLanduseDataObject
ID : str
SCHEMA : StructType
default_crs : int
df
interface : GeoParquetInterface
partition_columns : str
read()
write(path: str, partition_columns: list[str])
BronzeTransportationDataObject
ID : str
SCHEMA : StructType
default_crs : int
df
interface : GeoParquetInterface
partition_columns : Optional[list[str]]
read()
write(path: str, partition_columns: list[str])
Reads
GridEnrichment
COMPONENT_ID : str
clear_destination_directory : object
current_quadkey
do_elevation_enrichment : object
do_land_cover_enrichment : object
grid_generator : InspireGridGenerator
grid_resolution : int
input_data_objects : dict
output_data_objects : dict
ple_coefficient_weights
prior_calculation_repartition_size : object
prior_weights
quadkey_udf : NoneType
transportation_category_buffer_m
add_elevation_to_grid()
assign_mapping_values(sdf: DataFrame, values_map: Dict[Any, Any], map_column: str, values_column: str, default_value: Any) : Da
calculate_landuse_prior(grid_tiles: DataFrame) : DataFrame
calculate_landuse_ratios_per_tile(grid_tiles: DataFrame, landuse_sdf: DataFrame) : DataFrame
calculate_transportation_buffer(transportation_sdf: DataFrame, category_buffer_m: Dict[str, float]) : DataFrame
calculated_landuse_ratios_weighted_sum(grid_tiles: DataFrame, weights_dict: Dict[str, float], sum_column_name: str) : DataFrame
find_intersection_ratio(grid_tiles: DataFrame, landuse: DataFrame, landuse_class: str) : DataFrame
initalize_data_objects()
map_landuse_to_grid(grid_sdf: DataFrame) : DataFrame
merge_transportation_by_quadkey(sdf: DataFrame, crs: str, quadkey_level: int) : DataFrame
populate_grid_tiles_with_empty_ratios(grid_tiles: DataFrame) : DataFrame
transform()
Writes
SilverEnrichedGridDataObject
ID : str
MANDATORY_COLUMNS : list
OPTIONAL_COLUMNS : list
SCHEMA : StructType
default_crs : int
df
first_write : bool
interface : GeoParquetInterface
partition_columns : Optional[list[str]]
read()
write(path: str, partition_columns: list[str])

- **Code Structure:**

The code structure follows the format set by the core package, and the general repository structure.
The location of the module script in the repository is as follows:

```
multimno
    └── components
        └── execution
            └── grid_enrichment
                └── grid_enrichment.py
```

grid_enrichment.py contains one class named GridEnrichmentwhich is a subclass of Component.
The GridEnrichment class overrides transform method of base Component class.

5.2.19 GEOZONESGRIDMAPPING

5.2.19.1 MODULE DESCRIPTION

- **Module Name:** GeozonesGridMapping
- **Objectives:** Map given geographic zone datasets to reference grid tile centroids.
- **Functionality:** For each given zoning dataset the component enriches grid tiles with information about zoning unit which they are intersecting by performing spatial join of zoning polygons to grid centroids. In case of hierarchical zoning system, spatial join is performed on the lowest level of hierarchy. Higher level zone IDs are then derived from parent_id column of a zoning dataset and combined into hierarchical id. Functionality specification:
    - 3.2.19 GeozonesGridMapping
- **Data Inputs and Outputs:**
    - Inputs:
        - I.11 Reference Grid
        - I.35 Geographic Zones
        - I.34 Administrative Units
    - Outputs:
        - I.36 Zones – Grid Map

5.2.19.2DEVELOPMENT DESIGN:

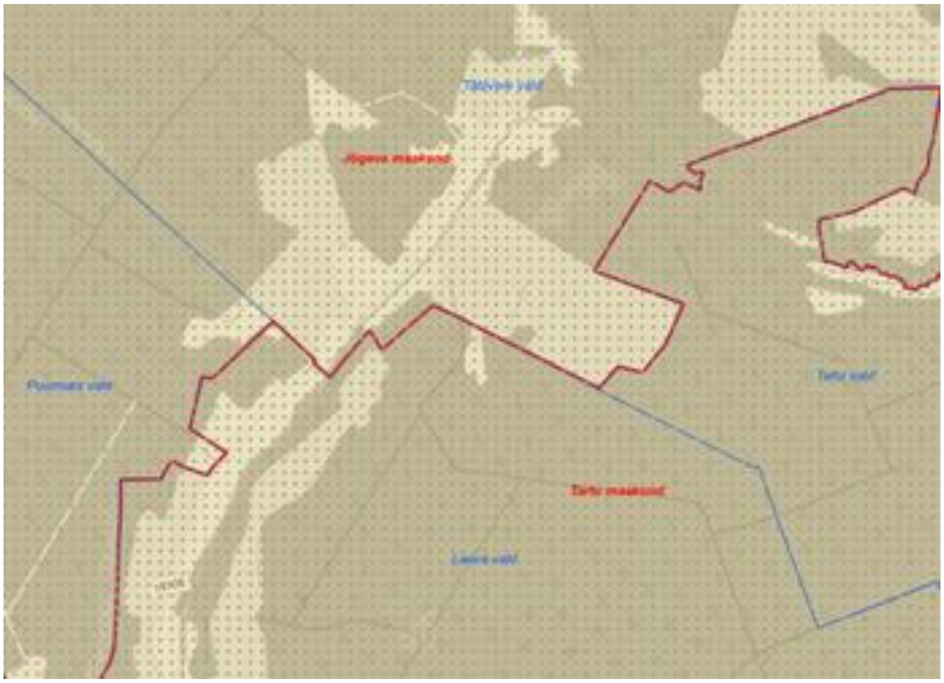
- **Key Algorithms/Processes:**

Initialisation:

1. Read configuration parameters for selecting zoning dataset IDs to perform mapping, zoning type (e.g., administrative units or other geographic zones).
2. Clear the destination directory if configured.
3. Load input data objects for grid, other geographic zones or administrative units based on the selected zoning type.
4. Initialize the output data object for the geozones grid map.

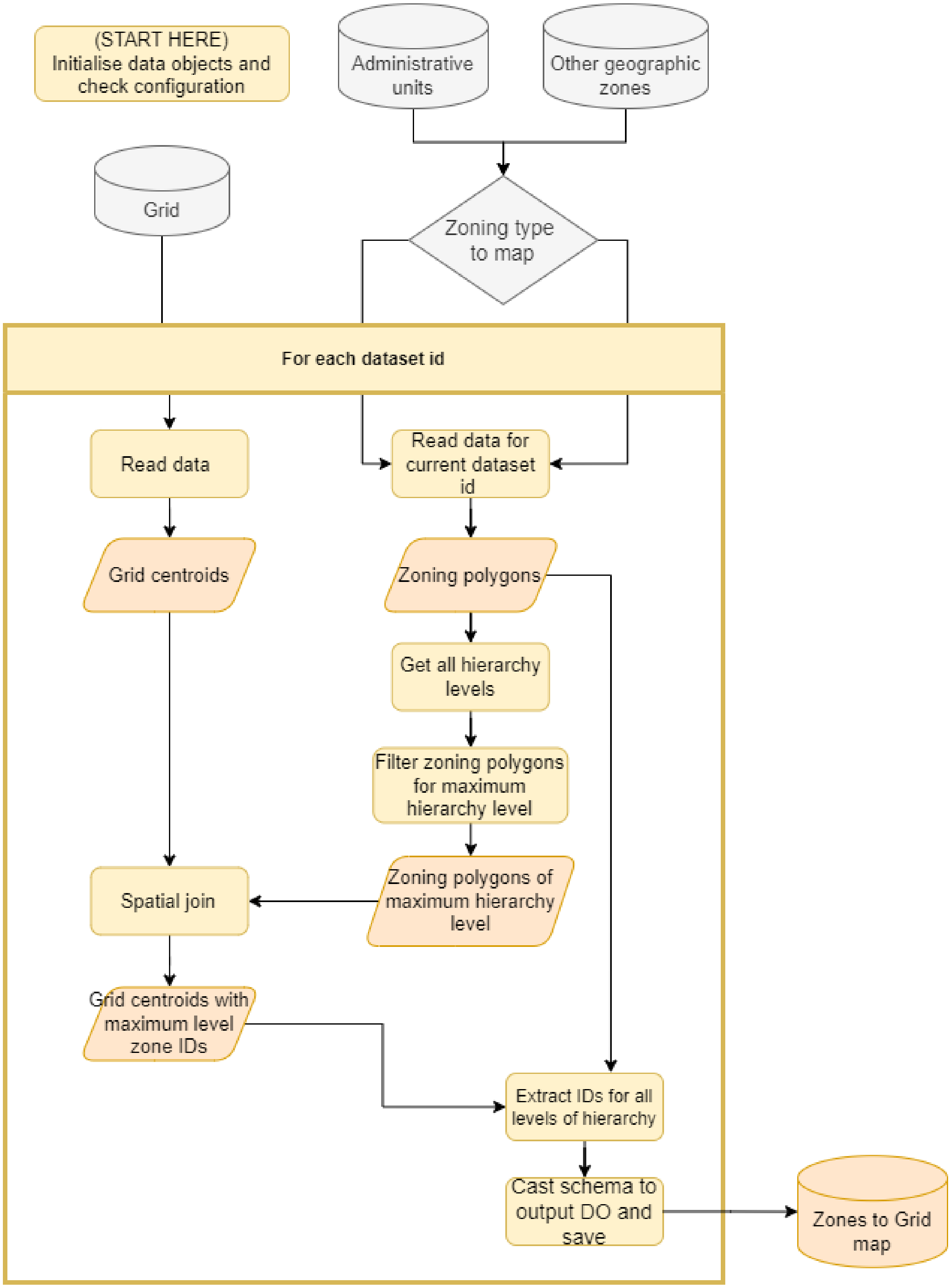
Processing:

1. For each dataset ID following steps are performed:
    1. Filter the current zoning dataset by the dataset ID from corresponding data object.
    2. Retrieve the hierarchy levels of zoning units.
    3. Map zoning units on the maximum level of hierarchy to the grid by performing spatial join of grid centroids to zoning polygons.

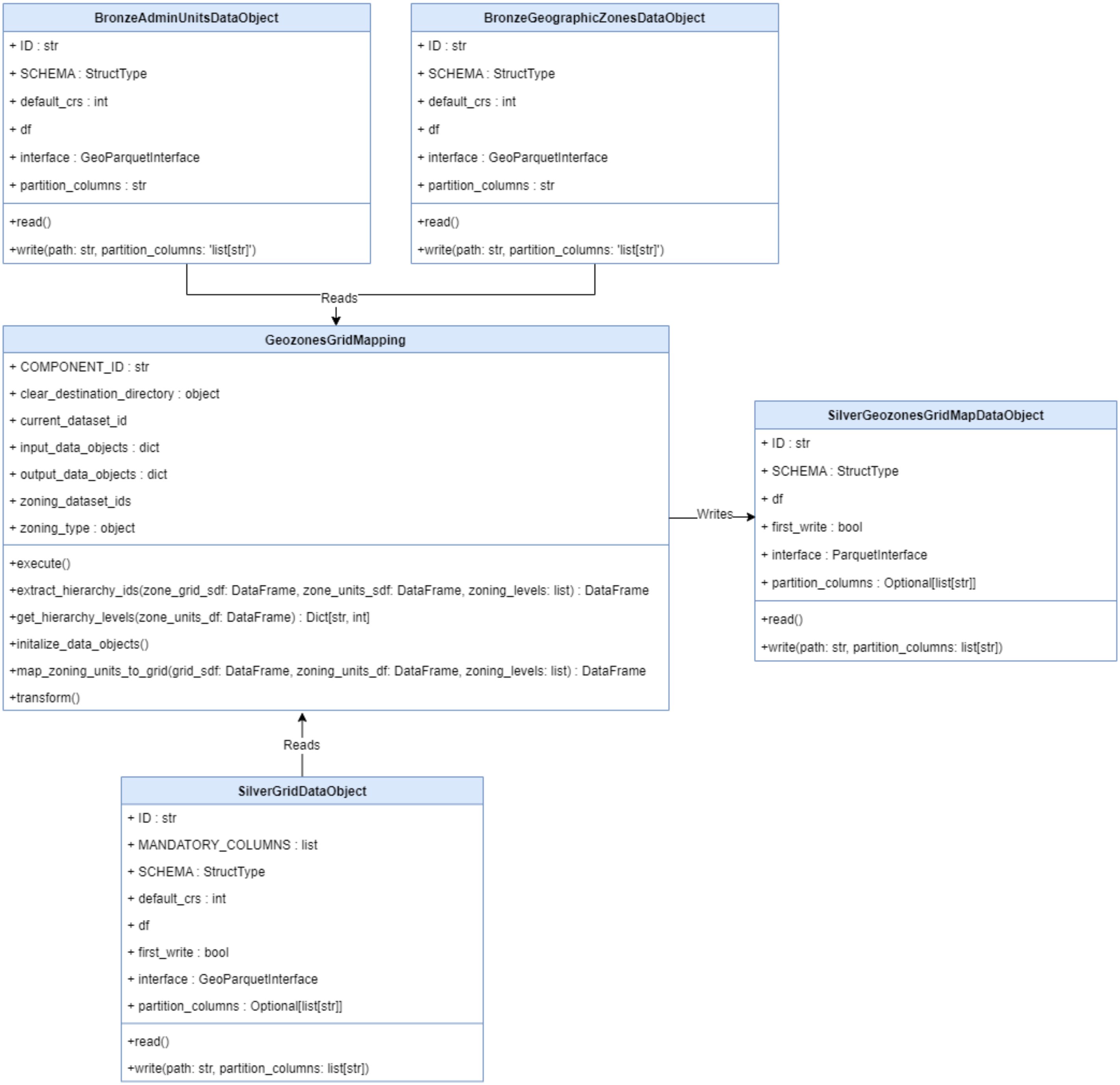


4. Extract IDs for all levels of zoning hierarchy and combine them into hierarchical ID.
5. Assign year, month, and day columns from the zoning dataset to the grid.
6. Apply schema casting to match the output data object's schema and write to storage partitioned by quadkey.

- **Data flow diagram:**


(START HERE)
Initialise data objects and check configuration
Administrative units
Other geographic zones
Grid
Zoning type to map
For each dataset id
Read data
Read data for current dataset id
Grid centroids
Zoning polygons
Get all hierarchy levels
Filter zoning polygons for maximum hierarchy level
Zoning polygons of maximum hierarchy level
Spatial join
Grid centroids with maximum level zone IDs
Extract IDs for all levels of hierarchy
Cast schema to output DO and save
Zones to Grid map

- **Class diagram:**


BronzeAdminUnitsDataObject
+ ID : str
+ SCHEMA : StructType
+ default_crs : int
+ df
+ interface : GeoParquetInterface
+ partition_columns : str
+read()
+write(path: str, partition_columns: 'list[str]')
BronzeGeographicZonesDataObject
+ ID : str
+ SCHEMA : StructType
+ default_crs : int
+ df
+ interface : GeoParquetInterface
+ partition_columns : str
+read()
+write(path: str, partition_columns: 'list[str]')
Reads
GeozonesGridMapping
+ COMPONENT_ID : str
+ clear_destination_directory : object
+ current_dataset_id
+ input_data_objects : dict
+ output_data_objects : dict
+ zoning_dataset_ids
+ zoning_type : object
+execute()
+extract_hierarchy_ids(zone_grid_sdf: DataFrame, zone_units_sdf: DataFrame, zoning_levels: list) : DataFrame
+get_hierarchy_levels(zone_units_df: DataFrame) : Dict[str, int]
+initalize_data_objects()
+map_zoning_units_to_grid(grid_sdf: DataFrame, zoning_units_df: DataFrame, zoning_levels: list) : DataFrame
+transform()
Writes
SilverGeozonesGridMapDataObject
+ ID : str
+ SCHEMA : StructType
+ df
+ first_write : bool
+ interface : ParquetInterface
+ partition_columns : Optional[list[str]]
+read()
+write(path: str, partition_columns: list[str])
Reads
SilverGridDataObject
+ ID : str
+ MANDATORY_COLUMNS : list
+ SCHEMA : StructType
+ default_crs : int
+ df
+ first_write : bool
+ interface : GeoParquetInterface
+ partition_columns : Optional[list[str]]
+read()
+write(path: str, partition_columns: list[str])

- **Code Structure:**

The code structure follows the format set by the core package, and the general repository structure.
The location of the module script in the repository is as follows:

```
multimno
    └── components
        └── execution
            └── geozones_grid_mapping
                └── geozones_grid_mapping.py
```

geozones_grid_mapping.py contains one class named GeozonesGridMappingwhich is a subclass of Component. The GeozonesGridMappingclass overrides transform method of base Component class.

5.2.20 PRESENTPOPULATIONESTIMATION

5.2.20.1 MODULE DESCRIPTION

- **Module Name:** PresentPopulationEstimation
- **Objectives:** This module estimates the present population (number of actual people) in each spatial unit at each time point (fixed timestamp). The spatial unit is either a grid tile or a zone/municipality (collection of grid tiles).
- **Functionality:** This module implements the methodology described in D2.2 method 14.1, variant 1.
- **Data Inputs and Outputs:**
  - Inputs:
    - I.16 MNO Event Data – Semantically Cleaned
    - I.15 Cell Connection and Posterior Probabilities
    - I.28 INSPIRE Grid
    - I.36 Zones – Grid Map
  - Outputs:
    - I.42 Present Population
    - I.41 Present Population Zone Level
- Configuration parameters
  - *tolerance_period_s*: Maximum allowed time difference for an event to be included in a time point.
  - *data_period_start*: Starting bound when to start time point generation. The first time point is created at this timestamp.
  - *data_period_end*: Ending bound when to end time point generation. No time points are generated later than this timestamp. A time point can be generated on this exact timestamp.
  - *time_point_gap_s*: Number of seconds between two generated time points.
  - *nr_of_user_id_partitions:* Total number of user_id_modulo partitions.
  - *nr_of_user_id_partitions_per_slice*: Number of user_id_modulo partitions to process in one batch.
  - *max_iterations*: Maximum number of iterations allowed for the Bayesian process.
  - *min_difference_threshold*: Minimum total difference between Bayesian process prior and posterior values needed to continue iterations of the process.
  - *output_aggregation_level*: Supported values: ‘grid’, ‘zone’. Determines which level the results are aggregated to.
  - *zoning_dataset_id*: Name of the zoning dataset to use. Only needed when *output_aggregation_level* is ‘zone’.
  - *zoning_hierarchical_level*: Level of hierarchical zoning to aggregate results to. Only needed when *output_aggregation_level* is ‘zone’.

5.2.20.2 DEVELOPMENT DESIGN

\ TIMELINE QUANTIZATION

Starting from *data_period_start* generate one timestamp after each *time_point_gap_s* until *data_period_end* is reached. A time point can be generated at exactly *data_period_end*, but not later.

timestamp
2023-07-14 10:00:00
2023-07-14 11:00:00
2023-07-14 12:00:00
2023-07-14 13:00:00
2023-07-14 14:00:00

\ DETERMINING RELEVANT EVENT DATA FOR THE TIME POINT

For each time point at timestamp *t*, the events included in its calculation are all events within the window *[t-tolerance_period_s, t+tolerance_period_s]*.
When selecting the data, first date-level filtering is applied to make use of date-partitioned storage of event data. Then timestamp-level filtering is applied to select the exact events.

\ ESTIMATION OF DEVICE COUNT PER CELL FOR TIME POINT

For each time point, for each cell, calculate the number of unique devices present. Both domestic and inbound data should be included (if available).

cell_id	count	timestamp
1	25436	2023-07-14 14:00:00
2	5342	2023-07-14 14:00:00
3	304334	2023-07-14 14:00:00
4	145755	2023-07-14 14:00:00

\ ITERATIVE PROCESS FOR THE ESTIMATION OF POPULATION PER GRID TILE

For each time point, for each grid tile, estimate the present population using cell weighted counts and cell connection probabilities. This is an iterative Bayesian procedure.
Determine grid to cell probabilities by selecting the cell to grid connection probability data that matches the current time point. Then for each grid tile, sum and normalize the cell to grid probabilities to determine the grid to cell probability.
For the iterative process, first initialize the population values of each grid tile: calculate the sum of *weighted_count* of all cells, then for each grid tile, set the initial *population* value to *weighted_counts_sum/n_grid*, where *n_grid* is the total number of grid tiles.
In each iteration:

1. For each (*cell_id*, *grid_id*) pair, calculate the value *a* as *population*grid_prob*, where *population* is this cell's population value and *grid_prob* is the grid to cell connection probability of this grid_id.

2. For each *cell_id*, calculate the sum *sum_a* across all of its (*cell_id*, *grid_id, a*) rows. Get (*cell_id, sum_a)* rows.
3. Normalize *a*: for each (*cell_id*, *grid_id, a*) row, join with (*cell_id, sum_a)* on cell_id. Divide *a* by *sum_a* and replace *a* with the new value.
4. Apply weighting by count: for each (*cell_id*, *grid_id, a*) row, join with device count per cell data (*cell_id, count*) on cell_id. Multiply *a* with *count* and replace *a* with the new value.
5. For each *grid_id*, calculate value *new_population* as the sum of *a* across all matching (*cell_id*, *grid_id, a*) rows.
6. Determine difference: for each *grid_id*, calculate the absolute difference between *population* and *new_population*. Calculate *sum_diff* as the sum of absolute differences across all *grid_id*s.
7. Replace *population* with *new_population*.
8. Check for iteration conditions. Repeat the loop if *sum_diff* is above threshold **and** iteration count is below threshold.

grid_id	population	timestamp
1	654	2023-07-14 14:00:00
2	234	2023-07-14 14:00:00
3	1654	2023-07-14 14:00:00

\ IF AGGREGATING BY GRID, THEN THE RESULTS ARE DONE.

Write results partitioned by day, month, year calculated from the time point timestamp.

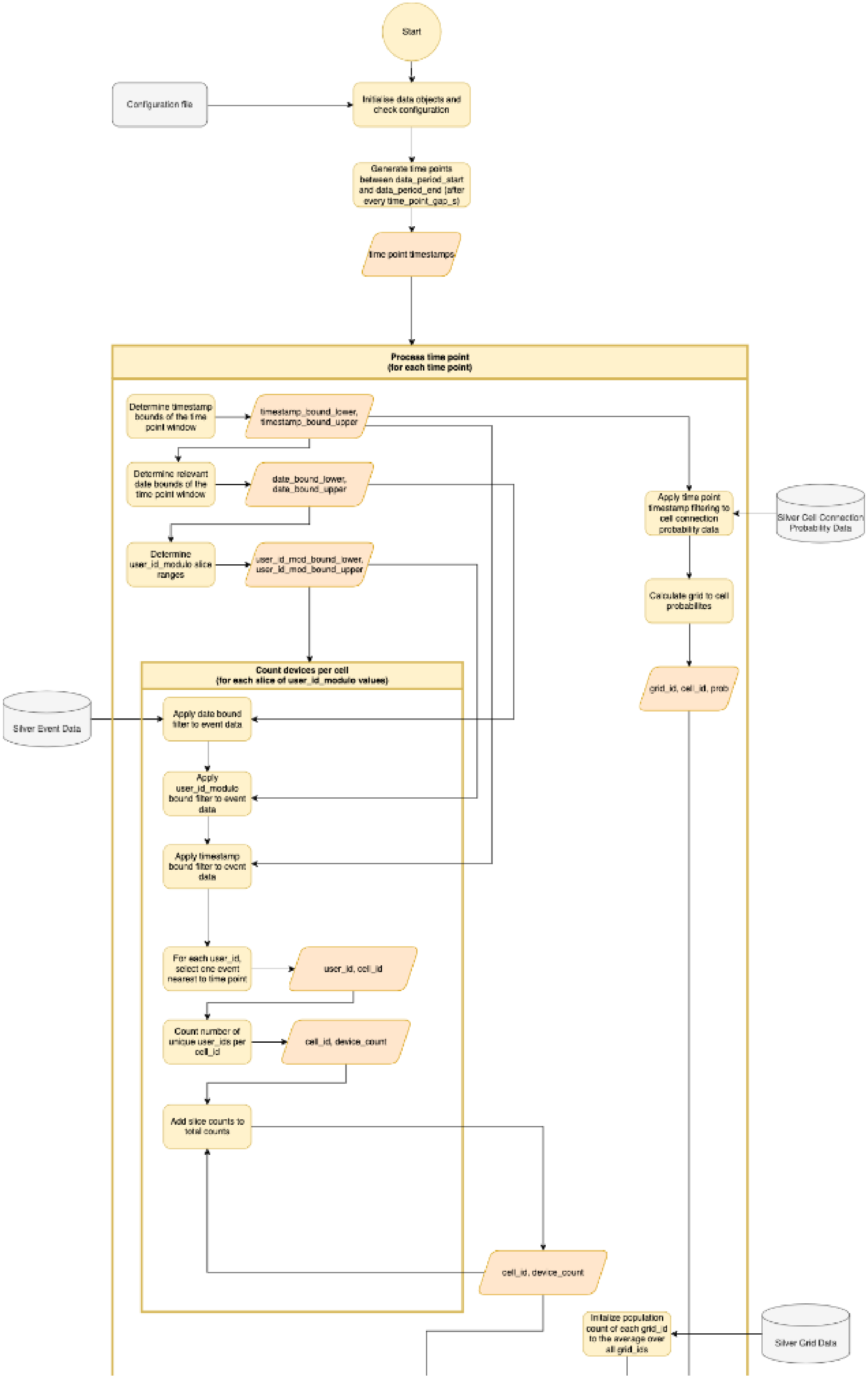
\ IF AGGREGATING BY ZONE, MAP GRIDS TO ZONE IDS.

From the zones to grid mapping dataset, select rows where dataset_id matches *zoning_dataset_id*. From the hierarchical_id column, extract the id from level *zoning_hierarchical_level* at set it as the zone_id. Join the population per grid data with the zones to grid mapping data on grid_id, then calculate the sum of population grouped by zone_id.

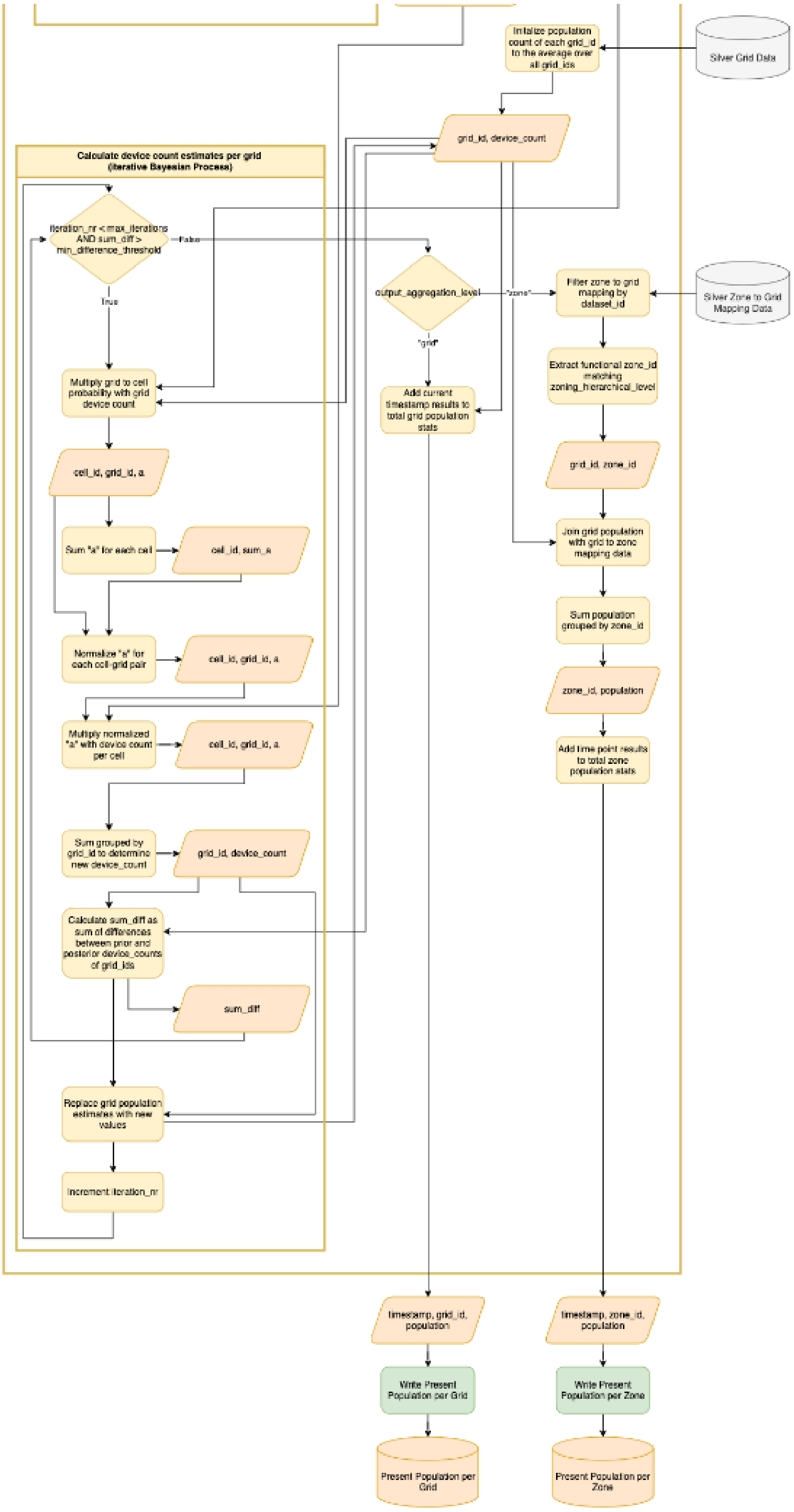
\ WRITE RESULTS

If aggregating by grid, the output data object is I.42 Present Population.
If aggregating by zone, the output data object is I.41 Present Population Zone Level.
In either case, write results partitioned by day, month, year calculated from the time point timestamp.

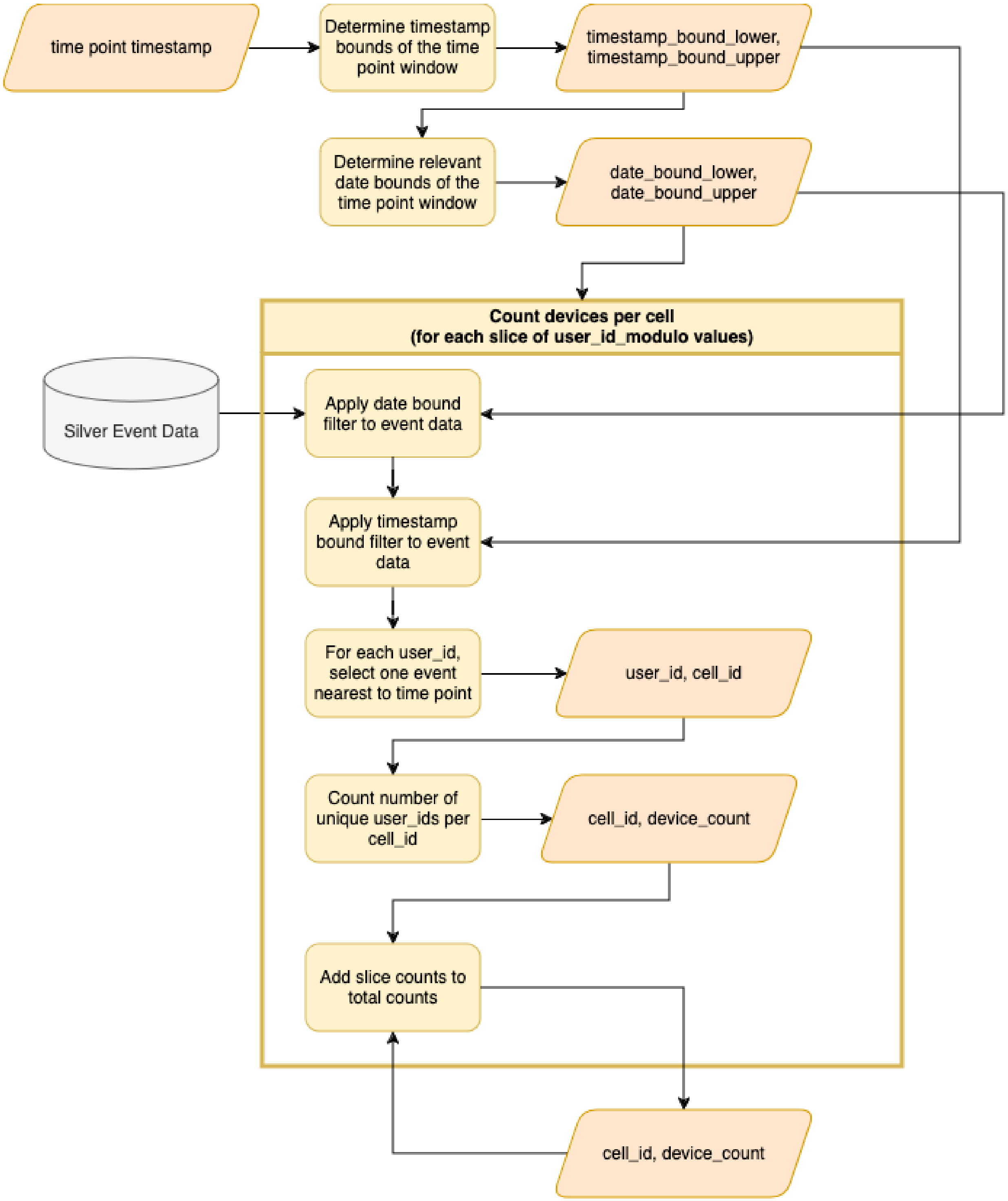
\ DATA FLOW DIAGRAM (PART 1)


Start
Configuration file
Initialise data objects and check configuration
Generate time points between data_period_start and data_period_end (after every time_point_gap_s)
time point timestamps
Process time point
(for each time point)
Determine timestamp bounds of the time point window
timestamp_bound_lower, timestamp_bound_upper
Determine relevant date bounds of the time point window
date_bound_lower, date_bound_upper
Determine user_id_modulo slice ranges
user_id_mod_bound_lower, user_id_mod_bound_upper
Apply time point timestamp filtering to cell connection probability data
Silver Cell Connection Probability Data
Calculate grid to cell probabilites
grid_id, cell_id, prob
Count devices per cell
(for each slice of user_id_modulo values)
Silver Event Data
Apply date bound filter to event data
Apply user_id_modulo bound filter to event data
Apply timestamp bound filter to event data
For each user_id, select one event nearest to time point
user_id, cell_id
Count number of unique user_ids per cell_id
cell_id, device_count
Add slice counts to total counts
cell_id, device_count
Initalize population count of each grid_id to the average over all grid_ids
Silver Grid Data

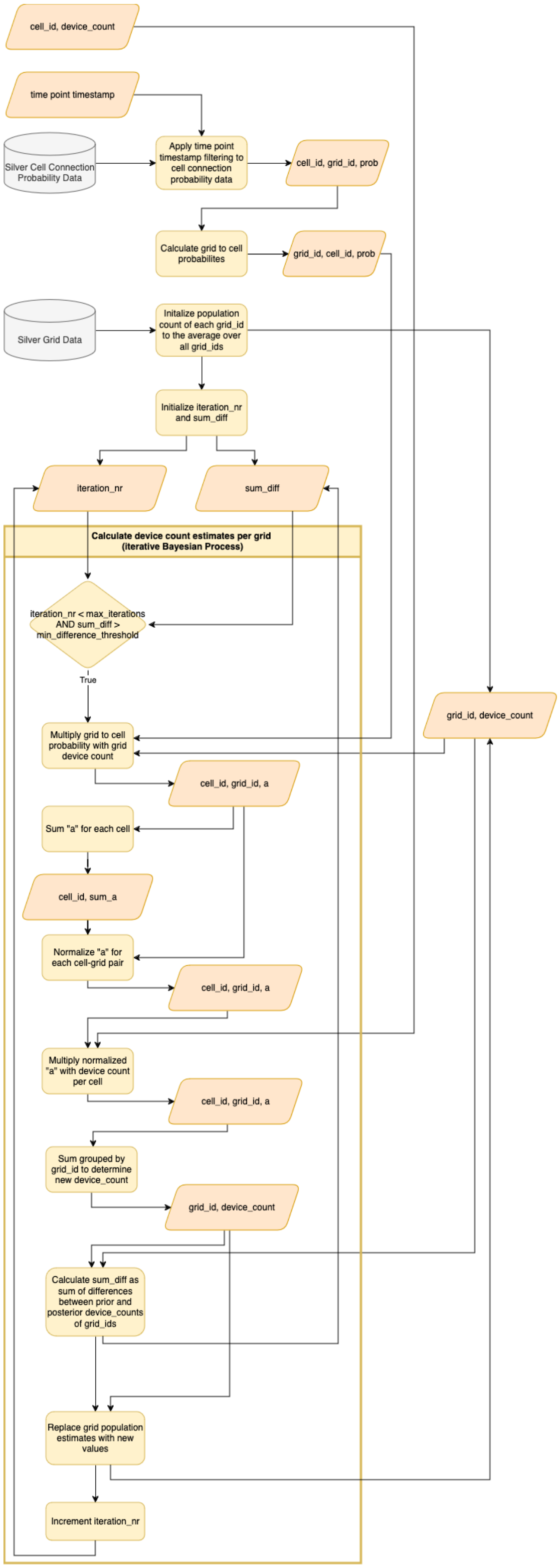
\ DATA FLOW DIAGRAM (PART 2)


Initalize population count of each grid_id to the average over all grid_ids
Silver Grid Data
grid_id, device_count
Calculate device count estimates per grid (iterative Bayesian Process)
iteration_nr < max_iterations AND sum_diff > min_difference_threshold
False
True
output_aggregation_level
"zone"
"grid"
Filter zone to grid mapping by dataset_id
Silver Zone to Grid Mapping Data
Multiply grid to cell probability with grid device count
Add current timestamp results to total grid population stats
Extract functional zone_id matching zoning_hierarchical_level
cell_id, grid_id, a
grid_id, zone_id
Sum "a" for each cell
cell_id, sum_a
Join grid population with grid to zone mapping data
Sum population grouped by zone_id
Normalize "a" for each cell-grid pair
cell_id, grid_id, a
zone_id, population
Multiply normalized "a" with device count per cell
cell_id, grid_id, a
Add time point results to total zone population stats
Sum grouped by grid_id to determine new device_count
grid_id, device_count
Calculate sum_diff as sum of differences between prior and posterior device_counts of grid_ids
sum_diff
Replace grid population estimates with new values
Increment iteration_nr
timestamp, grid_id, population
timestamp, zone_id, population
Write Present Population per Grid
Write Present Population per Zone
Present Population per Grid
Present Population per Zone

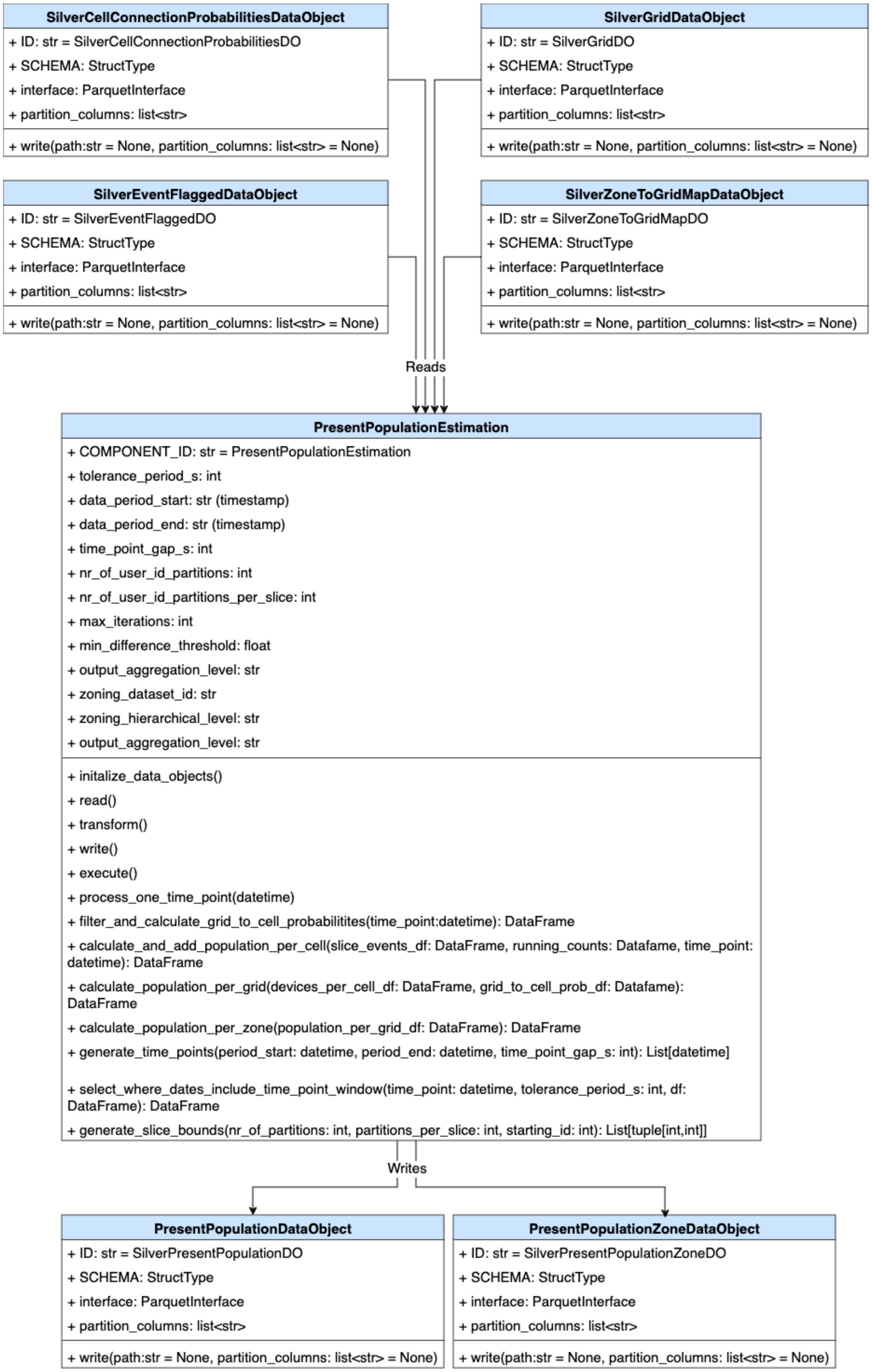
\ DATA FLOW DIAGRAM (COUNT DEVICES PER CELL)


time point timestamp
Determine timestamp bounds of the time point window
timestamp_bound_lower, timestamp_bound_upper
Determine relevant date bounds of the time point window
date_bound_lower, date_bound_upper
Count devices per cell (for each slice of user_id_modulo values)
Silver Event Data
Apply date bound filter to event data
Apply timestamp bound filter to event data
For each user_id, select one event nearest to time point
user_id, cell_id
Count number of unique user_ids per cell_id
cell_id, device_count
Add slice counts to total counts
cell_id, device_count

\ DATA FLOW DIAGRAM (CALCULATE DEVICES PER GRID)


cell_id, device_count
time point timestamp
Silver Cell Connection Probability Data
Apply time point timestamp filtering to cell connection probability data
cell_id, grid_id, prob
Calculate grid to cell probabilites
grid_id, cell_id, prob
Silver Grid Data
Initalize population count of each grid_id to the average over all grid_ids
Initialize iteration_nr and sum_diff
iteration_nr
sum_diff
Calculate device count estimates per grid (iterative Bayesian Process)
iteration_nr < max_iterations AND sum_diff > min_difference_threshold
True
grid_id, device_count
Multiply grid to cell probability with grid device count
cell_id, grid_id, a
Sum "a" for each cell
cell_id, sum_a
Normalize "a" for each cell-grid pair
cell_id, grid_id, a
Multiply normalized "a" with device count per cell
cell_id, grid_id, a
Sum grouped by grid_id to determine new device_count
grid_id, device_count
Calculate sum_diff as sum of differences between prior and posterior device_counts of grid_ids
Replace grid population estimates with new values
Increment iteration_nr

\ CLASS DIAGRAM



```
SilverCellConnectionProbabilitiesDataObject
+ ID: str = SilverCellConnectionProbabilitiesDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
```

```
SilverGridDataObject
+ ID: str = SilverGridDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
```

```
SilverEventFlaggedDataObject
+ ID: str = SilverEventFlaggedDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
```

```
SilverZoneToGridMapDataObject
+ ID: str = SilverZoneToGridMapDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
```

```
PresentPopulationEstimation
+ COMPONENT_ID: str = PresentPopulationEstimation
+ tolerance_period_s: int
+ data_period_start: str (timestamp)
+ data_period_end: str (timestamp)
+ time_point_gap_s: int
+ nr_of_user_id_partitions: int
+ nr_of_user_id_partitions_per_slice: int
+ max_iterations: int
+ min_difference_threshold: float
+ output_aggregation_level: str
+ zoning_dataset_id: str
+ zoning_hierarchical_level: str
+ output_aggregation_level: str
+ initalize_data_objects()
+ read()
+ transform()
+ write()
+ execute()
+ process_one_time_point(datetime)
+ filter_and_calculate_grid_to_cell_probabilitites(time_point:datetime): DataFrame
+ calculate_and_add_population_per_cell(slice_events_df: DataFrame, running_counts: Datafame, time_point: datetime): DataFrame
+ calculate_population_per_grid(devices_per_cell_df: DataFrame, grid_to_cell_prob_df: Datafame): DataFrame
+ calculate_population_per_zone(population_per_grid_df: DataFrame): DataFrame
+ generate_time_points(period_start: datetime, period_end: datetime, time_point_gap_s: int): List[datetime]
+ select_where_dates_include_time_point_window(time_point: datetime, tolerance_period_s: int, df: DataFrame): DataFrame
+ generate_slice_bounds(nr_of_partitions: int, partitions_per_slice: int, starting_id: int): List[tuple[int,int]]
```

```
PresentPopulationDataObject
+ ID: str = SilverPresentPopulationDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
```

```
PresentPopulationZoneDataObject
+ ID: str = SilverPresentPopulationZoneDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list<str>
+ write(path:str = None, partition_columns: list<str> = None)
```

\ CODE STRUCTURE

The code structure follows the format set by the core package, and the general repository structure.
The location of the module script in the repository is as follows:

```
/multimno_internal/
└── multimno
    └── components
        └── execution
            └── present_population
                └── present_population_estimation.py
```

present_population_estimation.py contains one class named PresentPopulationEstimation which is a subclass of Component.
The PresentPopulationEstimation class overwrites __init__, transform and execute in the Component class.
__init__ method initializes the data objects and reads the necessary values from the config file.
transform performs all necessary transformations and calculation of activity statistics for daily data. transform contains calls to other smaller functions that perform the actual data manipulation.
execute is responsible for calling read, write and transform for each unique date in the dataset. The processing is done one time point at a time.

5.2.21 MIDTERMPERMANENCESCORE

5.2.21.1 MODULE DESCRIPTION

- **Module Name:** MidtermPermanenceScore
- **Objectives:** Process Daily Permanence Score to obtain mid-term permanence score metrics, including frequency and regularity of stays, for different sub-monthly and sub-daily periods.
- **Functionality:** needed functionalities are outlined in the software requirement specifications:
  - 3.2.20 MidTermPermanenceEstimation
- **Data Inputs and Outputs:**
  - Input:
    - I.21 Daily Permanence Score
    - I.40 Holiday Dates Calendar
    - I.13 Cell Footprints
  - Output:
    - I.38 Mid-Term Permanence Metrics

5.2.21.2DEVELOPMENT DESIGN

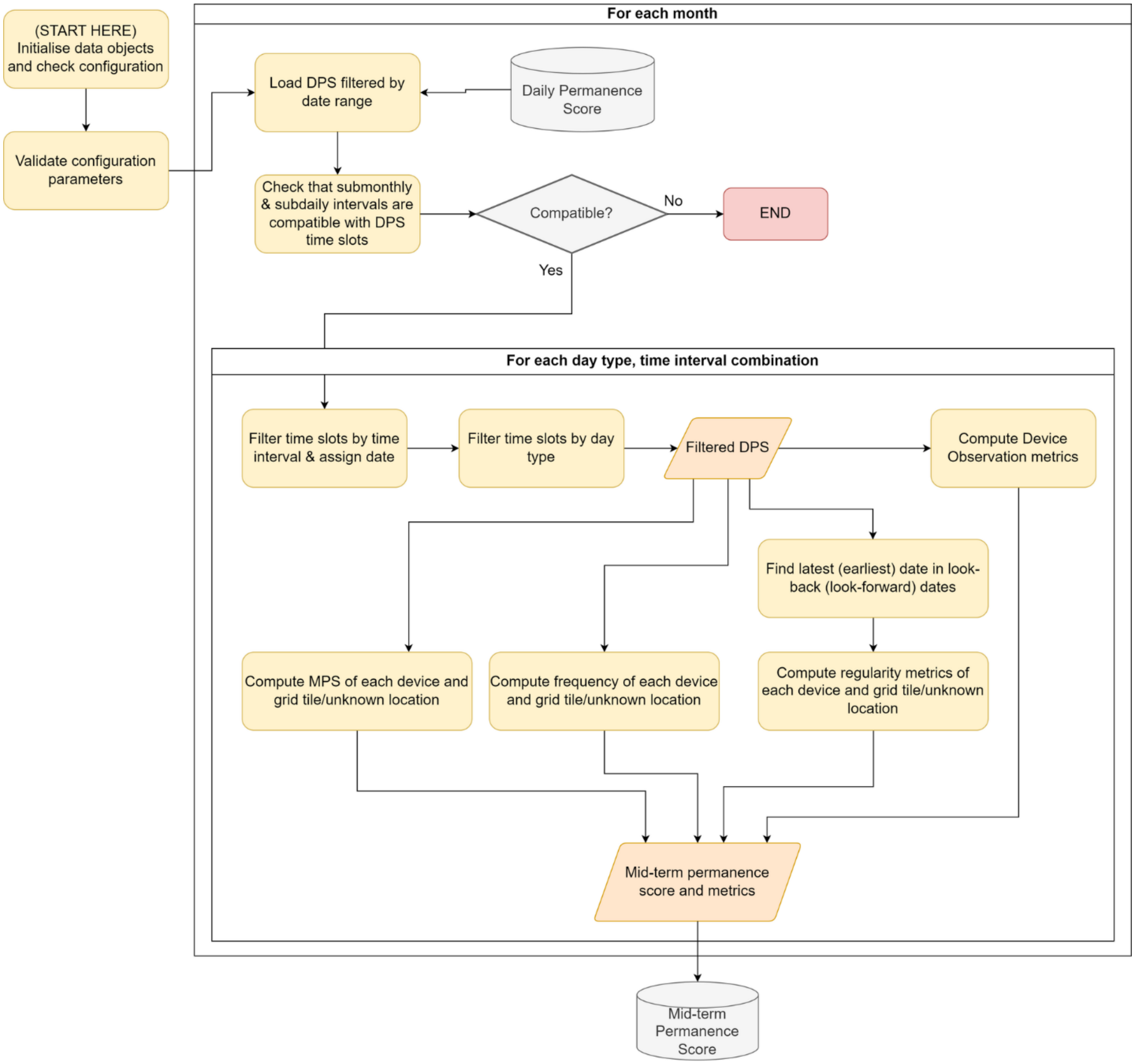
- **Key Algorithms/Processes:**
  - Initialise data objects.
  - Parse and validate configuration parameters:
    - Read months for which the mid-term metrics will be computed.
    - Read the number of days before and after each month that will be used to compute the regularity metrics.
    - Read the hour that marks the start of a day.
    - Read the definition of each time interval (i.e., subdaily period), and reject non-allowed time intervals

    - It is not allowed, when interval_end is not 00:00, that interval_end < interval_start < day_start
    - (Except for night_time) It is not allowed that interval_start < day_start < interval_end
  - Read start and end days of the week defining the weekend.
  - Read each combination (day_type, time_interval) for which the mid-term metrics are to be computed in each month.
- For each month to be studied:
  - Read Daily Permanence Score data necessary for the processing of this month.
  - Select only rows where DPS > 0 (i.e., DPS = 1).
  - Check that the duration of the time slots of each date of Daily Permanence Score data is compatible with the time intervals' start and end times (in particular, their minutes) by taking one row from each date. If they are not compatible, raise an error.
  - For each day type and time interval combination to be studied:
    - Filter out the time slots that do not belong to this time interval.
    - Assign the correct date that each time slot belongs to, according to the definition of a day following the hour that marks the start of each day: a time slot belongs to the date that contains its start time.
    - Filter out the time slots that do not belong to a date of the current day type:
      - In particular, work days are defined as those dates that are not part of the weekend and are not holidays.
    - Find, for each grid tile and device, the latest date in the regularity look-back dates with any time slot with DPS = 1, if it exists.
    - Find, for each grid tile and device, the earliest date in the regularity look-forward dates with any time slot with DPS = 1, if it exists.
    - For each device, calculate the number of time slots in this month, day type and time interval in which any grid tile had DPS = 1. The sum of these DPS values (equal to the count of these time slots) is equal to the "Device Observation" Mid-term Permanence Score. Similarly, the number of dates in which any grid tile in any time slot had DPS = 1 is equal to the "Device Observation" frequency. Store these values.
    - Compute the mid-term permanence score of a device and grid tile as the sum of the DPS values of its time slots in this month, day type and time interval.
    - For each device and grid tile (as well as unknown location) find the dates in this month, day type and time interval in which the DPS value of any time slot was equal to 1.
      - Compute the mid-term frequency of this device and location as the number of these dates.
      - Compute the day difference or gap between the consecutive dates of this list, considering the following: i) if there was a latest date in the regularity look-back dates, put it at the beginning of the ordered list, and if not, put the start date of the look-back period instead; ii) if there was an earliest date in the regularity look-forward dates, put it at the end of the ordered list, and if not, put the end date of the look-forward period instead.
      - Compute the regularity mean as the mean of these day distances.
      - Compute the regularity standard deviation as the sample standard deviation of these day distances:

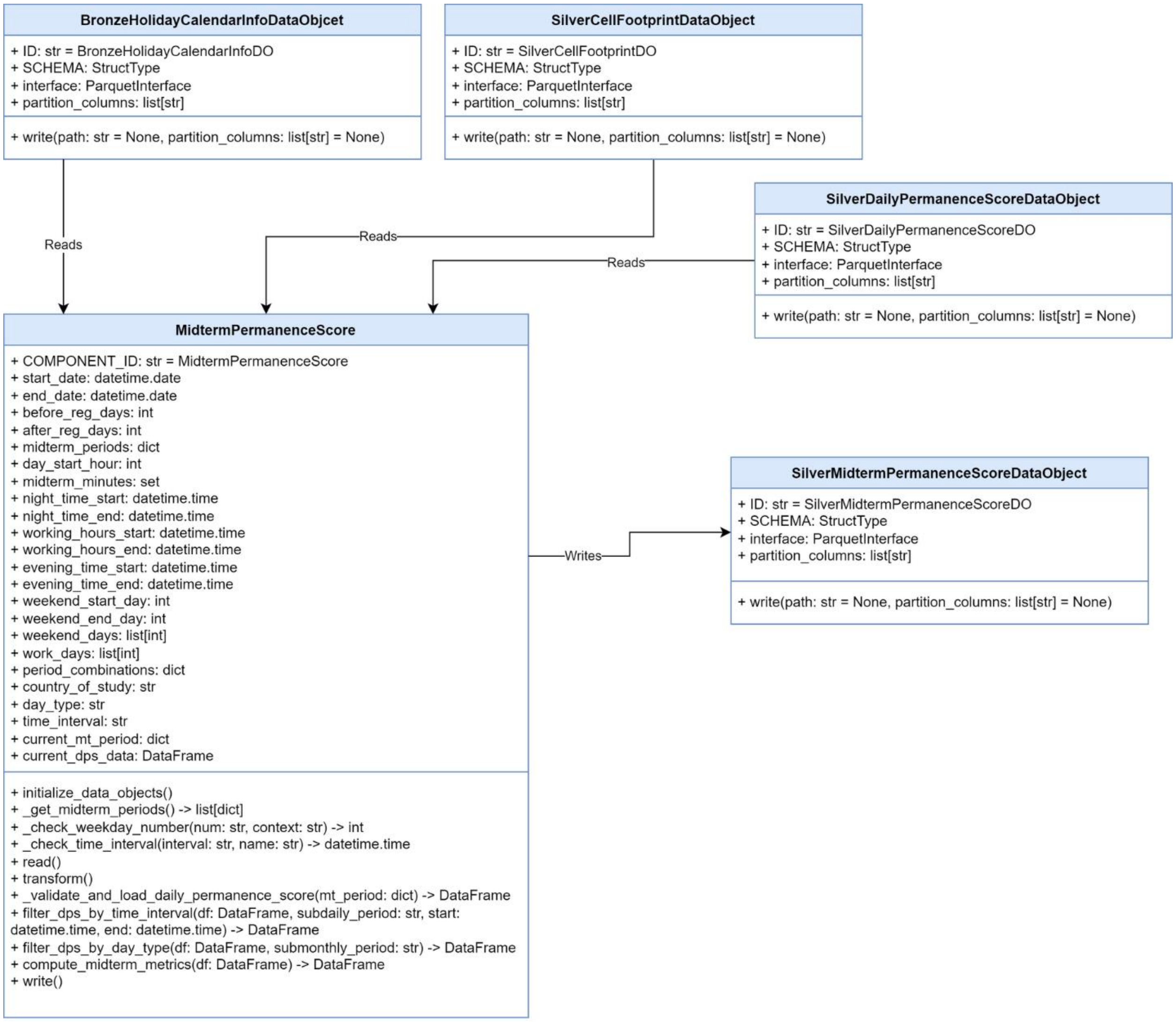
$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{n}(x_i - \bar{x})}$$

- Save the mid-term permanence score and metrics computed for this month and all combinations of day types and time intervals considered.

- **Data flow diagram:**


(START HERE) Initialise data objects and check configuration
Validate configuration parameters
For each month
Load DPS filtered by date range
Daily Permanence Score
Check that submonthly & subdaily intervals are compatible with DPS time slots
Compatible?
No
END
Yes
For each day type, time interval combination
Filter time slots by time interval & assign date
Filter time slots by day type
Filtered DPS
Compute Device Observation metrics
Find latest (earliest) date in look-back (look-forward) dates
Compute MPS of each device and grid tile/unknown location
Compute frequency of each device and grid tile/unknown location
Compute regularity metrics of each device and grid tile/unknown location
Mid-term permanence score and metrics
Mid-term Permanence Score

- **Class diagram:**


BronzeHolidayCalendarInfoDataObjcet
+ ID: str = BronzeHolidayCalendarInfoDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
SilverCellFootprintDataObject
+ ID: str = SilverCellFootprintDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
SilverDailyPermanenceScoreDataObject
+ ID: str = SilverDailyPermanenceScoreDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
Reads
Reads
Reads
MidtermPermanenceScore
+ COMPONENT_ID: str = MidtermPermanenceScore
+ start_date: datetime.date
+ end_date: datetime.date
+ before_reg_days: int
+ after_reg_days: int
+ midterm_periods: dict
+ day_start_hour: int
+ midterm_minutes: set
+ night_time_start: datetime.time
+ night_time_end: datetime.time
+ working_hours_start: datetime.time
+ working_hours_end: datetime.time
+ evening_time_start: datetime.time
+ evening_time_end: datetime.time
+ weekend_start_day: int
+ weekend_end_day: int
+ weekend_days: list[int]
+ work_days: list[int]
+ period_combinations: dict
+ country_of_study: str
+ day_type: str
+ time_interval: str
+ current_mt_period: dict
+ current_dps_data: DataFrame
+ initialize_data_objects()
+ _get_midterm_periods() -> list[dict]
+ _check_weekday_number(num: str, context: str) -> int
+ _check_time_interval(interval: str, name: str) -> datetime.time
+ read()
+ transform()
+ _validate_and_load_daily_permanence_score(mt_period: dict) -> DataFrame
+ filter_dps_by_time_interval(df: DataFrame, subdaily_period: str, start: datetime.time, end: datetime.time) -> DataFrame
+ filter_dps_by_day_type(df: DataFrame, submonthly_period: str) -> DataFrame
+ compute_midterm_metrics(df: DataFrame) -> DataFrame
+ write()
Writes
SilverMidtermPermanenceScoreDataObject
+ ID: str = SilverMidtermPermanenceScoreDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
/multimno_internal/
└── src
    └── components
        └── execution
            └── midterm_permanence_score
                └── midterm_permanence_score.py
```

  - midterm_permanence_score.py contains one class named MidtermPermanenceScorewhich is a subclass of Component. It also contains the function frequency_and_regularity, a PySpark UDF that computes the mid-term frequency and regularity metrics for each device and grid tile or unknown location.
  - The MidtermPermanenceScoreclass overrides some of the methods of Component
    - The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - The transform method handles all the logic behind the component.
  - The MidtermPermanenceScore also has the following methods:
    - _get_midterm_periods returns a list of dictionaries, where each dictionary contains the start and end date of the month of study of each mid-term period, together with the start and end date of the additional dates used for computing the regularity metrics.
    - _check_weekday_number parses and validates a numerical day of the week.
    - _check_time_interval parses and validates the start or end of a time interval/sub-daily period.
    - _validate_and_load_daily_permanence_score reads the Daily Permanence Data Object data that will be used for a particular mid-term period and validates that the time slot duration of every date to be used is compatible with the time intervals defined in the configuration file.
    - filter_dps_by_time_interval filters out the time slots that do not belong to a particular time interval, and adds the "date" column, assigning each time slot to its corresponding date according to the day start hour parameter.
    - filter_dps_by_day_type filters out the time slots that do not belong to a particular day type, based on the "date" column previously generated by the filter_dps_by_time_interval method.
    - compute_midterm_metrics computes the mid-term permanence score and metrics.

5.2.22 LONGTERMPERMANENCESCORE

5.2.22.1 MODULE DESCRIPTION

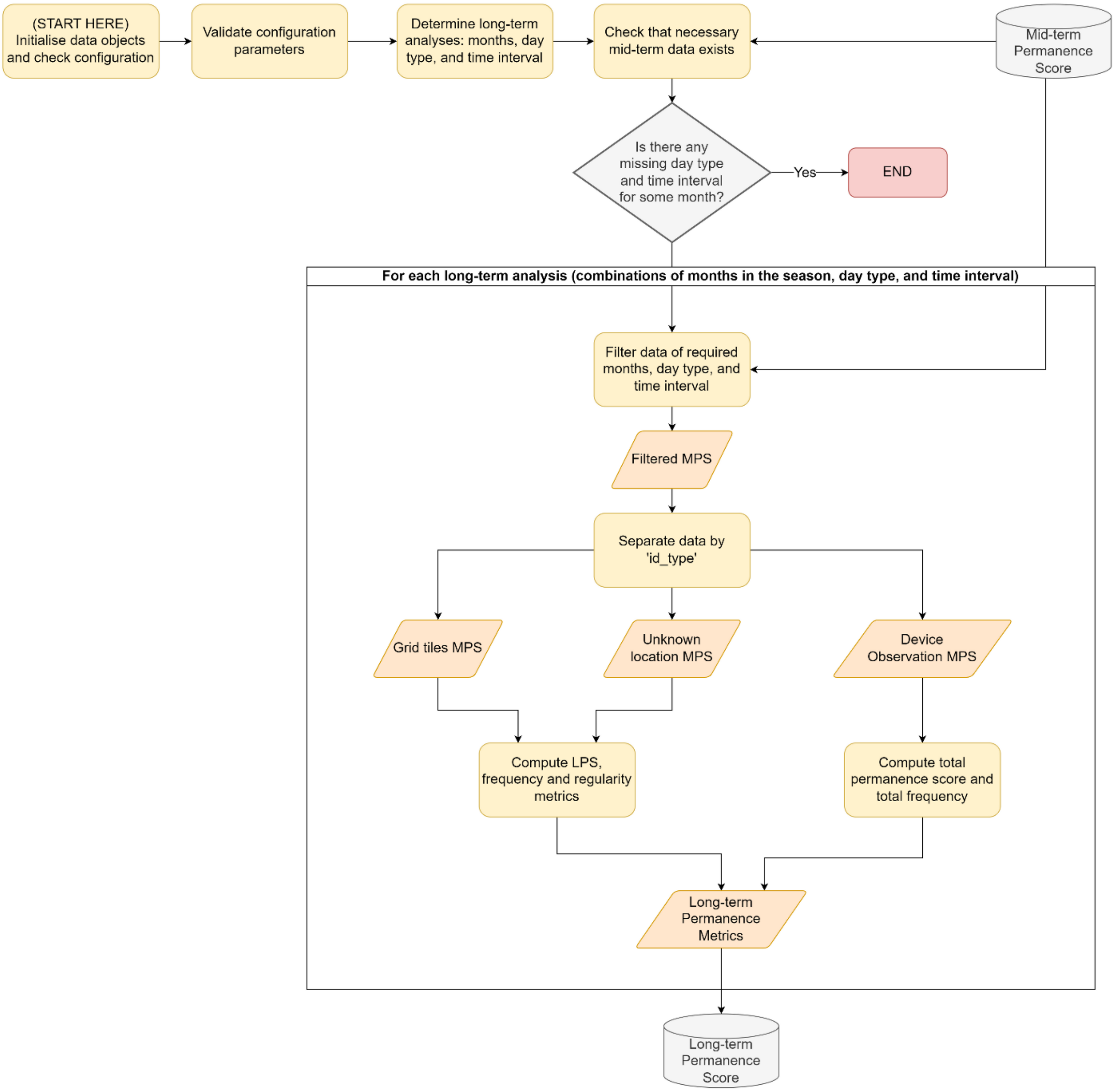
- **Module Name:** LongtermPermanenceScore
- **Objectives:** Process Mid-term Permanence Score to obtain long-term permanence score metrics, including total frequency, mean and standard deviation of mid-term frequency, and mean and standard deviation of the mid-term regularity mean metric, for different sub-yearly, sub-monthly, and sub-daily period combinations.
- **Functionality:** needed functionalities are outlined in the software requirement specifications:
  - 3.2.21 LongTermPermanenceEstimation

- **Data Inputs and Outputs:**
    - Input:
        - I.38 Mid-Term Permanence Metrics
    - Output:
        - I.39 Long-Term Permanence Metrics

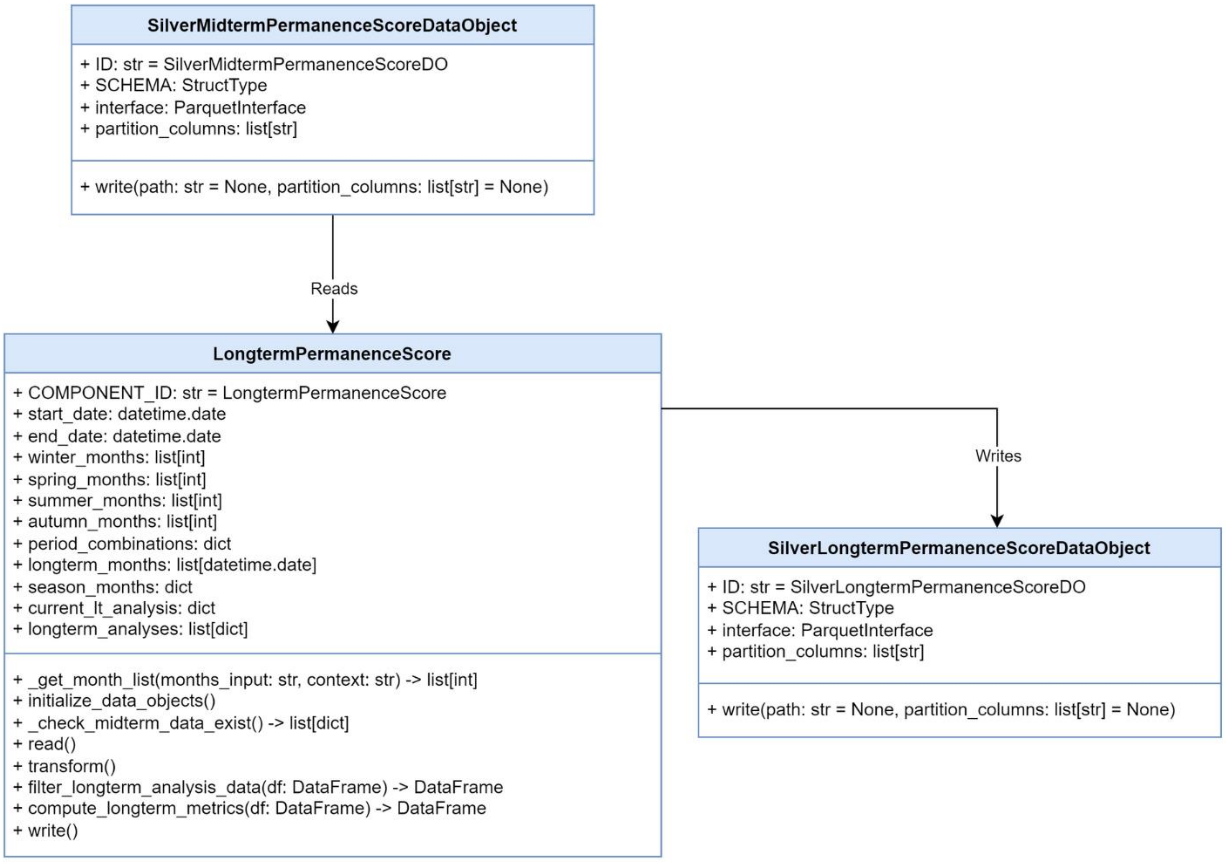
5.2.22.2DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**
    - Initialise data objects.
    - Parse and validate configuration parameters:
        - Read initial and final month that define the complete long-term analysis.
        - Read the months that are assigned to each of the four seasons: winter, spring, summer, and autumn, as integers from 1 to 12. They must not be repeated in the same season or appear in more than one season.
        - Read and validate all desired combinations of season, day type, and time interval (i.e., sub-yearly, sub-monthly, and sub-daily periods) for which the long-term permanence metrics will be computed separately.
        - Check that, if some season has been requested, it has been assigned via configuration at least one month. If not, raise an error.
        - For each season, day type, and time interval, determine the concrete set of months between the initial and final month (both inclusive) that belong to this combination.
    - For each season, day type, and time interval combination, check that for its assigned months there is mid-term permanence score data available. If not, warn the user and stop the component.
    - For each season, day type, and time interval combination:
        - Compute the long-term permanence score and total frequency of each device and grid tile / unknown location / device observation as the sum of the mid-term permanence score and mid-term frequency respectively.
        - Compute the mean and standard deviation of the mid-term frequency, as well as the mean and standard deviation of the mid-term regularity mean metric for each device and grid tile / unknown location.

- **Data flow diagram:**


(START HERE) Initialise data objects and check configuration
Validate configuration parameters
Determine long-term analyses: months, day type, and time interval
Check that necessary mid-term data exists
Mid-term Permanence Score
Is there any missing day type and time interval for some month?
Yes
END
For each long-term analysis (combinations of months in the season, day type, and time interval)
Filter data of required months, day type, and time interval
Filtered MPS
Separate data by 'id_type'
Grid tiles MPS
Unknown location MPS
Device Observation MPS
Compute LPS, frequency and regularity metrics
Compute total permanence score and total frequency
Long-term Permanence Metrics
Long-term Permanence Score

- **Class diagram:**


SilverMidtermPermanenceScoreDataObject
+ ID: str = SilverMidtermPermanenceScoreDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)
Reads
LongtermPermanenceScore
+ COMPONENT_ID: str = LongtermPermanenceScore
+ start_date: datetime.date
+ end_date: datetime.date
+ winter_months: list[int]
+ spring_months: list[int]
+ summer_months: list[int]
+ autumn_months: list[int]
+ period_combinations: dict
+ longterm_months: list[datetime.date]
+ season_months: dict
+ current_lt_analysis: dict
+ longterm_analyses: list[dict]
+ _get_month_list(months_input: str, context: str) -> list[int]
+ initialize_data_objects()
+ _check_midterm_data_exist() -> list[dict]
+ read()
+ transform()
+ filter_longterm_analysis_data(df: DataFrame) -> DataFrame
+ compute_longterm_metrics(df: DataFrame) -> DataFrame
+ write()
Writes
SilverLongtermPermanenceScoreDataObject
+ ID: str = SilverLongtermPermanenceScoreDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]
+ write(path: str = None, partition_columns: list[str] = None)

- **Code Structure:** The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
/multimno_internal/
└── src
    └── components
        └── execution
            └── longterm_permanence_score
                └── longterm_permanence_score.py
```

  - longterm_permanence_score.py contains one class named LongtermPermanenceScorewhich is a subclass of Component.
  - The LongtermPermanenceScoreclass overrides some of the methods of Component
    - The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.
    - The transform method handles all the logic behind the component.
  - The LongtermPermanenceScore also has the following methods:
    - _get_month_list parses and validates a list of integers representing the months that will comprise a season and returns them as a list of integers between 1 and 12.
    - _check_midterm_data_exist checks that, for each combination of season, day type and time interval, the mid-term permanence score of the months that belong to that combination has been computed for those months, day type, and time interval. If there is some data missing, the component stops and warns the user of the missing data. If the check passes, the function returns a list of dictionaries, each containing the season, day type, time interval, and list of months that form an individual long-term analysis.
    - filter_longterm_analysis_data filters the mid-term permanence score data that is going to be used for the current long-term analysis being performed, by selecting only the necessary months, day types, and time intervals.
    - compute_longterm_metrics calculates the long-term permanence score and metrics for the current long-term analysis being performed.

5.2.23 USUALENVIRONMENTLABELING

5.2.23.1 MODULE DESCRIPTION

- **Module Name:** UsualEnvironmentLabeling
- **Objectives:** The objective of this module is to get measures on a large time scale (e.g. 6 months, 1 year) at the device level. For each device, this module aims to get as output a proxy its Usual Environment, and a tentative identification of Home Location, Second Home (not to be implemented yet) and Work/Study place.
- **Functionality:** needed functionalities are outlined in the software requirement specifications:
  - 3.2.22 UsualEnvironmentLabeling
- **Data Inputs and Outputs:**
  - Input:
    - I.39 Long-Term Permanence Metrics
  - Output:
    - I.37 UE Labels

- - - I.43 Labeling Quality Metrics

5.2.23.2DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**

Initialisation:

1. Read configuration parameters for component processing, date range for which to read dataset (start month and end month), and thresholds to consider.
2. **Clear the destination directory if configured.**
3. Load input data object for Long-term Permanence Metrics.
4. Initialize the output data objects for UE Labels and Labeling Quality Metrics.

Processing:

1. Get specified start date and end date:
   - Find the start_date of the specified period by selecting the first day of the start_month
   - Find the end_date of the specified period by selecting the last day of the end_month
2. Filter Long-term Permanence Metrics dataset to required time range by filtering ‘start_date' = start_date and 'end_date’ = end_date. Set this filtered Long-term Permanence Metrics dataset as the new Long-term Permanence Metrics dataset to be used from here onwards.
3. Check that all of the following combinations of values for the columns ‘day_type' and 'time_interval’ are present (at least once) in the Long-term Permanence Metrics dataset:
   - ‘all_days’, ‘all_intervals’
   - ‘all_days’, ‘night-time’
   - 'work_days', ‘working_time’
4. If some of these combinations of values for the columns ‘day_type' and 'time_interval’ are not present in the input data object, exit method with an error message.
5. Detect ‘rarely observed devices’ and ‘discountinously observed devices’:
   - Filter Long-term Permanence Metrics dataset by ‘id_type' = ‘device_observation’, ‘day_type’ = ‘all_days’ and ‘time_interval’ = 'all_intervals’ to obtain Device Observation Metrics dataset.
   - Now, for ‘rarely observed devices’:
     - Filter rows of Device Observation Metrics dataset for which ‘lps’ < total_ps_threshold
     - Find the list of unique values of ‘user_id’ column from the resulting filtered Device Observation Metrics dataset.
     - Count the length of this list of unique user values, and save to device_filter_1_rule integer variable
   - Now, for ‘discountinously observed devices’:
     - Filter rows of Device Observation Metrics dataset for which ‘total_frequency’ < freq_days_threshold
     - Find the list of unique values of ‘user_id’ column from the resulting filtered Device Observation Metrics dataset.
     - Count the length of this list of unique user values, and save to device_filter_2_rule integer variable
6. Discard from the Long-term Permanence Metrics all those rows with a ‘user_id’ value that is included either in the ‘rarely observed devices’ list or in the ‘discountinously observed devices’ list. Set this filtered Long-term Permanence Metrics dataset as the new Long-term Permanence Metrics dataset to be used from here onwards.
7. Use Usual Environment Labeling function, explained in the corresponding section*, to produce the Usual Environment Tiles Dataset:
   - Usual Environment Tiles Dataset = Usual Environment Labeling (Long-term Permanence Metrics dataset, ue_gap_ps_threshold, ue_ps_threshold, ue_ndays_threshold)

8. Use Home Labeling function, explained in the corresponding section*, to produce the Home Tiles Dataset:
    - Home Tiles Dataset = Home Labeling (Long-term Permanence Metrics dataset, gap_ps_threshold, home_ps_threshold, home_ndays_threshold)
9. Use Work Labeling function, explained in the corresponding section*, to produce the Work Tiles Dataset:
    - Work Tiles Dataset = Work Labeling (Long-term Permanence Metrics dataset, gap_ps_threshold, work_ps_threshold, work_ndays_threshold)
10. Join Usual Environment Tiles, Home Tiles and Work Tiles datasets into a UE Labels dataset:
    - A) Create the UE Labels dataset as a copy of the Usual Environment Tiles.
    - B) Add the Home Tiles to the UE Labels dataset:
        - B.1.) For rows in which ‘grid_id’ value is already in the UE Labels dataset:
            - replace ‘label' value of these rows (currently ‘no_label’) by 'home’.
            - add ‘location_label_rule’ value from Home Tiles dataset.
        - B.2.) For rows in which ‘grid_id’ value is not in the UE Labels dataset:
            - directly add row from Home Tiles to the UE Labels dataset.
            - set ‘ue_label_rule’ as ‘ue_na’ for this row.
    - C) Add the Work Tiles to the UE Labels dataset:
        - C.1.) For rows in which ‘grid_id’ value is already in the UE Labels dataset:
            - C.1.1.) if the value of this row in the UE Labels dataset, for column ‘label', is 'no_label’:
                - replace ‘label' value of these rows (currently ‘no_label’) by 'work’.
                - add ‘location_label_rule’ value from Work Tiles dataset.
            - C.1.2.) if the value of this row, in the UE Labels dataset, for column ‘label', is not ‘no_label’ (thus, it is 'home’), then keep this row in the UE Labels and add a new one:
                - directly add row from Work Tiles to the UE Labels dataset.
                - set ‘ue_label_rule’ as the ‘ue_label_rule’ of the old row for this new row.
        - C.2.) For rows in which ‘grid_id’ value is not in the UE Labels dataset:
            - directly add row from Work Tiles to the UE Labels dataset.
            - set ‘ue_label_rule’ as ‘ue_na’ for this row.
11. Write UE Labels dataset to output as a parquet UE Labels Data Object with the corresponding specifications.
12. Generate Labeling Quality Metrics dataset by using previously calculated device_filter_1_rule and device_filter_2_rule counts, and by counting the number of occurrences in the UE Labels dataset for each of the specified label/rule_label combinations:
    - ‘ue_1_rule’: number of rows in UE Labels dataset in which ‘ue_label_rule’ == ‘ue_1’.
    - ‘ue_2_rule’: number of rows in UE Labels dataset in which ‘ue_label_rule’ == ‘ue_2’.
    - ‘h_1_rule’: number of rows in UE Labels dataset in which ‘location_label_rule’ == ‘h_1’.
    - ‘h_2_rule’: number of rows in UE Labels dataset in which ‘location_label_rule’ == ‘h_2’.
    - ‘h_3_rule’: number of rows in UE Labels dataset in which ‘location_label_rule’ == ‘h_3’.
    - ‘w_1_rule’: number of rows in UE Labels dataset in which ‘location_label_rule’ == ‘w_1’.
    - ‘w_2_rule’: number of rows in UE Labels dataset in which ‘location_label_rule’ == ‘w_2’.
    - ‘ue_na_rule’: number of rows in UE Labels dataset in which ‘ue_label_rule’ == ‘loc_na’.
    - ‘loc_na_rule’: number of rows in UE Labels dataset in which ‘location_label_rule’ == ‘loc_na’.
    - ‘h_non_ue’: number of rows in UE Labels dataset in which ‘ue_label_rule’ == ‘loc_na’ & ‘label’ == ‘home’.
    - ‘w_non_ue’: number of rows in UE Labels dataset in which ‘ue_label_rule’ == ‘loc_na’ & ‘label’ == ‘work’.

\ *USUAL ENVIRONMENT LABELING PROCESS:

- Arguments:
    - Long-term Permanence Metrics dataset
    - ue_gap_ps_threshold
    - ue_ps_threshold
    - ue_ndays_threshold (currently not used)
- Returns:
    - UE tiles

1. Filter Long-term Permanence Metrics dataset: keep rows in which ‘day_type’ = ‘all_days’ and ‘time_interval’ = ‘all_intervals' to obtain Long-term Permanence Metrics dataset (1):
    1. Apply Preprocessing function to obtain Pre-selected tiles (1):
        - Pre-selected tiles (1) = **Preprocessing** (Long-term Permanence Metrics dataset (1), ue_gap_ps_threshold, threshold_is_absolute=False)
    2. Calculate total assigned PS: Filter Long-term Permanence Metrics dataset (1) by id_type = ‘device_observation’, then reach ‘lps’ value of the only row in the resulting dataframe. This is tot_assigned_ps.
    3. Apply relative LPS filter:
        - abs_ps_threshold = tot_assigned_ps * ue_ps_threshold / 100
        - Filter rows of Pre-selected tiles (1) for which ‘lps’ > abs_ps_threshold to obtain:
            - > Selected tiles (1)
            - < Not selected tiles (1)
    4. Generate UE tiles (1): copy Selected Tiles (1) and add column ‘ue_label_rule’ = ‘ue_1’
2. Filter Long-term Permanence Metrics dataset: keep rows in which ‘day_type’ = ‘all_days’ and ‘time_interval’ = ‘night-time' to obtain Long-term Permanence Metrics dataset (2):
    1. Pre-selected tiles (2) = Long-term Permanence Metrics dataset (2), keeping only those rows with grid_id in Not selected tiles (1).
    2. Calculate total assigned PS: Filter Long-term Permanence Metrics dataset (2) by id_type = ‘device_observation’, then reach ‘lps’ value of the only row in the resulting dataframe. This is tot_assigned_ps.
    3. Apply relative LPS filter:
        - abs_ps_threshold = tot_assigned_ps * ue_ps_threshold / 100
        - Filter rows of Pre-selected tiles (2) for which ‘lps’ > abs_ps_threshold to obtain:
            - > Selected tiles (2)
            - < Not selected tiles (2)
    4. Generate UE tiles (2): copy Selected Tiles (2) and add column ‘ue_label_rule’ = ‘ue_2’
3. Filter Long-term Permanence Metrics dataset: keep rows in which ‘day_type’ = ‘work_days’ and ‘time_interval’ = ‘working_hours' to obtain Long-term Permanence Metrics dataset (3):
    1. Pre-selected tiles (3) = Long-term Permanence Metrics dataset (3), keeping only those rows with grid_id in Not selected tiles (2).
    2. Calculate total assigned PS: Filter Long-term Permanence Metrics dataset (3) by id_type = ‘device_observation’, then reach ‘lps’ value of the only row in the resulting dataframe. This is tot_assigned_ps.
    3. Apply relative LPS filter:
        - abs_ps_threshold = tot_assigned_ps * ue_ps_threshold / 100
        - Filter rows of Pre-selected tiles (3) for which ‘lps’ > abs_ps_threshold to obtain:
            - > Selected tiles (3)
            - < Not selected tiles (3)
    4. Generate UE tiles (3): copy Selected Tiles (3) and add column ‘ue_label_rule’ = ‘ue_2’
    5. Generate UE tiles (na): copy Not selected tiles (3) and add column ‘ue_label_rule’ = ‘ue_na’

4. Concatenate UE tiles (1), UE tiles (2), UE tiles (3) and UE tiles (na) tables, and add column 'label' = 'no_label' to generate UE tiles.

\ *HOME LABELING PROCESS:

- Arguments:
    - Long-term Permanence Metrics dataset
    - gap_ps_threshold
    - home_ps_threshold
    - home_ndays_threshold
- Returns:
    - Home tiles

1. Filter Long-term Permanence Metrics dataset: keep rows in which 'day_type' = 'all_days' and 'time_interval' = 'all_intervals' to obtain Long-term Permanence Metrics dataset (1):
    1. Apply Preprocessing function to obtain Pre-selected tiles (1):
        - Pre-selected tiles (1) = **Preprocessing** (Long-term Permanence Metrics dataset (1), gap_ps_threshold, threshold_is_absolute=True)
    2. Calculate total assigned PS: Filter Long-term Permanence Metrics dataset (1) by id_type = 'device_observation', then reach 'lps' value of the only row in the resulting dataframe. This is tot_assigned_ps.
    3. Apply relative LPS filter:
        - abs_ps_threshold = tot_assigned_ps * home_ps_threshold / 100
        - Filter rows of Pre-selected tiles (1) for which 'lps' > abs_ps_threshold to obtain:
            - > Selected tiles (1)
            - < Not selected tiles (1)
    4. Generate Home tiles (1): copy Selected Tiles (1) and add column 'location_label_rule' = 'h_1'
2. Filter Long-term Permanence Metrics dataset: keep rows in which 'day_type' = 'all_days' and 'time_interval' = 'night-time' to obtain Long-term Permanence Metrics dataset (2):
    1. Pre-selected tiles (2) = Long-term Permanence Metrics dataset (2), keeping only those rows with grid_id in Not selected tiles (1).
    2. Calculate total assigned PS: Filter Long-term Permanence Metrics dataset (2) by id_type = 'device_observation', then reach 'lps' value of the only row in the resulting dataframe. This is tot_assigned_ps.
    3. Apply relative LPS filter:
        - abs_ps_threshold = tot_assigned_ps * home_ps_threshold / 100
        - Filter rows of Pre-selected tiles (2) for which 'lps' > abs_ps_threshold to obtain:
            - > Selected tiles (2)
            - < Not selected tiles (2)
    4. Generate Home tiles (2): copy Selected Tiles (2) and add column 'location_label_rule' = 'h_2'
3. Filter Long-term Permanence Metrics dataset: keep rows in which 'day_type' = 'all_days' and 'time_interval' = 'all_intervals' to obtain Long-term Permanence Metrics dataset (3):
    1. Pre-selected tiles (3) = Long-term Permanence Metrics dataset (3), keeping only those rows with grid_id in Not selected tiles (2).
    2. Calculate total observed days: Filter Long-term Permanence Metrics dataset (3) by id_type = 'device_observation', then reach 'total_frequency' value of the only row in the resulting dataframe. This is tot_observed_days.
    3. Apply relative n days filter:
        - abs_ndays_threshold = tot_observed_days * home_ndays_threshold / 100
        - Filter rows of Pre-selected tiles (3) for which 'total_frequency' > abs_ndays_threshold to obtain:
            - > Selected tiles (3)

            - < Not selected tiles (3)
        4. Generate Home tiles (3): copy Selected Tiles (3) and add column ‘location_label_rule’ = ‘h_3’
        5. Generate Home tiles (na): copy Not selected tiles (3) and add column ‘location_label_rule’ = ‘loc_na’
4. Concatenate Home tiles (1), Home tiles (2), Home tiles (3) and Home tiles (na) tables, and add column ‘label’ = ‘home’ to generate Home tiles.

\ *WORK LABELING PROCESS:

- Arguments:
    - Long-term Permanence Metrics dataset
    - gap_ps_threshold
    - work_ps_threshold
    - work_ndays_threshold
- Returns:
    - Work tiles

1. Filter Long-term Permanence Metrics dataset: keep rows in which ‘day_type’ = ‘work_days’ and ‘time_interval’ = ‘working_hours' to obtain Long-term Permanence Metrics dataset (1):
    1. Apply Preprocessing function to obtain Pre-selected tiles (1):
        - Pre-selected tiles (1) = **Preprocessing** (Long-term Permanence Metrics dataset (1), gap_ps_threshold, threshold_is_absolute=True)
    2. Calculate total assigned PS: Filter Long-term Permanence Metrics dataset (1) by id_type = ‘device_observation’, then reach ‘lps’ value of the only row in the resulting dataframe. This is tot_assigned_ps.
    3. Apply relative LPS filter:
        - abs_ps_threshold = tot_assigned_ps * work_ps_threshold / 100
        - Filter rows of Pre-selected tiles (1) for which ‘lps’ > abs_ps_threshold to obtain:
            - > Selected tiles (1)
            - < Not selected tiles (1)
    4. Generate Work tiles (1): copy Selected Tiles (1) and add column ‘location_label_rule’ = ‘w_1’
2. Filter Long-term Permanence Metrics dataset: keep rows in which ‘day_type’ = ‘work_days’ and ‘time_interval’ = ‘working_hours' to obtain Long-term Permanence Metrics dataset (2):
    1. Pre-selected tiles (2) = Long-term Permanence Metrics dataset (2), keeping only those rows with grid_id in Not selected tiles (1).
    2. Calculate total observed days: Filter Long-term Permanence Metrics dataset (2) by id_type = ‘device_observation’, then reach ‘total_frequency’ value of the only row in the resulting dataframe. This is tot_observed_days.
    3. Apply relative n days filter:
        - abs_ndays_threshold = tot_observed_days * work_ndays_threshold / 100
        - Filter rows of Pre-selected tiles (2) for which ‘total_frequency’ > abs_ndays_threshold to obtain:
            - > Selected tiles (2)
            - < Not selected tiles (2)
    4. Generate Work tiles (2): copy Selected Tiles (3) and add column ‘location_label_rule’ = ‘w_2’
    5. Generate Work tiles (na): copy Not selected tiles (3) and add column ‘location_label_rule’ = ‘loc_na’
3. Concatenate Work tiles (1), Work tiles (2), Work tiles (3) and Work tiles (na) tables, and add column ‘label’ = ‘work’ to generate Work tiles.

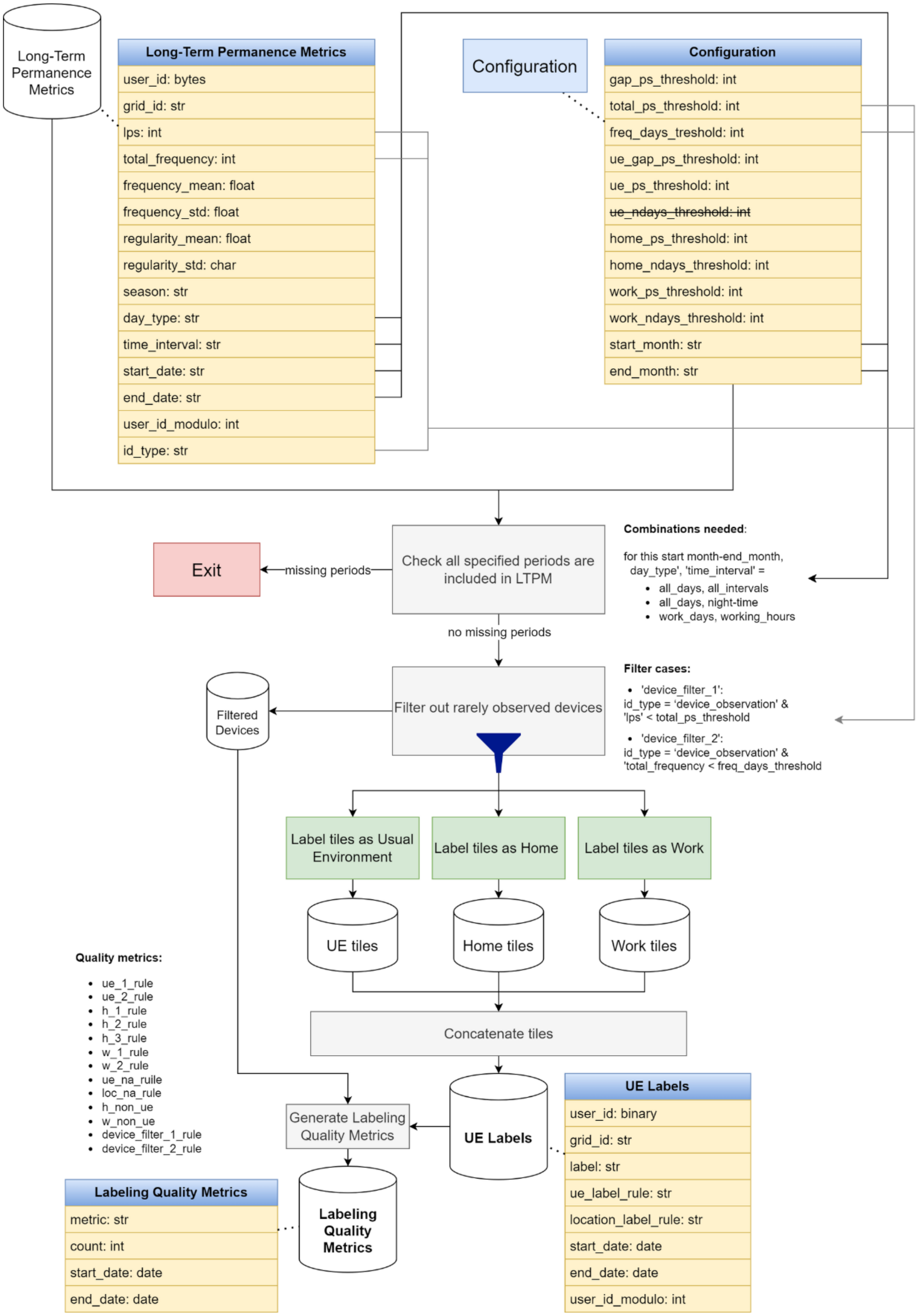
\ PREPROCESSING FUNCTION (CUT TILES AT GAP):

- Arguments:
  - Long-term Permanence Metrics dataset (filtered for a specific day/time)
  - gap_ps_threshold
  - threshold_is_absolute (bool)
- Returns:
  - Pre-selected tiles

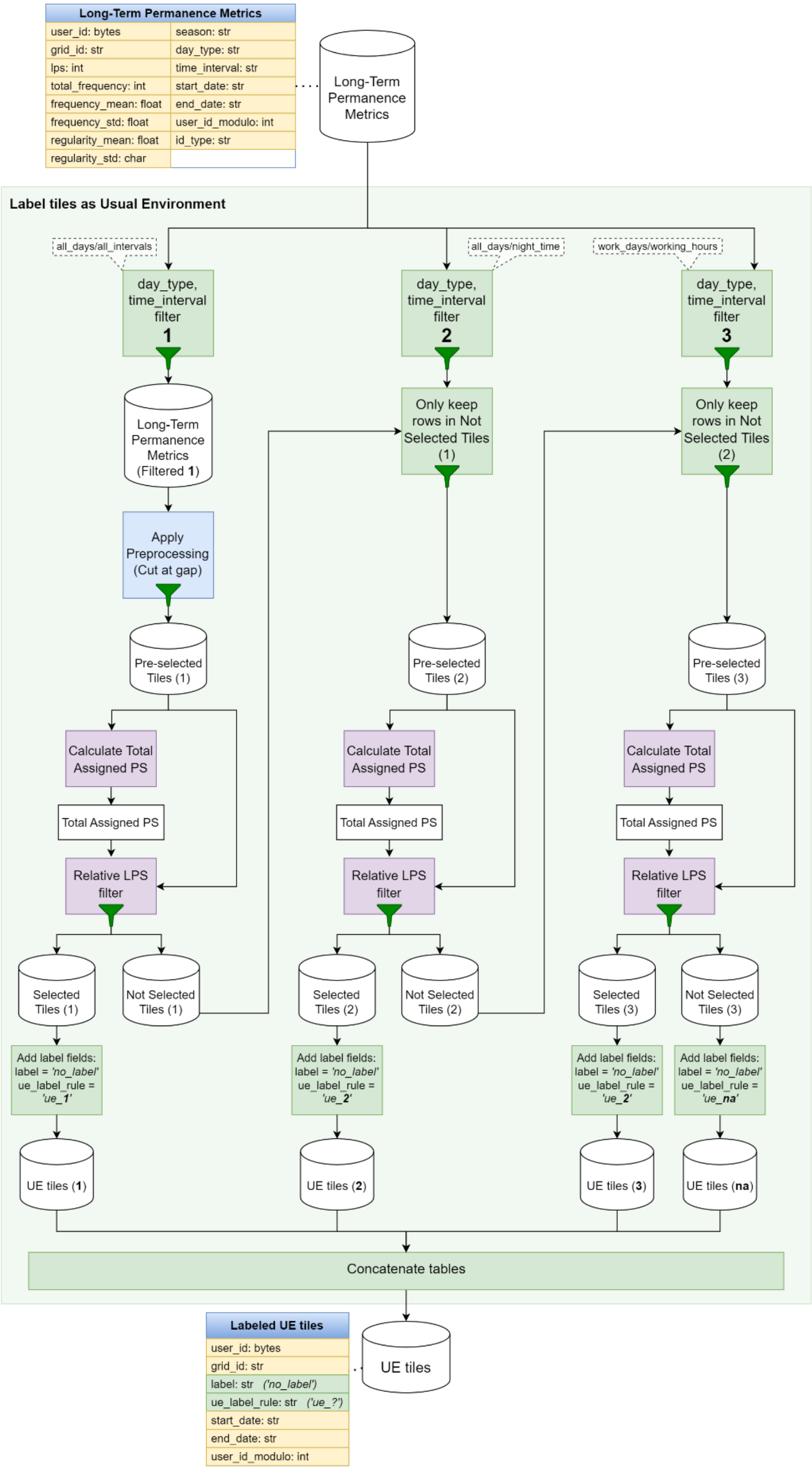
1. Filter Long-term Permanence Metrics dataset: keep rows in which ‘id_type’ = ‘grid’.
2. Sort resulting table by ‘lps’ field (descending).
3. Add ‘lps_difference’ column to resulting table:
   1. lps_difference = lps(current_row) - lps(previous_row)
4. Find first occurrence of ‘lps_difference’ >= gap_ps_threshold, then filter out all rows below.
5. The resulting dataframe are the Pre-selected tiles.

\ DATA FLOW DIAGRAMS:

- **KEY ALGORITHMS/PROCESSES (GENERAL VIEW):**



- **USUAL ENVIROMENT LABELING DETAIL:**

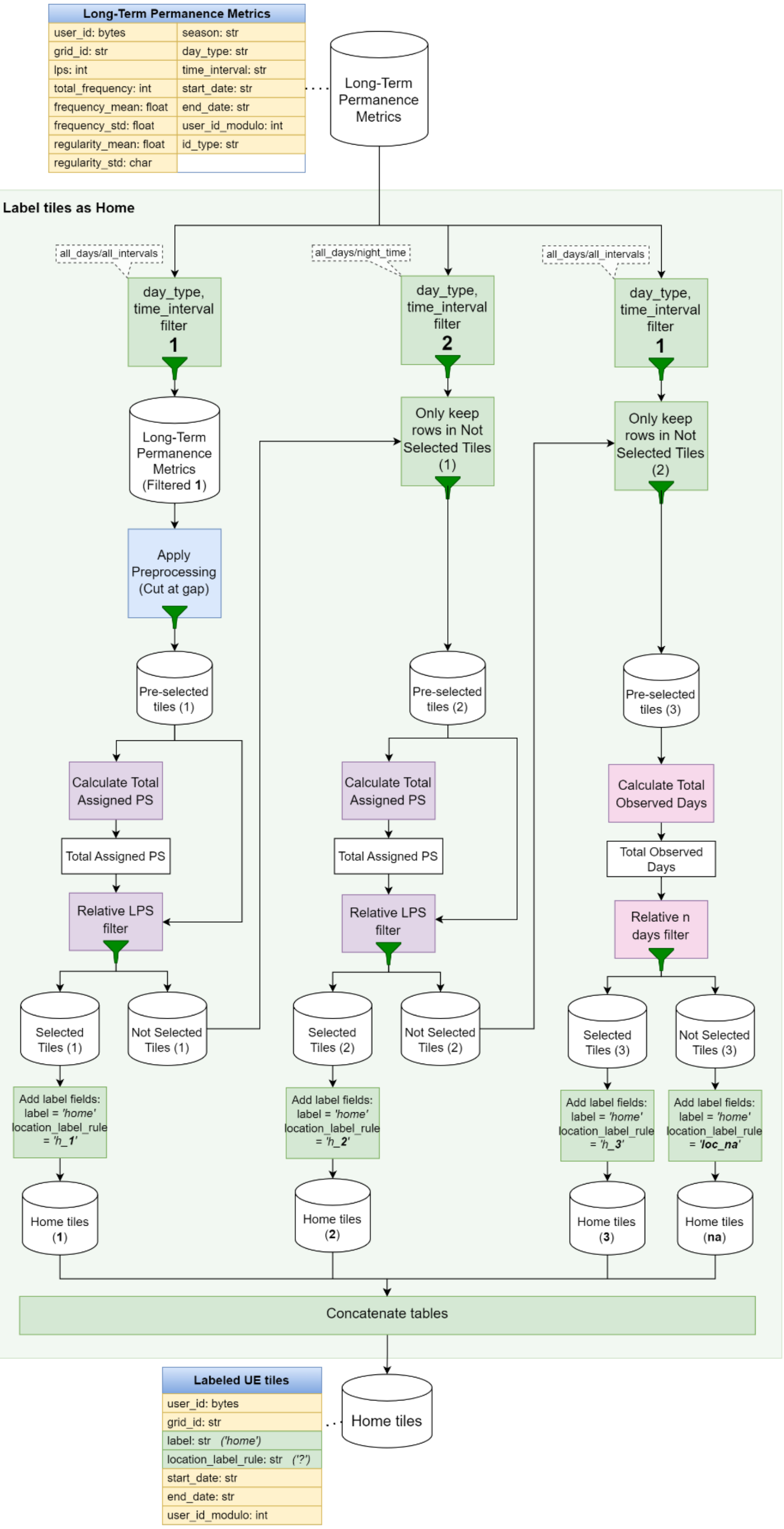


Long-Term Permanence Metrics	
user_id: bytes	season: str
grid_id: str	day_type: str
lps: int	time_interval: str
total_frequency: int	start_date: str
frequency_mean: float	end_date: str
frequency_std: float	user_id_modulo: int
regularity_mean: float	id_type: str
regularity_std: char	

Long-Term Permanence Metrics
Label tiles as Usual Environment
all_days/all_intervals
all_days/night_time
work_days/working_hours
day_type, time_interval filter 1
day_type, time_interval filter 2
day_type, time_interval filter 3
Long-Term Permanence Metrics (Filtered 1)
Only keep rows in Not Selected Tiles (1)
Only keep rows in Not Selected Tiles (2)
Apply Preprocessing (Cut at gap)
Pre-selected Tiles (1)
Pre-selected Tiles (2)
Pre-selected Tiles (3)
Calculate Total Assigned PS
Calculate Total Assigned PS
Calculate Total Assigned PS
Total Assigned PS
Total Assigned PS
Total Assigned PS
Relative LPS filter
Relative LPS filter
Relative LPS filter
Selected Tiles (1)
Not Selected Tiles (1)
Selected Tiles (2)
Not Selected Tiles (2)
Selected Tiles (3)
Not Selected Tiles (3)
Add label fields: label = 'no_label' ue_label_rule = 'ue_1'
Add label fields: label = 'no_label' ue_label_rule = 'ue_2'
Add label fields: label = 'no_label' ue_label_rule = 'ue_2'
Add label fields: label = 'no_label' ue_label_rule = 'ue_na'
UE tiles (1)
UE tiles (2)
UE tiles (3)
UE tiles (na)
Concatenate tables
UE tiles

Labeled UE tiles
user_id: bytes
grid_id: str
label: str ('no_label')
ue_label_rule: str ('ue_?')
start_date: str
end_date: str
user_id_modulo: int

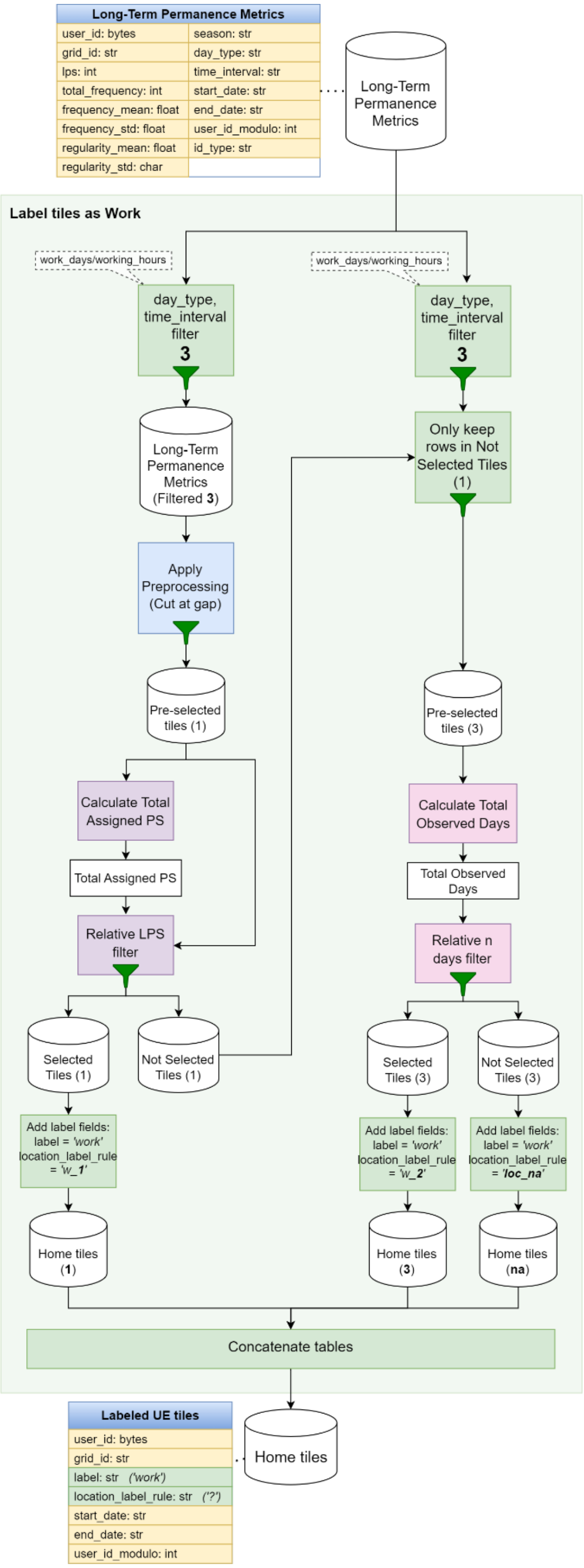
- **HOME LABELING DETAIL:**


Long-Term Permanence Metrics
Label tiles as Home
all_days/all_intervals
all_days/night_time
all_days/all_intervals
day_type, time_interval filter 1
day_type, time_interval filter 2
day_type, time_interval filter 1
Long-Term Permanence Metrics (Filtered 1)
Only keep rows in Not Selected Tiles (1)
Only keep rows in Not Selected Tiles (2)
Apply Preprocessing (Cut at gap)
Pre-selected tiles (1)
Pre-selected tiles (2)
Pre-selected tiles (3)
Calculate Total Assigned PS
Calculate Total Assigned PS
Calculate Total Observed Days
Total Assigned PS
Total Assigned PS
Total Observed Days
Relative LPS filter
Relative LPS filter
Relative n days filter
Selected Tiles (1)
Not Selected Tiles (1)
Selected Tiles (2)
Not Selected Tiles (2)
Selected Tiles (3)
Not Selected Tiles (3)
Add label fields: label = 'home' location_label_rule = 'h_1'
Add label fields: label = 'home' location_label_rule = 'h_2'
Add label fields: label = 'home' location_label_rule = 'h_3'
Add label fields: label = 'home' location_label_rule = 'loc_na'
Home tiles (1)
Home tiles (2)
Home tiles (3)
Home tiles (na)
Concatenate tables
Home tiles

Long-Term Permanence Metrics	
user_id: bytes	season: str
grid_id: str	day_type: str
lps: int	time_interval: str
total_frequency: int	start_date: str
frequency_mean: float	end_date: str
frequency_std: float	user_id_modulo: int
regularity_mean: float	id_type: str
regularity_std: char	

Labeled UE tiles
user_id: bytes
grid_id: str
label: str *('home')*
location_label_rule: str *('?')*
start_date: str
end_date: str
user_id_modulo: int

- **WORK LABELING DETAIL:**



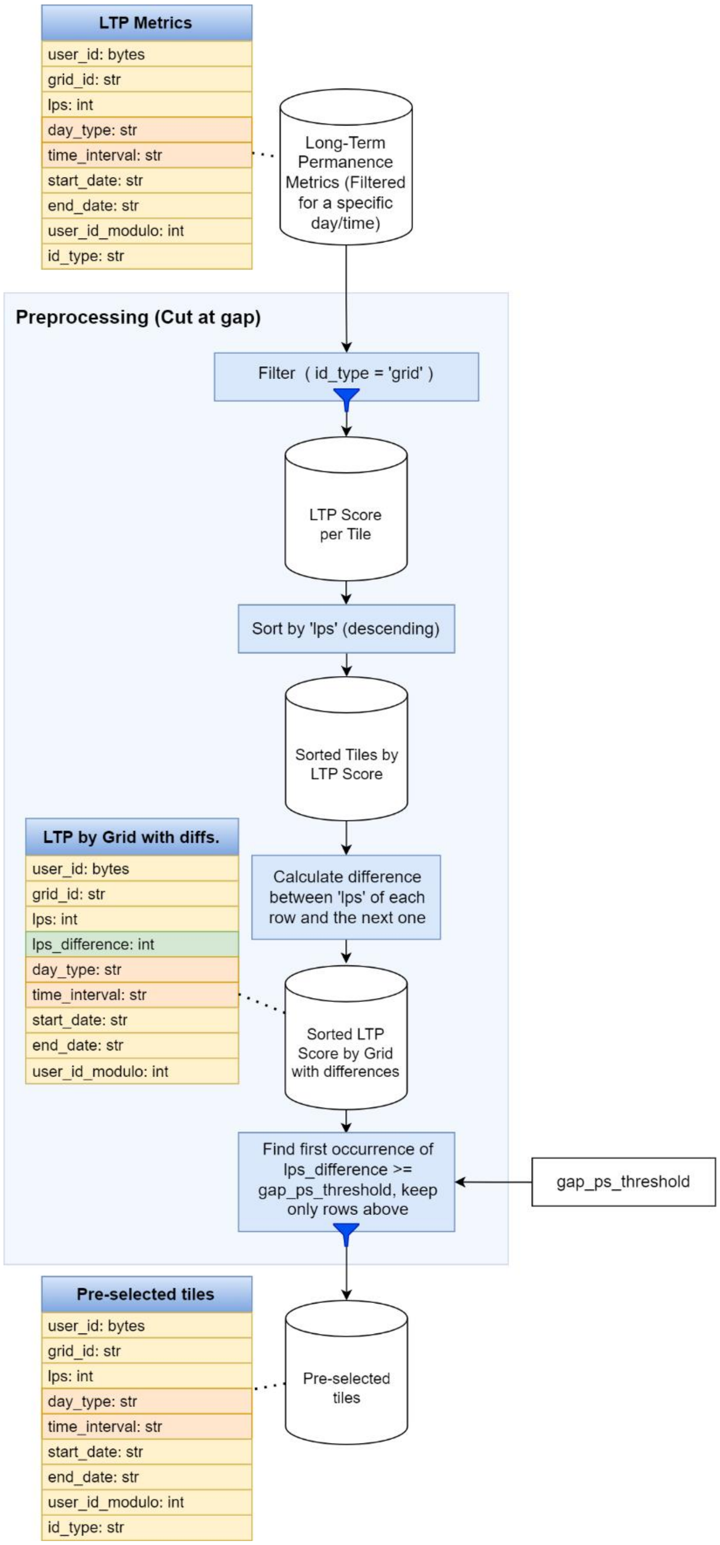
Long-Term Permanence Metrics	
user_id: bytes	season: str
grid_id: str	day_type: str
lps: int	time_interval: str
total_frequency: int	start_date: str
frequency_mean: float	end_date: str
frequency_std: float	user_id_modulo: int
regularity_mean: float	id_type: str
regularity_std: char	

Long-Term Permanence Metrics
Label tiles as Work
work_days/working_hours
day_type, time_interval filter 3
work_days/working_hours
day_type, time_interval filter 3
Long-Term Permanence Metrics (Filtered 3)
Only keep rows in Not Selected Tiles (1)
Apply Preprocessing (Cut at gap)
Pre-selected tiles (1)
Pre-selected tiles (3)
Calculate Total Assigned PS
Calculate Total Observed Days
Total Assigned PS
Total Observed Days
Relative LPS filter
Relative n days filter
Selected Tiles (1)
Not Selected Tiles (1)
Selected Tiles (3)
Not Selected Tiles (3)
Add label fields: label = 'work' location_label_rule = 'w_1'
Add label fields: label = 'work' location_label_rule = 'w_2'
Add label fields: label = 'work' location_label_rule = 'loc_na'
Home tiles (1)
Home tiles (3)
Home tiles (na)
Concatenate tables
Home tiles

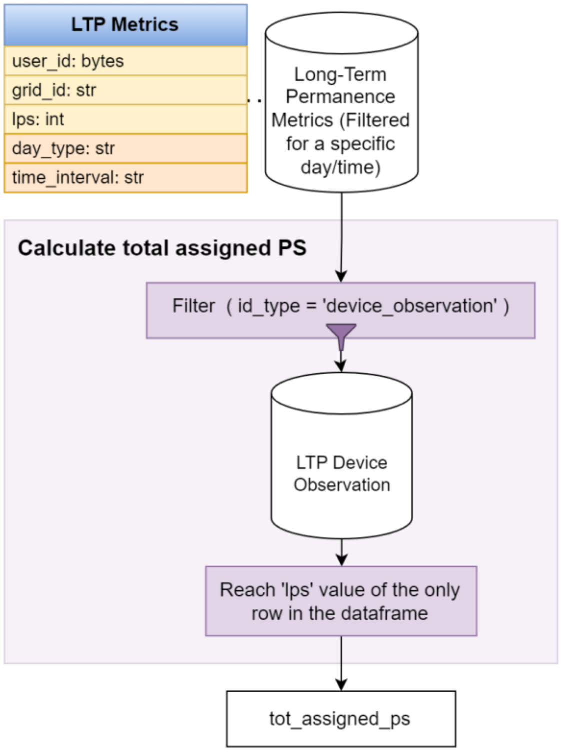
Labeled UE tiles
user_id: bytes
grid_id: str
label: str *('work')*
location_label_rule: str *('?')*
start_date: str
end_date: str
user_id_modulo: int

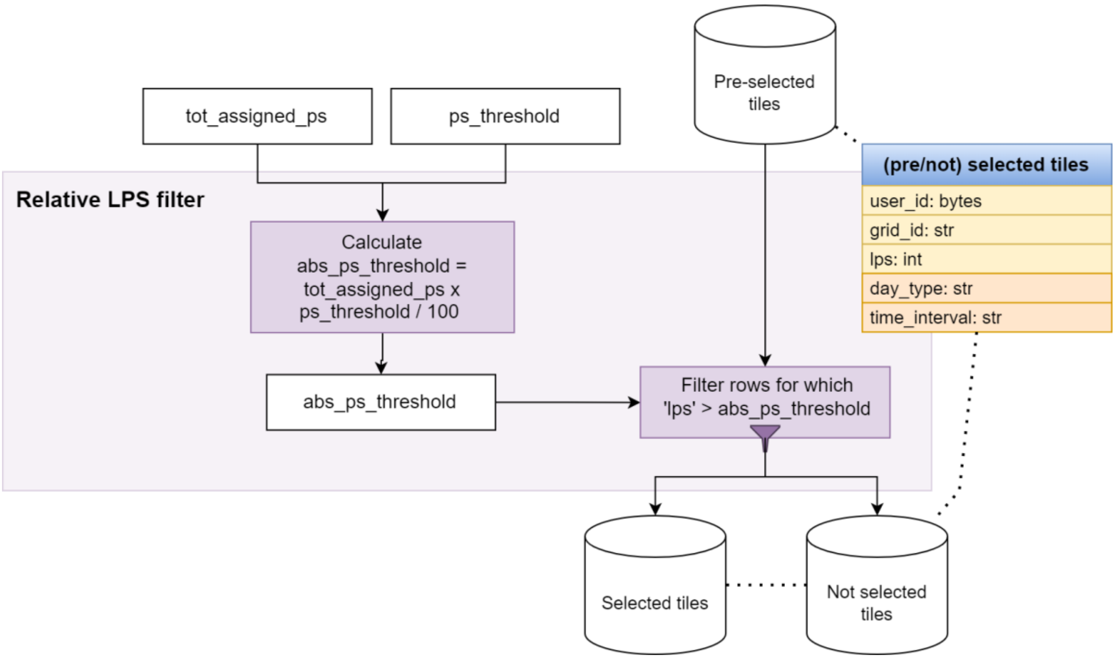
\ FUNCTION DETAILS:

- **PREPROCESSING (CUT TILE LIST AT GAP) FUNCTION:**

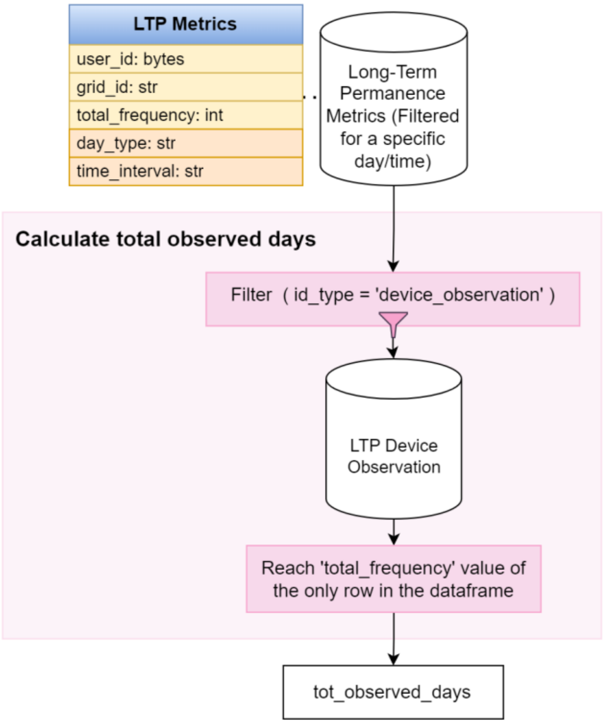


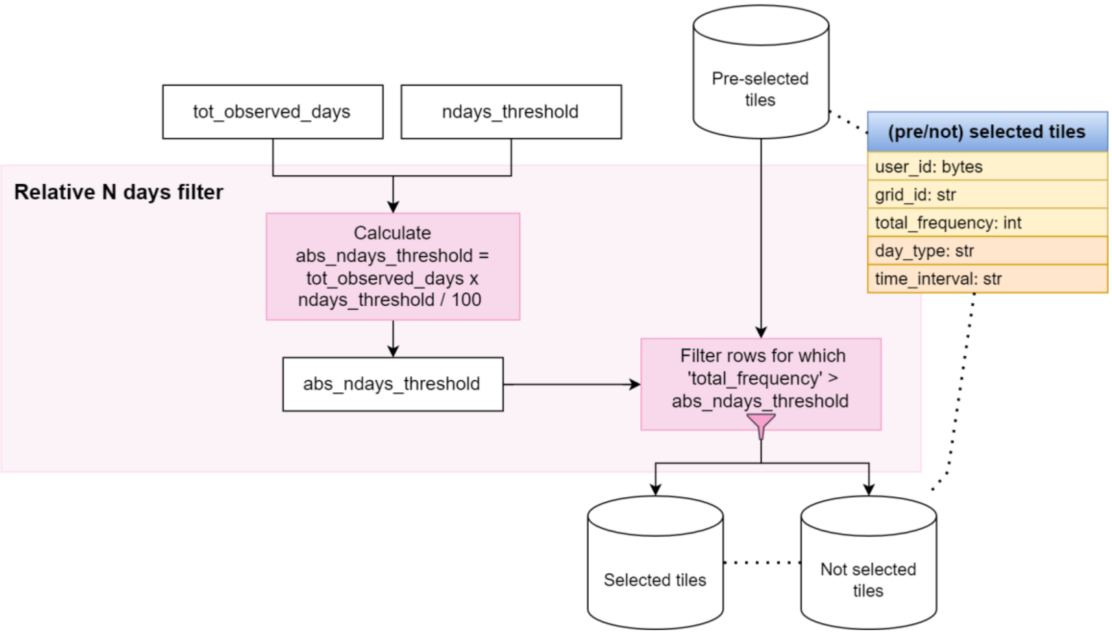
- **CALCULATE TOTAL ASSIGNED PS FUNCTION AND RELATIVE LPS FILTER FUNCTION:**


LTP Metrics
user_id: bytes
grid_id: str
lps: int
day_type: str
time_interval: str
Long-Term Permanence Metrics (Filtered for a specific day/time)
Calculate total assigned PS
Filter (id_type = 'device_observation')
LTP Device Observation
Reach 'lps' value of the only row in the dataframe
tot_assigned_ps

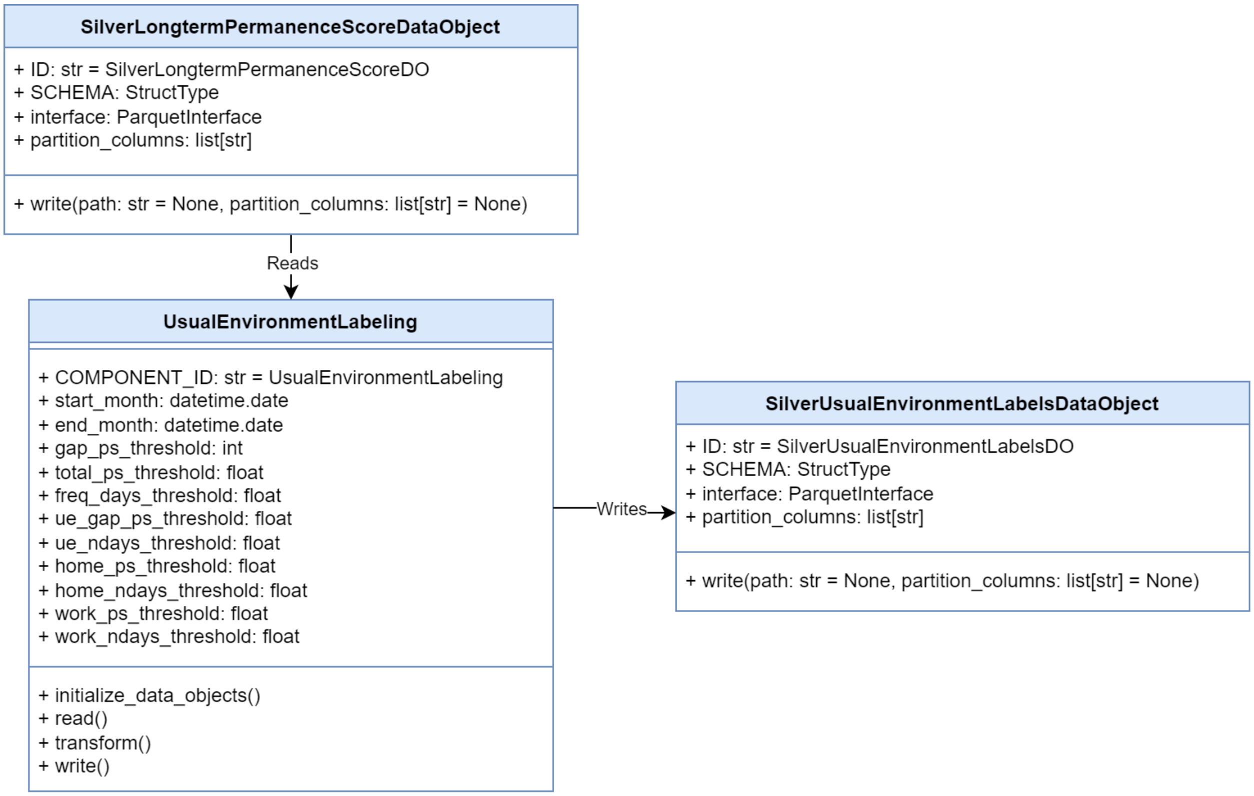


- **CALCULATE TOTAL OBSERVED DAYS FUNCTION AND RELATIVE N DAYS FUNCTION:**


LTP Metrics
user_id: bytes
grid_id: str
total_frequency: int
day_type: str
time_interval: str
Long-Term Permanence Metrics (Filtered for a specific day/time)
Calculate total observed days
Filter (id_type = 'device_observation')
LTP Device Observation
Reach 'total_frequency' value of the only row in the dataframe
tot_observed_days



CLASS DIAGRAM:



SilverLongtermPermanenceScoreDataObject

+ ID: str = SilverLongtermPermanenceScoreDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]

+ write(path: str = None, partition_columns: list[str] = None)

UsualEnvironmentLabeling

+ COMPONENT_ID: str = UsualEnvironmentLabeling
+ start_month: datetime.date
+ end_month: datetime.date
+ gap_ps_threshold: int
+ total_ps_threshold: float
+ freq_days_threshold: float
+ ue_gap_ps_threshold: float
+ ue_ndays_threshold: float
+ home_ps_threshold: float
+ home_ndays_threshold: float
+ work_ps_threshold: float
+ work_ndays_threshold: float

+ initialize_data_objects()
+ read()
+ transform()
+ write()

SilverUsualEnvironmentLabelsDataObject

+ ID: str = SilverUsualEnvironmentLabelsDO
+ SCHEMA: StructType
+ interface: ParquetInterface
+ partition_columns: list[str]

+ write(path: str = None, partition_columns: list[str] = None)

\ CODE STRUCTURE:

The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
/multimno_internal/
└── src
    └── components
        └── execution
            └── user_environment_labeling
                └── user_environment_labeling.py
```

user_environment_labeling.py contains one class named UserEnvironmentLabeling which is a subclass of Component. The UserEnvironmentLabeling class overrides some of the methods of Component:

The __init__ method first call its parent's __init__ method, which sets up the Spark session, initialises data objects and reads the configuration file.

transform method performs all necessary filtering and transformations pertaining to the user environment labeling calculation.

5.2.24 USUALENVIRONMENTAGGREGATION

5.2.24.1 MODULE DESCRIPTION

- **Module Name:** UEAggregation
- **Objectives:** Aggregate individual devices usual environment tiles over reference INSPIRE grid
- **Functionality:** The component takes Usual Environment Labels dataset for the given period and performs aggregation of individual devices over grid tiles. The component computes device weight in each tile in its usual environment either based on assumption of uniform distribution so that all tiles have the same weight or takes into account prior probabilities from land use information. Tile weights of all devices are then summed up per each tile.
    - Functionality specification: 3.2.23 UsualEnvironmentAggregation
- **Data Inputs and Outputs:**
    - Inputs:
        - I.37 UE Labels
        - I.31 Enriched Grid
    - Outputs:
        - I.44 Aggregated Usual Environments

5.2.24.2DEVELOPMENT DESIGN

- **Key Algorithms/Processes:**

Initialisation:

1. Read configuration parameters for component processing, date range for which to read labeling dataset, either to use land use information for device weights.
2. Clear the destination directory if configured.
3. Load input data objects for UE labels, and, optionally, enriched grid
4. Initialize the output data object for aggregated usual environment.

Processing:

1. Filter Usual Environment Labels dataset to required time range
2. Assign tile weights (tw) to each tile in device usual environment
    1. If landuse information is not used, all tile weights are assigned as 1.
    2. If landuse information is used, tile weights are assigned using landuse prior probabilities values.
3. Repeat next steps for all tiles for UE counts and for each label type (home, work):
    1. Calculate device weights weigth_td for each tile as:

    weight_td (grid_i) = tw (grid_i) / Σj(tw (grid_j))

    Where:

    grid_i: is a target grid tile, i.e., a tile that is included in the current device's usual environment, and for which we are calculating pue.

    weight_td (grid_i): is the weight of the device in the target grid tile (grid_1).

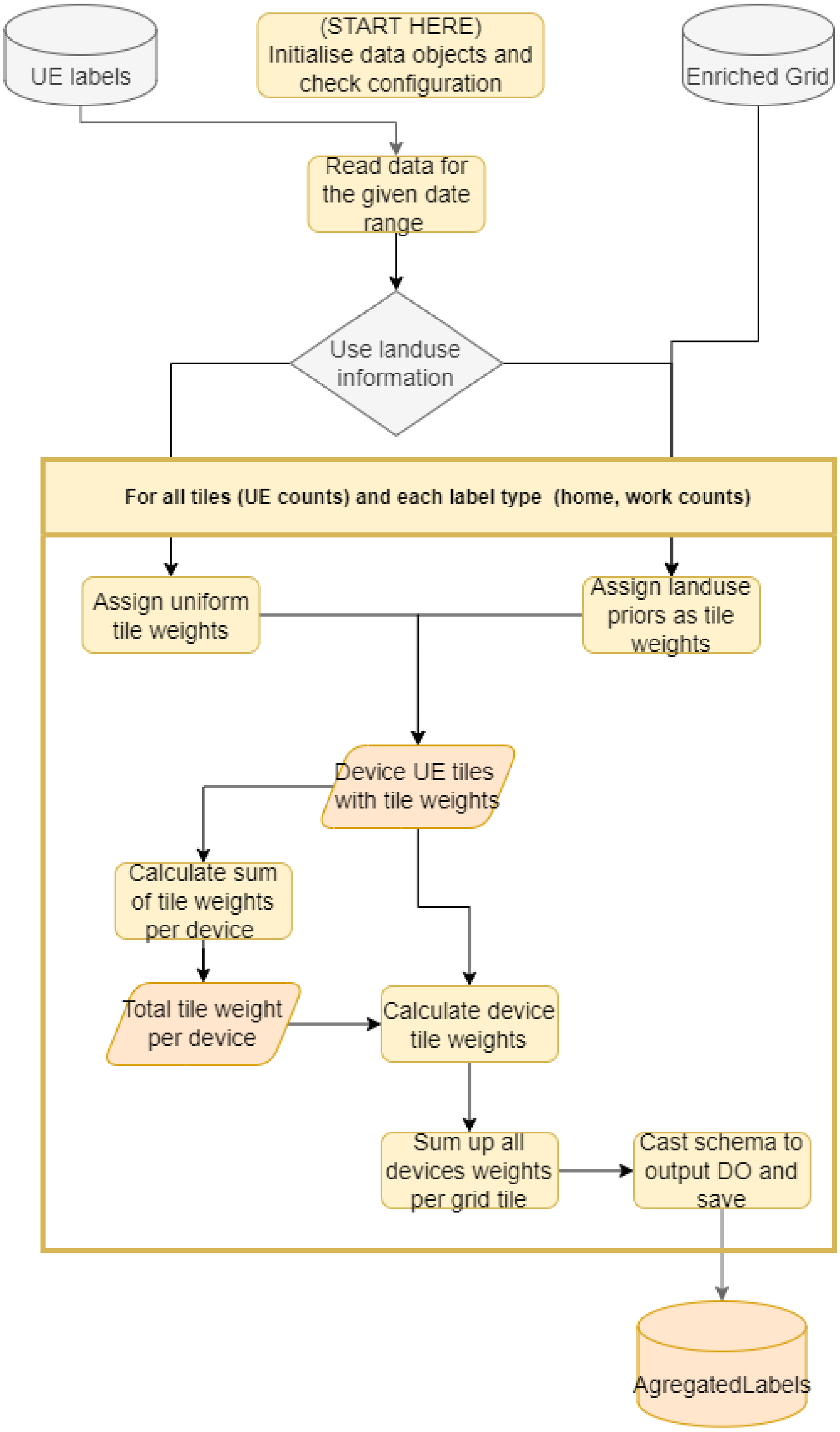
    tw (grid_i): is the tile weight for target grid tile (grid_1), either 1 or coming from the enriched grid data.

    Σj(tw (grid_j)): is the sum of the tile weights of all the grid tiles in the device's usual environment.

  b. Sum up all device weights per tile using grid_id

  c. Write aggregated results as a parquet partitioned by start and end date

- **Data flow diagram:**


UE labels
(START HERE)
Initialise data objects and check configuration
Enriched Grid
Read data for the given date range
Use landuse information
For all tiles (UE counts) and each label type (home, work counts)
Assign uniform tile weights
Assign landuse priors as tile weights
Device UE tiles with tile weights
Calculate sum of tile weights per device
Total tile weight per device
Calculate device tile weights
Sum up all devices weights per grid tile
Cast schema to output DO and save
AgregatedLabels

- **Code Structure:**

The code structure follows the format set by the core package, and the general repository structure. The location of the module script in the repository is as follows:

```
multimno
    └── components
        └── aggregation
            └── ue_aggregation
                └── ue_aggregation.py
```

ue_aggregation.py contains one class named UEAggregation which is a subclass of Component. The UEAggregation class overrides transform method of base Component class.

ANNEX I – DATA OBJECTS

I.1 MNO EVENT DATA – RAW

NAME	BRONZEEVENTDATAOBJECT
Description	'*MNO Event Data*' contains geolocation data from MNO subscribers. Data shall be created using at least one of the following data sources: (i) **CDRs** and/or (ii) **signalling data**. Additional information from MNO Apps can be added in order to improve the quality of the dataset, but this information is not mandatory. CDRs information shall contain all the information coming from voice, messages, internet connections, etc. CDRs shall also include roaming-in and roaming-out data. Each record of the dataset corresponds to a **MNO data event**, containing at least information about the identifier of the user, the timestamp of the event and the identifier of the cell to which the user is connected. When location information is estimated at point level (e.g. through signal triangulation or GPS data) information can be also be provided. This dataset shall only contain information about **personal mobile devices**. IoT, M2M and other related devices not associated to people shall not be included in the dataset.
Owner/Holder	MNO
Object/Unit/Record	Mobile network event associated to a specific subscriber
Contents	**Mandatory fields**: • **user_id:** o Type: Binary o Requirements: 32 bytes (256 bits) field. o Description: Unique pseudonymized identifier of the device, generated by hashing the user's IMSI using the SHA-256 function. • **timestamp:** o Type: String o Requirements: String with date and time following ISO:8601 format: YYYY-MM-DDThh:mm.ss o Description: Point in time where the event took place. • **mcc:** o Type: Integer o Requirement: 3 digits code o Description: Mobile Country Code derived from the user's IMSI • **mnc:** o Type: String o Requirement: 2- or 3-digits code o Description: Mobile Network Code, a code of a home operator. It might help to assess the selectivity bias that is in place due to preferential roaming agreements between MNOs. This must be string, as it can start with 0 digit. Possible options can also be 01 or 001, so it cannot be integer. • **plmn:** o Type: Integer o Requirement: 5- or 6-digits code. Mandatory only for outbound data o Description: Network identifier of the foreign roaming partner MNO consists of PLMN=MCC+MNC. • **cell_id:** o Type: String o Requirements: 14- or 15-character length string. All characters must be numbers. Optional if 'latitude' and 'longitude' are not null. o Description: Identifier of the cell following CGI and eCGI standards.

NAME	BRONZEEVENTDATAOBJECT
	• **latitude:** ○ Type: Float ○ Requirements: Latitude value in WGS84 system. Value must be within WGS84 bounds. Optional if ‘cell_id’ is not null. ○ Description: Latitude value of the location of the event. • **longitude**: ○ Type: Float ○ Requirements: Longitude value in WGS84 system. Value must be within WGS84 bounds. Optional if ‘cell_id’ is not null. ○ Description: Longitude value of the location of the event. **Optional fields**: • **loc_error**: ○ Type: Float ○ Requirements: Positive value. If ‘latitude’ and ‘longitude’ are null, this field shall be set to null. ○ Description: Location error in meters.

\ EXAMPLE

user_id	timestamp	mcc	mnc	plmn	cell_id	latitude	longitude	loc_error
000000000000..01	2023-01-01T00:00:00	214	01	null	214030412038931	–3.62958	40.51873	100.0
000000000000..10	2023-01-01T00:01:15	214	01	null	214030412038931	–3.62952	40.51871	100.0
000000000000..11	2023-01-01T12:05:03	214	01	null	214035484123541	null	null	null

I.2 MNO EVENT DATA – SYNTACTICALLY CLEANED

NAME	SILVEREVENTDATAOBJECT
Description	This data is basically the same as MNO Event Data - Raw. The only difference is that the events with syntactic errors have been removed.
Object/Unit/Record	Mobile network event associated to a specific subscriber
Contents	**Mandatory fields**: ● **year:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Year the event took place. ● **month:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Month the event took place. ● **day:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Day the event took place. ● **user_id:** ○ Type: Binary ○ Requirements: 32 bytes (256 bits) field. ○ Description: Unique pseudonymized identifier of the device. ● **user_id_modulo:** ○ Type: Integer ○ Requirements: Integer of 8 bits. ○ Description: Modulo division result, as applied to the integer part of the user_id column. ● **timestamp:** ○ Type: Time ○ Requirements: Parquet time type in hour, minutes and seconds. ○ Description: Point in time where the event took place. ● **mcc:** ○ Type: Integer ○ Requirement: 3 digits code ○ Description: Mobile Country Code derived from the user's IMSI. ● **mnc:** ○ Type: String ○ Requirement: 2- or 3-digits code ○ Description: Mobile Network Code, a code of a home operator. It might help to assess the selectivity bias that is in place due to preferential roaming agreements between MNOs. This must be string, as it can start with 0 digit. Possible options can also be 01 or 001, so it cannot be integer. ● **plmn:** ○ Type: Integer ○ Requirement: 5- or 6-digits code. Mandatory only for outbound data ○ Description: Network identifier of the foreign roaming partner MNO consists of PLMN=MCC+MNC. ● **cell_id:** ○ Type: String ○ Requirements: 14- or 15-character length string. All characters must be numbers. Optional if "latitude" and "longitude" are not null. ○ Description: Identifier of the cell following CGI and eCGI standards.

NAME	SILVEREVENTDATAOBJECT
	• **latitude:** ○ Type: Float ○ Requirements: Latitude value in WGS84 system. Value must be within WGS84 bounds. Optional if "cell_id" is not null. ○ Description: Latitude value of the location of the event. • **longitude**: ○ Type: Float ○ Requirements: Longitude value in WGS84 system. Value must be within WGS84 bounds. Optional if "cell_id" is not null. ○ Description: Longitude value of the location of the event. **Optional fields**: • **loc_error**: ○ Type: Integer ○ Requirements: Positive value ○ Description: Location error in meters.

\ EXAMPLE

year	month	day	user_id	timestamp	mcc	mnc	plmn	cell_id	lon	lat	loc_error
2023	01	01	000000000000..01	00:00:00	214	01	null	214030412038931	40.51873	-3.62958	100
2023	01	01	000000000000..01	00:01:15	214	01	null	214030412038931	40.51871	-3.62952	100
2023	01	01	000000000000..10	12:05:03	214	01	null	214035484123541	null	null	null

I.3 MNO EVENT DATA SYNTACTIC QUALITY METRICS – BY COLUMN

NAME	SILVEREVENTDATASYNTACTICQUALITYMETRICSBYCOLUMN
Description	Quality metrics produced by EventCleaning. It includes counts of records removed or labelled by variable and by type of error.
Object/Unit/Record	Quality metrics
Contents	**Mandatory fields:** • **result_timestamp**: ○ Type: TimestampType ○ Requirements: Timestamp ○ Description: Timestamp of the start of the process when the metrics were produced. One process can generate multiple metrics. • **date**: ○ Type: DateType ○ Requirements: The date that the data was about. ○ Description: The date for which the quality metrics were produced. • **variable:** ○ Type: StringType ○ Requirements: Must be a name of a column ○ Description: The name of the field to which the metric refers to. it could be null if the error refers to more then a variable. • **type of error** ○ Type: ShortType ○ Requirements: Integer of 16 bits. ○ Description: Shows which error occurred. Possible errors are in table below. • **type of transformation** ○ Type: ShortType ○ Requirements: Integer of 16 bits ○ Description: Shows which type of transformation occurred. Possible transformations are in table below. • **value**: ○ Type: IntegerType ○ Requirements: Integer of 32 bits. ○ Description: Count of records with the characteristics in the previous field

\ EXAMPLE

TYPE_OF_ERROR	ERROR_TYPE_DESCRIPTION
1	Missing value
2	Not right syntactic format
3	Out of admissible values
4	Inconsistency between variables
5	No location (no cell_id and no latitude&longitude), for that type or error there is None for variable column
6	Out of bounding box
7	No domain columns
9	No error
10	Different location duplicate
11	Same location duplicate

TYPE_OF_TRANSFORMATION	ERROR_TYPE_DESCRIPTION
1	Converted timestamps
2	Other conversion
9	No transformation

RESULT_TIMESTAMP	DATE	VARIABLE	TYPE_OF_ERROR	TYPE OF TRANSFORMATION	VALUE
2023-01-01 12:00:00	2022-12-01	cell_id	1	-	1000
2023-01-01 12:00:00	2022-12-01	cell_id	2	-	20
2023-01-01 12:00:00	2022-12-01	cell_id	9	-	10000
2023-01-01 12:00:00	2022-12-01	timestamp	-	1	1

I.4 MNO EVENT DATA SYNTACTIC QUALITY METRICS – FREQUENCY DISTRIBUTION

NAME	SILVEREVENTDATASYNTACTICQUALITYMETRICSFREQUENCYDISTRIBUTION
Description	Quality metrics produced by EventCleaning. This data object includes a table to show distribution of records by **user_id** and **cell_id** before and after the application of MNO Event Data Syntactic Quality Metrics method.
Object/Unit/Record	Quality metrics
Contents	**Mandatory fields**: • **cell_id:** ○ Type: StringType ○ Requirements: 14- or 15-character length string. All characters must be numbers. ○ Description: Identifier of the cell following CGI and eCGI standards. • **user_id:** ○ Type: BinaryType ○ Requirements: 32 bytes (256 bits) field. ○ Description: Unique pseudonymized identifier of the device, generated by hashing the user's IMSI using the SHA-256 function. • **initial_frequency:** ○ Type: IntegerType ○ Requirements: Integer of 32 bits. ○ Description: Number of records with given cell_id and user_id before filtering. • **final_frequency:** ○ Type: IntegerType ○ Requirements: Integer of 32 bits. ○ Description: Number of records with given cell_id and user_id after filtering. • **date:** ○ Type: DateType ○ Requirements: Date of the data in UTC. ○ Description: Date of the data in UTC.

\ EXAMPLE

cell_id	user_id	date	initial_frequency	final_frequency
214030412038931	000000000000..01	2023-07-20	200	10
214030412038931	000000000000..01	2023-07-21	600	600

I.5 MNO EVENT DATA QUALITY WARNINGS – LOG TABLE

NAME	SILVEREVENTDATASYNTACTICQUALITYWARNINGSLOGTABLE
Description	Data Object is meant to store warnings in unified format.
Object/Unit/Record	Quality warnings
Contents	**Mandatory fields**: • **date:** ○ Type: Date ○ Description: date a warning happened. • **measure_definition:** ○ Type: String ○ Description: A name of warning group, e.g. 'Error rate …' • **lookback_period:** ○ Type: String ○ Description: The text representation of a lookback period, e.g. 'week' meaning 7 days • **daily_value:** ○ Type: Float ○ Description: The value that does not meet warning condition. • **condition_value:** ○ Type: Float ○ Description: Value to compare with daily_value to check if condition is met. • **condition**: ○ Type: String ○ Description: Condition description. • **warning_text**: ○ Type: String ○ Description: Warning description

\ EXAMPLE

date	measure_definition	lookback_period	daily_value	condition_value	condition	warning_text
2024-01-29	Error rate for date	week	23.41	22.48	Error rate is over the upper control limit calculated on the basis of average and standard deviation of the distribution of the error rate in previous	The error rate after syntactic checks application is unexpectedly high with respect to previous period, taking into account its usual variability

I.6 MNO EVENT DATA – DEDUPLICATED

NAME	SILVEREVENTDATAOBJECT
Description	This data is schematically identical to I.1 MNO Event Data – Raw. The difference is that duplicated rows have been removed.
Object/Unit/Record	Mobile network event associated to a specific subscriber
Contents	**Mandatory fields**: • **year:** o Type: Integer 16 o Requirements: Integer of 16 bits. o Description: Year the event took place. • **month:** o Type: Integer 8 o Requirements: Integer of 8 bits. o Description: Month the event took place. • **day:** o Type: Integer 8 o Requirements: Integer of 8 bits. o Description: Day the event took place. • **user_id:** o Type: Binary o Requirements: 32 bytes (256 bits) field. o Description: Unique pseudonymized identifier of the device. • **user_id_modulo:** o Type: Integer o Requirements: Integer of 8 bits. o Description: Modulo division result, as applied to the integer part of the user_id column. • **timestamp:** o Type: Time o Requirements: Parquet time type in hour, minutes and seconds. o Description: Point in time where the event took place. • **mcc:** o Type: Integer o Requirement: 3 digits code o Description: Mobile Country Code derived from the user's IMSI. • **cell_id:** o Type: String o Requirements: 14- or 15-character length string. All characters must be numbers. Optional if 'latitude' and 'longitude' are not null. o Description: Identifier of the cell following CGI and eCGI standards. • **latitude:** o Type: Float o Requirements: Latitude value in WGS84 system. Value has to be within WGS84 bounds. Optional if 'cell_id' is not null. o Description: Latitude value of the location of the event. • **longitude**: o Type: Float o Requirements: Longitude value in WGS84 system. Value has to be within WGS84 bounds. Optional if 'cell_id' is not null. o Description: Longitude value of the location of the event. **Optional fields**: • **loc_error**: o Type: Integer

NAME	SILVEREVENTDATAOBJECT
	○ Requirements: Positive value ○ Description: Location error in meters.

\ EXAMPLE

year	month	day	user_id	timestamp	mcc	cell_id	lon	lat	loc_error
2023	01	01	000000000000..01	00:00:00	214	214030412038931	40.51873	-3.62958	100
2023	01	01	000000000000..01	00:01:15	214	214030412038931	40.51871	-3.62952	100

I.7 CELL LOCATIONS WITH PHYSICAL PROPERTIES - RAW

NAME	BRONZENETWORKDATAOBJECT
Description	Contains information about the location and physical properties of network cells for a specific day. Data updated along with MNO event data representing the network parameters for all active cells for a specific date.
Owner/Holder	MNO
Object/Unit/Record	Characteristic of a specific cell
Contents	**Mandatory fields**: • **cell_id**: o Type: String o Requirements: 14-digit or 15-digit numeric code following CGI and eCGI standards o Description: Code uniquely identifying one cell. • **latitude:** o Type: Float o Requirements: Latitude value in WGS84 system. Value must be within WGS84 bounds. o Description: Latitude of cell location (location of the antenna). • **longitude:** o Type: Float o Requirements: Longitude in WGS84 system. Value must be within WGS84 bounds. o Description: Longitude of cell location (location of the antenna). • **directionality:** o Type: Integer o Requirements: value is either 0 or 1 o Description: 0 for omnidirectional antennas and 1 for directional antenas. • **azimuth_angle:** o Type: Float, nullable o Requirements: value between 0 and 360 if ‘directionality’ equal to 1, null otherwise. o Description: angle in degrees of the main propagation direction with respect to the North clockwise; for directional cells only. **Optional fields**: • **altitude:** o Type: Float o Requirements: o Description: Altitude (meters) of the antenna base from the sea level. • **antenna_height:** o Type: Float o Requirements: Positive value o Description: Height of the antenna in meters from ground • **elevation_angle:** o Type: Float o Requirements: value between -90 and 90 o Description: Antenna placement angle; also known as tilt • **horizontal_beam_width:** o Type: Float o Requirements: value between 0 and 360 o Description: The angular extent of the cell beam in the horizontal plane • **vertical_beam_width:** o Type: Float o Requirements: value between 0 and 360

NAME	BRONZENETWORKDATAOBJECT
	○ Description: The angular extent of the cell beam in the vertical plane • **power:** ○ Type: Float ○ Requirements: Positive value ○ Description: W • **range:** ○ Type: Float ○ Requirements: Positive value ○ Description: maximum coverage range of the cell, in metres • **frequency:** ○ Type: Integer ○ Requirements: Positive value ○ Description: MHz • **technology:** ○ Type: String ○ Requirements: ○ Description: Technology of the cell. • **valid_date_start:** ○ Type: String ○ Requirements: String with date and time following ISO:8601 format: YYYY-MM-DDThh:mm.ss. Has to be earlier than **valid_period_end**. ○ Description: Start of time window in which the antenna is operational in this location. Period start timestamp is *included* within the time window. • **valid_date_end:** ○ Type: String, nullable ○ Requirements: String with date and time following ISO:8601 format: YYYY-MM-DDThh:mm.ss. Has to be later than **valid_period_start**. It shall be set to null if it still operational. ○ Description: End of time window in which the antenna is operational in this location. Period end timestamp is *excluded* from the time window. • **cell_type:** ○ Type: String ○ Requirements: ○ Description: picocell, femtocell, etc. • **year:** ○ Type: Integer 16. ○ Description: Year the register corresponds to. • **month:** ○ Type: Integer 8. ○ Description: Month the register corresponds to. • **day:** ○ Type: Integer 8. ○ Description: Day the register corresponds to.

\ EXAMPLE

cell_id	latitude	longitude	altitude	antenna_height	directionality	azimuth_angle	elevation_angle	horizontal_beam_width	vertical_beam_width	power	frequency	technology	valid_date_start	valid_date_end	cell_type	year	month	day
214030412038931	-3.62958	40.51873	20.0	42	1	90	4	65	9	3	3500	LTE	2023-07-20T10:00:00	2023-12-31T23:30:00	TBD	2023	10	10
214035484123541	-3.8245	40.8952	30.5	12	0	null	5	42	9	7	1800	LTE	2023-07-20T12:34:56	null	TBD	2023	10	10

I.8 CELL LOCATIONS WITH PHYSICAL PROPERTIES – CLEANED

NAME	SILVERNETWORKDATAOBJECT
Description	Contains syntactically cleaned information about the location and physical properties of network cells for a specific day.
Object/Unit/Record	Characteristic of a specific cell
Contents	**Mandatory fields**: • **cell_id**: ○ Type: String ○ Description: Code uniquely identifying one cell. • **latitude:** ○ Type: Float ○ Description: Latitude of cell location (location of the antenna). • **longitude:** ○ Type: Float ○ Description: Longitude of cell location (location of the antenna). • **altitude:** ○ Type: Float, nullable ○ Description: Altitude (meters) of the antenna base from the sea level. • **antenna_height:** ○ Type: Float, nullable ○ Description: Height of the antenna in meters from ground • **directionality:** ○ Type: Integer, nullable ○ Description: 0 for omnidirectional antennas and 1 for directional antenas. • **azimuth_angle:** ○ Type: Float, nullable ○ Description: angle in degrees of the main propagation direction with respect to the North clockwise; for directional cells only. • **elevation_angle:** ○ Type: Float, nullable ○ Description: Antenna placement angle; also known as tilt • **horizontal_beam_width:** ○ Type: Float, nullable ○ Description: The angular extent of the cell beam in the horizontal plane • **vertical_beam_width:** ○ Type: Float, nullable ○ Description: The angular extent of the cell beam in the vertical plane • **power:** ○ Type: Float, nullable ○ Description: W • **range:** ○ Type: Float, nullable ○ Description: maximum coverage range of the cell, in metres • **frequency:** ○ Type: Integer, nullable ○ Description: MHz • **technology:** ○ Type: String, nullable ○ Description: Technology of the cell. • **valid_date_start:** ○ Type: String, nullable ○ Description: Start of time window in which the antenna is operational in this location. Period start timestamp is *included* within the time window.

NAME	SILVERNETWORKDATAOBJECT
	• **valid_date_end:** ○ Type: String, nullable ○ Description: End of time window in which the antenna is operational in this location. Period end timestamp is *excluded* from the time window. • **cell_type:** ○ Type: String, nullable ○ Description: normal, picocell, femtocell, etc. • **year:** ○ Type: Integer 16. ○ Description: Year corresponding to the register. • **month:** ○ Type: Integer 8. ○ Description: Month correspoding to the register. • **day:** ○ Type: Integer 8. ○ Description: Day corresponding to the register.

\ EXAMPLE

cell_id	latitude	longitude	altitude	antenna_height	directionality	azimuth_angle	elevation_angle	horizontal_beam_width	vertical_beam_width	power	frequency	technology	valid_date_start	valid_date_end	cell_type	year	month	day
214030412038931	-3.62958	40.51873	20.0	42	1	90	4	65	9	3	3500	LTE	2023-07-20T10:00:00	2023-12-31T23:30:00	normal	2023	07	01
214035484123541	-3.8245	40.8952	30.5	12	0	null	5	42	9	7	1800	LTE	2023-07-20	null	microcell	2023	07	01

I.9 MNO NETWORK TOPOLOGY DATA QUALITY METRICS

NAME	SILVERNETWORKDATAQUALITYMETRICSBYCOLUMN
Description	Quality metrics produced by NetworkCleaning.
Object/Unit/Record	Quality metrics
Contents	**Mandatory fields**: • **result_timestamp**: ○ Type: Timestamp ○ Requirements: - ○ Description: Timestamp of the start of the process when the metrics were produced. One process can generate multiple metrics. • **date**: ○ Type: Date ○ Requirements: - ○ Description: Date of the dataset to which the quality metrics refer (not from topology data but from parameters) • **field_name**: ○ Type: String ○ Requirements: Either null or same as the name of a column present in input data ○ Description: Name of the field to which the metric refers to. Value is null if the metric refers to multiple fields. • **type_code**: ○ Type: Integer ○ Requirements: One value from the type codes (see table below). ○ Description: Numeric code indicating the type of the metric. See table below. • **value**: ○ Type: Integer ○ Requirements: - ○ Description: Numeric value of the metric. • **year:** ○ Type: Integer 16 ○ Description: Year the event took place. Partition column • **month:** ○ Type: Integer 8 ○ Description: Month the event took place. Partition column. • **day:** ○ Type: Integer 8 ○ Description: Day the event took place. Partition column.

\ CODE TYPES

CODE	SHORT DESCRIPTION	DESCRIPTION
0	no errors	
1	value is null	
2	value is not within the set of accepted values	
3	unsupported input data type	
4	unable to parse correctly	
100	total rows at the start of method	
101	total rows at the end of method	

\ EXAMPLE

result_timestamp	date	field_name	type_code	value	year	month	day
2023-06-12 12:00:00	01-01-2023	cell_id	0	1900	2023	1	1
2023-06-12 12:00:00	01-01-2023	cell_id	1	95	2023	1	1
2023-06-12 12:00:00	01-01-2023	cell_id	2	5	2023	1	1
2023-06-12 12:00:00	01-01-2023	-	100	2000	2023	1	1
2023-06-12 12:00:00	01-01-2023	-	101	1900	2023	1	1

I.10 MNO NETWORK TOPOLOGY DATA QUALITY WARNINGS – LOG TABLE

NAME	SILVERNETWORKDATASYNTACTICQUALITYWARNINGSLOGTABLE
Description	Quality warnings log table produced by NetworkQualityWarnings.
Object/Unit/Record	Quality Warning logs
Contents	**Mandatory fields**: • **title**: ○ Type: String ○ Requirements: 'MNO Network Topology Data Quality Warnings'. ○ Description: Title of the log table warnings. • **timestamp**: ○ Type: Timestamp ○ Requirements: - ○ Description: Timestamp of the start of the process when the quality warnings were produced. • **date**: ○ Type: Date ○ Requirements: - ○ Description: Date of the dataset to which the quality metrics analysed refer to (contained in their own 'date' field). • **measure_definition**: ○ Type: String ○ Requirements: - ○ Description: Name of the metric or measure that was studied in order to raise a warning. • **daily_value**: ○ Type: Float ○ Requirements: Non-negative value. ○ Description: Value that the metric that raised this warning had in this date. • **condition**: ○ Type: String ○ Requirements: - ○ Description: Logical condition that the daily_value had to verify in order to raise a warning. • **parameter_time**: ○ Type: String ○ Requirements: 'week', 'month', or 'quarter'. ○ Description: Lookback period length used to compute the average and sample standard deviation of the metric being studied. • **condition_value**: ○ Type: Float ○ Requirements: Non-negative value ○ Description: Threshold value that the daily_value was compared with in order to fulfill the condition of this warning. • **warning_text**: ○ Type: String ○ Requirements: - ○ Description: Verbose description of what this warning means and implies.

\ CODE TYPES

CODE	SHORT DESCRIPTION	DESCRIPTION
0	no errors	
1	value is null	
2	value is not within the set of accepted values	
3	unsupported input data type	
4	unable to parse correctly	
100	total rows at the start of method	
101	total rows at the end of method	

\ EXAMPLE

title	timestamp	date	measure_definition	daily_value	condition	parameter_time	condition_value	warning_text
MNO Network Topology Data Quality Warnings	2024-02-01 10:00:00	2024-01-29	Error rate	3.73	Error rate is over the upper control limit calculated on the basis of average and standard deviation of the distribution of the error rate in the previous period. Upper control limit = 3.70.	week	3.70	The error rate after syntactic checks application is unexpectedly high with respect to previous period, taking into account its usual variability

I.11 REFERENCE GRID

DESCRIPTION	INSPIRE GRID GEOMETRY WITH ADDITIONAL INFORMATION
Object/Unit/Record	Grid centroid geometry with additional information
Contents	**Mandatory fields**: ● **grid_id**: o Type: String o Requirements: string following INSPIRE specification format o Description: Code uniquely identifying one grid tile. ● **geometry:** o Type: Binary o Requirements: ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035) o Description: grid centroids point geometry **Optional fields**: ● **elevation:** o Type: Float o Requirements: o Description: Elevation of a grid centroids ● **land_use_main** o Type: string o Main land use category ● **prior_probabilty_value** o Type: float o Prior probability value.

\ EXAMPLE

grid_id	elevation	land_use_main	prior_probabilty_value	geometry
100mN4056000E5275300	12.1	RURAL	0.00	POINT()
100mN4056000E5275400	11.9	URBAN	0.70	POINT()

I.12 CELLS SIGNAL STRENGTHS

DESCRIPTION	THE SIGNAL STRENGTH VALUES PER CELL PER GRID TILE
Object/Unit/Record	Cell / grid tile combination
Contents	Mandatory fields: • **cell_id** ○ Type: String ○ Description: Unique ID of cell • **grid_id** ○ Type: String ○ Description: Unique ID of grid tile • **valid_date_start** ○ Type: Timestamp ○ Description: Start date of validity period (inclusive) • **valid_date_end** ○ Type: Timestamp ○ Description: End date of validity period (exclusive) • **signal_strength** ○ Type: Float ○ Description: Signal strength in dBm • **year:** ○ Type: Integer 16. ○ Description: Year the intersection group determined. • **month:** ○ Type: Integer 8. ○ Description: Month the intersection group determined. • **day:** ○ Type: Integer 8. ○ Description: Day the intersection group determined. Optional fields: • **distance_to_cell** ○ Type: Integer ○ Description: Distance of grid tile to cell location may be necessary for some calculation during the Location Assignation Module (e.g., taking into account the Timing Advance parameter of the MNO event data).

\ EXAMPLE

cell_id	grid_id	valid_date_start	valid_date_end	signal_strength	distance_to_cell	year	month	day
214030412038931	123231342131341	2023-01-01	2023-01-01	-120	4623	2023	01	01
214030412038931	123231342131342	2023-01-01	2023-01-01	-78	4627	2023	01	01
214030412038932	123231342131341	2023-02-01	2023-02-01	-59	4629	2023	02	01

I.13 CELL FOOTPRINTS

DESCRIPTION	THE SIGNAL DOMINANCE (CELL FOOTPRINT) VALUES PER GRID TILE
Object/Unit/Record	Cell / grid tile combination
Contents	**Mandatory fields:** • **cell_id** o Type: String o Description: Unique ID of cell • **grid_id** o Type: String o Description: Unique ID of grid tile • **valid_date_start** o Type: Date o Description: Start date of validity period (inclusive) • **valid_date_end** o Type: Date o Description: End date of validity period (exclusive) • **signal_dominance** o Type: Float o Description: Signal dominance value (0 to 1) • **year:** o Type: Integer 16. o Description: Year the intersection group determined. • **month:** o Type: Integer 8. o Description: Month the intersection group determined. • **day:** o Type: Integer 8. o Description: Day the intersection group determined.

\ EXAMPLE

cell_id	grid_id	valid_date_start	valid_date_end	signal_dominance	year	month	day
123456789101112	123231342131341	2023-01-01	2023-01-02	0.5405	2023	01	01
123456789101112	123231342131342	2023-01-01	2023-01-02	0.4193	2023	01	01

I.14 CELL INTERSECTION GROUPS

NAME	SILVERCELLINTERSECTIONGROUPSDATAOBJECT
Description	Groups of cells which coverage areas (cell footprints) overlaps. Contains all overlapping combinations of pairs, triples, quads and so on up to the maximum size of the overlapping cells cluster.
Object/Unit/Record	Groups of cells which coverage areas overlaps for a single day
Contents	**Mandatory fields**: • **group_id:** ○ Type: String. ○ Description: Intersection group ID. • **cells:** ○ Type: Array of strings. ○ Description: Array of overlapping cells. • **group_size:** ○ Type: Integer. ○ Description: The number of cells in this intersection group. • **valid_date_start** ○ Type: Timestamp ○ Description: Start date of validity period (inclusive) • **valid_date_end** ○ Type: Timestamp ○ Description: End date of validity period (exclusive) • **year:** ○ Type: Integer 16. ○ Description: Year the intersection group determined. • **month:** ○ Type: Integer 8. ○ Description: Month the intersection group determined. • **day:** ○ Type: Integer 8. ○ Description: Day the intersection group determined.

\ EXAMPLE

group_id	cells	group_size	valid_date_start	valid_date_end	year	month	day
2_13	[214030412038931, 214030412038932]	2	2023-01-01	2023-01-01	2023	01	01
2_14	[214030412038964, 214030412038965]	2	2023-01-01	2023-01-01	2023	01	01
4_2	[214030412038931, 214030412038932, 214030412038935, 214030412038938]	4	2023-02-01	2023-02-01	2023	02	01

I.15 CELL CONNECTION AND POSTERIOR PROBABILITIES

DESCRIPTION	CELL CONNECTION AND POSTERIOR PROBABILITY VALUES PER GRID TILE AND CELL_ID.
Object/Unit/Record	Cell / grid tile combination
Contents	**Mandatory fields:** • **cell_id** ○ Type: String ○ Description: Unique ID of cell • **grid_id** ○ Type: String ○ Description: Unique ID of grid tile • **valid_date_start** ○ Type: Date ○ Description: Start date of validity period (inclusive) • **valid_date_end** ○ Type: Date ○ Description: End date of validity period (exclusive) • **cell_connection_probability** ○ Type: Float ○ Description: Connection probability value within range [0, 1] • **posterior_probability** ○ Type: float ○ Posterior probability value within range [0, 1] • **year:** ○ Type: Integer 16. ○ Description: Year the intersection group determined. • **month:** ○ Type: Integer 8. ○ Description: Month the intersection group determined. • **day:** ○ Type: Integer 8. ○ Description: Day the intersection group determined.

\ EXAMPLE

cell_id	grid_id	valid_date_start	valid_date_end	cell_connection_probability	posterior_probability	year	month	day
123456789101112	123231342131341	2023-01-01	2023-01-02	0.5405	0.2192	2023	01	01
123456789101112	123231342131342	2023-01-01	2023-01-02	0.4193	0.5411	2023	01	01

I.16 MNO EVENT DATA – SEMANTICALLY CLEANED

NAME	SILVEREVENTFLAGGEDDATAOBJECT
Description	Mobile network event data associated to a specific subscriber, after semantic checks has been completed with semantic error flags.
Object/Unit/Record	Mobile network event associated to a specific subscriber with semantic error flags
Contents	**Mandatory fields**: • **user_id:** ○ Type: Binary ○ Description: Unique pseudonymized identifier of the device. • **timestamp:** ○ Type: Time ○ Description: Point in time where the event took place. • **mcc:** ○ Type: Integer ○ Description: Mobile Country Code derived from the user's IMSI. • **mnc:** ○ Type: String ○ Description: Mobile Network Code, a code of a home operator. It might help to assess the selectivity bias that is in place due to preferential roaming agreements between MNOs. This must be string, as it can start with 0 digit. Possible options can also be 01 or 001, so it cannot be integer. • **plmn:** ○ Type: Integer ○ Requirement: 5- or 6-digits code. Mandatory only for outbound data ○ Description: Network identifier of the foreign roaming partner MNO consists of PLMN=MCC+MNC. • **cell_id:** ○ Type: String ○ Description: Identifier of the cell following CGI and eCGI standards. Optional if “latitude” and “longitude” are not null. • **latitude:** ○ Type: Float ○ Description: Latitude value of the location of the event. Optional if “cell_id” is not null. • **longitude**: ○ Type: Float ○ Description: Longitude value of the location of the event. Optional if “cell_id” is not null. • **error_flag**: ○ Type: Integer, referring to global error type code ○ Description: Error flag referring to a error type code of the specific identified error • **loc_error**: ○ Type: Integer ○ Description: Location error in meters. • **year:** ○ Type: Integer 16 ○ Description: Year the event took place. • **month:** ○ Type: Integer 8 ○ Description: Month the event took place. • **day:** ○ Type: Integer 8

NAME	SILVEREVENTFLAGGEDDATAOBJECT
	○ Description: Day the event took place. ● **user_id_modulo** ○ Type: Integer ○ Description: Partition key

I.17 MNO DEVICE SEMANTIC QUALITY METRICS

NAME	SILVEREVENTSEMANTICQUALITYMETRICS
Description	Quality metrics obtained in SemanticCleaning.
Object/Unit/Record	Quality metric
Contents	**Mandatory fields**: • **result_timestamp**: ○ Type: Timestamp ○ Requirements: - ○ Description: Timestamp of the start of the process when the metrics were produced. One process can generate multiple metrics. • **variable**: ○ Type: String ○ Requirements: Same as the name of a column present in input data ○ Description: Name of the field to which the metric refers to. Value is null if the metric refers to multiple fields. • **type_of_error**: ○ Type: Integer ○ Requirements: One value from the type codes (see table below). ○ Description: Numeric code indicating the type of the metric. See table below. • **value**: ○ Type: Integer ○ Requirements: - ○ Description: Numeric value of the metric. • **year:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Year of the datasets used. • **month:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Month of the datasets used. • **day:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Day of the datasets used.

\ CODE TYPES

CODE	SHORT DESCRIPTION	DESCRIPTION
0	No error	
1	Cell ID non-existent	Event made a reference to a non-existent cell ID
2	Invalid cell ID	Event made a reference to an existent cell ID, but the cell was not operative when the event was registered
3	Incorrect event location	
4	Suspicious event location	
5	Different location duplicate	Event has the same timestamp for the same user on either a previous or following row, but not identical values in the columns cell_id, longitude, latitude.

EXAMPLE

result_timestamp	variable	type_of_error	value	year	month	day
2024-03-01T09:03:08.432637Z	cell_id	3	50	2024	02	19
2024-03-01T09:03:08.432637Z	cell_id	2	103	2024	02	19

I.18 MNO EVENT DATA AT DEVICE LEVEL SEMANTIC QUALITY WARNINGS – LOG TABLE

NAME	SILVEREVENTSEMANTICQUALITYWARNINGSLOGTABLE
Description	Quality warnings log table produced by SemanticQualityWarnings.
Object/Unit/Record	Quality Warning logs
Contents	**Mandatory fields**: • **date**: o Type: Date o Requirements: - o Description: Date of the datasets that produced the quality metrics. • **Error 1**: o Type: Float o Requirements: - o Description: Percentage of the error type 1 over the total of events in this date. • **Error 2**: o Type: Float o Requirements: - o Description: Percentage of the error type 2 over the total of events in this date. • **Error 3**: o Type: Float o Requirements: - o Description: Percentage of the error type 3 over the total of events in this date. • **Error 4**: o Type: Float o Requirements: - o Description: Percentage of the error type 4 over the total of events in this date. • **Error 5**: o Type: Float o Requirements: - o Description: Percentage of the error type 5 over the total of events in this date. • **Error 1 upper control limit**: o Type: Float o Requirements: Can be null if there any date of the lookback period was missing. o Description: Threshold value that the percentage of this error type must surpass in order to raise a warning. • **Error 2 upper control limit**: o Type: Float o Requirements: Can be null if there any date of the lookback period was missing. o Description: Threshold value that the percentage of this error type must surpass in order to raise a warning. • **Error 3 upper control limit**: o Type: Float o Requirements: Can be null if there any date of the lookback period was missing. o Description: Threshold value that the percentage of this error type must surpass in order to raise a warning. • **Error 4 upper control limit**: o Type: Float o Requirements: Can be null if there any date of the lookback period was missing. o Description: Threshold value that the percentage of this error type must surpass in order to raise a warning. • **Error 5 upper control limit**: o Type: Float

NAME	SILVEREVENTSEMANTICQUALITYWARNINGSLOGTABLE
	○ Requirements: Can be null if there any date of the lookback period was missing. ○ Description: Threshold value that the percentage of this error type must surpass in order to raise a warning. • **Error 1 display warning**: ○ Type: Boolean ○ Requirements: - ○ Description: Whether a warning regarding error type 1 was raised. • **Error 2 display warning**: ○ Type: Boolean ○ Requirements: - ○ Description: Whether a warning regarding error type 2 was raised. • **Error 3 display warning**: ○ Type: Boolean ○ Requirements: - ○ Description: Whether a warning regarding error type 3 was raised. • **Error 4 display warning**: ○ Type: Boolean ○ Requirements: - ○ Description: Whether a warning regarding error type 4 was raised. • **Error 5 display warning**: ○ Type: Boolean ○ Requirements: - ○ Description: Whether a warning regarding error type 5 was raised. • **year:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Year of the datasets used. • **month:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Month of the datasets used. • **day:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Day of the datasets used. • **execution_id** ○ Type: Timestamp ○ Requirements: - ○ Description: execution ID of the calculation of a given row of warnings by using the moment in time they were computed as identifier.

\ EXAMPLE

date	Error 1	Error 2	Error 3	Error 4	Error 1 upper control limit	Error 2 upper control limit	Error 3 upper control limit	Error 4 upper control limit	Error 1 display warning	Error 2 display warning	Error 3 display warning	Error 4 display warning	year	month	day	execution_id
2023-07-15	0.000023%	0.013808%	0.023779%	0.006592%					FALSE	FALSE	FALSE	FALSE	2023	7	15	2024-03-01T09:03:08.432637Z
2023-07-16	0.000033%	0.012508%	0.027985%	0.006316%					FALSE	FALSE	FALSE	FALSE	2023	7	16	2024-03-01T10:03:08.432637Z
2023-07-17	0.000015%	0.011912%	0.009095%	0.005089%	0.000038%	0.015686%	0.031660%	0.007711%	FALSE	FALSE	FALSE	FALSE	2023	7	17	2024-03-01T11:03:08.432637Z
2023-07-18	0.000016%	0.013569%	0.008527%	0.006879%	0.000037%	0.015761%	0.032400%	0.007504%	FALSE	FALSE	FALSE	FALSE	2023	7	18	2024-03-01T12:03:08.432637Z
2023-07-19	0.000003%	0.012459%	0.005626%	0.006862%	0.000034%	0.015630%	0.033952%	0.007731%	FALSE	FALSE	FALSE	FALSE	2023	7	19	2024-03-01T13:03:08.432637Z
2023-07-20	24.860163%	0.015080%	4.693755%	0.004985%	0.000036%	0.015171%	0.032098%	0.007613%	TRUE	FALSE	TRUE	FALSE	2023	7	20	2024-03-01T14:03:08.432637Z
2023-07-21	0.000023%	3.115898%	0.011088%	3.522027%	20.949944%	0.015196%	3.957762%	0.007699%	FALSE	TRUE	FALSE	TRUE	2023	7	21	2024-03-01T15:03:08.432637Z

I.19 DEVICE ACTIVITY STATISTICS

DESCRIPTION	METRICS PRODUCED BY THE METHOD *DEVICE ACTIVITY STATISTICS*
Object / Unit / Record	Metrics / statistics
Contents	Description of the metrics computed for the specific device along with their values and the choice of period parameter (if needed). • **user_id:** ○ Type: Binary ○ Requirements: 32 bytes (256 bits) field. ○ Description: Unique pseudonymized identifier of the device. • **year:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Year the metrics refer to in the local timezone. • **month:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Month the metrics refer to in the local timezone. • **day:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Day the metrics refer to in the local timezone. • **event_cnt:** ○ Type: Integer 32 ○ Requirements: Integer of 32 bits. ○ Description: Number of events per day. • **unique_cell_cnt:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Number of unique cells per day. • **unique_location_cnt:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Number of different locations per day (based on the location point of the cell). • **sum_distance_m:** ○ Type: Integer 32 ○ Requirements: Integer of 32 bits. ○ Description: Sum of the distances between the events (based on the location point of the cell). • **unique_hour_cnt:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. Up to 24. ○ Description: Number of unique hours in the date with events. • **mean_time_gap:** ○ Type: Integer 32 ○ Requirements: Integer of 32 bits. ○ Description: Average time gap between events (in seconds). • **stdev_time_gap:** ○ Type: Float ○ Requirements: Float ○ Description: Standard deviation of the time gap between events (in seconds).

DESCRIPTION	METRICS PRODUCED BY THE METHOD *DEVICE ACTIVITY STATISTICS*
	Notes • All the indicators are calculated per device per day. When longer period assessment of the device activity is needed (e.g., for specific use case), then this must be done by combining the metrics for different dates that are inside the necessary period. For simplicity and optimisation reasons, this longer-period aggregates are not stored in this data object. This is also necessary due to the requirement of periodical deletion of historical device-level data that can be successfully done using the 'date' here but could not be done very well with longer periods.

\ EXAMPLE

device_id	year	month	day	event_cnt	unique_cell_cnt	unique_location_cnt	sum_distance_m	unique_hour_cnt	mean_time_gap	stdev_time_gap
A	2023	1	1	12	10	10	45778	10	5090	2951.61
A	2023	1	2	8	2	2	7592	7	5118	3169.484
B	2023	1	1	12	10	10	45036	8	4358	3614.575
C	2023	1	1	11	1	1	0	10	5939	4039.195
C	2023	1	2	20	1	1	0	14	4173	3017.242
C	2023	1	3	12	1	1	0	10	7313	3111.024
C	2023	1	4	7	1	1	0	5	4062	1536.541
D	2023	1	1	112	80	80	1035035	9	276	163.491
E	2023	1	1	142	37	37	13083	2	28	17.225
F	2023	1	1	41	1	1	0	1	33	13.647
G	2023	1	1	24	13	13	51061	24	3600	0
H	2023	1	1	1	1	1	0	1	0	0

I.20 DAILY CONTINUOUS TIME SEGMENTS

NAME	SILVERTIMESEGMENTSDATAOBJECT
Description	Daily time segments of a specific user covering the 24 hours of a specific date under study. The individual MNO data events are grouped into time segments. Four categories are supported: (i) **stay** (set of events that are close in time and space during a minimum dwell time), (ii) **unknown** (gap of information), (iii) **undetermined** (punctual events that are not possible to classify either as stay or move) and (iv) **move** (rest of the day that it is not classified as any other group).
Object/Unit/Record	Time segment
Contents	• **user_id**: o Type: Binary. o Description: Unique pseudonymized identifier of the device. • **time_segment_id**: o Type: Integer. o Description: Unique identifier of the time segment associated to a specific user. • **start_timestamp:** o Type: timestamp ('YYYY-MM-DD hh:mm:ss') in UTC standard. o Description: the date and time of the first event of the time segment. • **end_timestamp:** o Type: timestamp ('YYYY-MM-DD hh:mm:ss') in UTC standard. o Description: the date and time of the last event of the time segment. • **mcc:** o Type: Integer. o Description: Mobile Country Code derived from the user's IMSI. • **cells:** o Type: Array of strings. o Description: set of cells identifiers associated to the time segment. • **state**: o Type: String. o Description: type of time segment (stay, move, undetermine or unknown). • **is_last**: o Type: Boolean. o Description: If the time segment is the last time segment of a user in a day. • **year:** o Type: Integer 16. o Description: Year the event took place. • **month:** o Type: Integer 8. o Description: Month the event took place. • **day:** o Type: Integer 8. o Description: Day the event took place. • **user_id_modulo** o Type: Integer. o Description: Partition key.

\ EXAMPLE

time_segment_id	device_id	start_timestamp	end_timestamp	cells	state	is_last
1	1	2023-01-01 00:00:00	2023-01-01 06:45:01	[214030412038931, 214030412038932, 214030412038935, 214030412038938]	stay	false
2	1	2023-01-01 06:45:01	2023-01-01 07:16:21	[214030412038940, 214035484123541, 214035484123544]	move	false
3	1	2023-01-01 07:16:21	2023-01-01 22:16:15	null	unknown	false
4	1	2023-01-01 22:16:15	2023-01-01 23:59:59	[214030412038931, 214030412038932]	stay	true
1	2	2023-01-01 00:00:00	2023-01-01 11:49:35	[214030412038964, 214030412038965]	stay	false
…	…	…	…	…	…	…

I.21 DAILY PERMANENCE SCORE

NAME	DAILYPERMANENCESCOREDATAOBJECT
Description	Contains each user's stay time at each cell during each of the time slots of each date, calculated from events data.
Object/Unit/Record	Stay duration in seconds of a given user in a given cell and a given time slot.
Contents	**Mandatory fields**: • **user_id:** o Type: Binary o Description: Unique pseudonymized identifier of the device. • **grid_id:** o Type: String o Description: Unique ID of grid tile, or "unknown". • **time_slot_initial_time:** o Type: Timestamp o Description: Initial time of time slot. • **time_slot_end_time:** o Type: Timestamp o Description: Final time of the time slot. • **dps:** o Type: Integer 8 o Description: Daily permanence score. • **year:** o Type: Integer 16 o Description: Year the event took place. • **month:** o Type: Integer 8 o Description: Month the event took place. • **day:** o Type: Integer 8 o Description: Day the event took place. • **user_id_modulo:** o Type: Integer 16 o Description: Partition key • **id_type:** o Type: String o Description: Partition key that takes two values: grid whenever the "grid_id" field contains an actual grid ID of the INSPIRE 100x100m grid, or unknown when the "grid_id" field contains the value unknown.

\ EXAMPLE

user_id	grid_id	time_slot_initial_time	time_slot_duration	dps	year	month	day	user_id_modulo
000000000000..01	100mN4056000E5275300	10:00	3600	1	2024	01	15	511
000000000000..01	100mN4056100E5275300	10:00	3600	2	2024	01	15	511
000000000000..01	100mN4056000E5275300	11:00	3600	4	2024	01	15	511

I.22 MNO EVENT DATA QUALITY WARNINGS – FOR PLOTS

NAME	SILVEREVENTDATASYNTACTICQUALITYWARNINGSFORPLOTS
Description	The data object is meant to store the data needed for plot creation, the plots' data are differentiated in three categories • daily 'Total initial frequency' along with its average and the control limits computed based on lookback period • daily 'Total final frequency' along with its average and the control limits computed based on lookback period • 'Error rate by date' along with its average and only upper control limit computed based on lookback period In the future releases it is planned to add a report creation option.
Object/Unit/Record	Quality warnings
Contents	**Mandatory fields**: • **date:** o Type: Date o Description: date of a value and its statistics were taken/computed. • **type_of_qw:** o Type: String o Description: Indicator of what type of data is stored, could be raw_data_size, clean_data_size, error_rate. • **lookback_period:** o Type: String ▪ Description: The text representation of a lookback period, e.g. 'week' meaning 7 days • **daily_value:** o Type: Float o Description: The value of either initial frequency, final frequency, or error rate calculated on this date. • **average:** o Type: Float o Description: mean computed based on a lookback period • **LCL**: o Type: Float o Description: Low Control limit, mean - X*std computed based on a lookback period (applicable only for data size values). • **UCL**: o Type: Float o Description: Upper Control limit, mean + X*std computed based on a lookback period

\ EXAMPLE

date	type_of_qw	lookback_period	daily_value	average	LCL	UCL
2024-01-29	Error rate	week	23.41	20.2	None	22.48

I.23 MNO NETWORK SYNTACTIC QUALITY WARNINGS LINE PLOT DATA

NAME	SILVERNETWORKSYNTACTICQUALITYWARNINGSLINEPLOTDATA
Description	The data object is meant to store the data needed for line plots that show the daily evolution of the number of rows before and after the syntactic checks, as well as the overall error rate. • Number of rows before syntactic cleaning: data for a line plot containing said number of rows for the lookback period and the study date, together with the average, upper control limit, and lower control limit over the lookback period. • Number of rows after syntactic cleaning: data for a line plot containing said number of rows for the lookback period and the study date, together with the average, upper control limit, and lower control limit over the lookback period. Error rate: data for a line plot containing the error rate of the dataset (i.e., percentage of non-erroneous rows) for the lookback period and the study date, together with the average and upper control limit over the lookback period.
Object/Unit/Record	Quality Warnings
Contents	**Mandatory fields**: • **date:** o Type: Date o Description: date that the 'daily_value' in this row refers to. • **daily_value:** o Type: Float o Description: Value that the variable of this row takes in the date 'date'. • **average:** o Type: float ▪ Description: Average of the variable of this row over the lookback period. • **LCL:** o Type: Float, nullable o Description: The lower control limit of the variable of this row over the lookback period. This value is null and not recorded for the error rate. • **UCL:** o Type: Float o Description: The upper control limit of the variable of this row over the lookback period. This value is null and not recorded for the error rate. • **variable**: o Type: String o Description: Variable that the data in this row refers to. Can be one of rows_before_syntactic_check, rows_after_syntactic_check, and error_rate. Partition column. • **year**: o Type: Integer 16 o Description: year of the study date of the execution of the quality warnings component. Partition column. • **month**: o Type: Integer 8 o Description: month of the study date of the execution of the quality warnings component. Partition column. • **day**: o Type: Integer 8 o Description: day of the study date of the execution of the quality warnings component. Partition column. • **timestamp:** o Type: Timestamp

NAME	SILVERNETWORKSYNTACTICQUALITYWARNINGSLINEPLOTDATA
	○ Description: timestamp of the execution of the quality warnings component that produced this data object, serving as a execution ID. Partition column.

\ EXAMPLE

date	daily_value	average	LCL	UCL	variable	year	month	day	timestamp
2023-01-01	62.071918	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-02	61.904762	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-03	61.616955	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-04	61.681549	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-05	61.911556	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-06	61.575344	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-07	61.970898	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-08	61.460957	61.818996	62.011372	null	error_rate	2023	1	8	2024-03-14 17:57:41.958112
2023-01-01	443.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-02	464.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-03	489.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-04	515.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-05	534.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-06	561.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-07	575.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-08	612.000000	511.571442	560.655884	462.486938	rows_after_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-01	1168.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-02	1218.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-03	1274.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-04	1344.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-05	1402.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-06	1460.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112
2023-01-07	1512.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112

date	daily_value	average	LCL	UCL	variable	year	month	day	timestamp
2023-01-08	1588.000000	1339.714233	1466.644287	1212.784180	rows_before_syntactic_check	2023	1	8	2024-03-14 17:57:41.958112

I.24 MNO NETWORK SYNTACTIC QUALITY WARNINGS PIE PLOT DATA

NAME	SILVERNETWORKSYNTACTICQUALITYWARNINGSPIEPLOTDATA
Description	The data object is meant to store the data needed for pie plots that show the counts of each type of error for each of the fields of the Network Topology data.
Object/Unit/Record	Quality Warnings
Contents	**Mandatory fields**: • **type_of_error:** o Type: String o Description: name of the type of error this row refers to. • **value:** o Type: Integer o Description: Count of this type of error for the variable this row refers to. • **variable**: o Type: String o Description: Variable that the data in this row refers to. Can be one of the fields of I.8 Cell Locations with Physical Properties – Cleaned. Partition column. • **year**: o Type: Integer 16 o Description: year of the study date of the execution of the quality warnings component. Partition column. • **month**: o Type: Integer 8 o Description: month of the study date of the execution of the quality warnings component. Partition column. • **day**: o Type: Integer 8 o Description: day of the study date of the execution of the quality warnings component. Partition column. • **timestamp:** o Type: Timestamp o Description: timestamp of the execution of the quality warnings component that produced this data object, serving as a execution ID. Partition column.

\ EXAMPLE

type_of_error	value	variable	year	month	day	timestamp
NULL_VALUE	145	altitude	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	antenna_height	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	78	antenna_height	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	35	azimuth_angle	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	69	cell_id	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	72	cell_id	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	cell_type	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	358	cell_type	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	59	directionality	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	directionality	2023	1	8	2024-03-14 17:57:41.958112

type_of_error	value	variable	year	month	day	timestamp
OUT_OF_RANGE	63	elevation_angle	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	elevation_angle	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	63	frequency	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	frequency	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	horizontal_beam_width	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	62	horizontal_beam_width	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	41	latitude	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	78	latitude	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	676	longitude	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	68	longitude	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	63	power	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	power	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	technology	2023	1	8	2024-03-14 17:57:41.958112
CANNOT_PARSE	19	valid_date_end	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	689	valid_date_end	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	162	valid_date_start	2023	1	8	2024-03-14 17:57:41.958112
CANNOT_PARSE	25	valid_date_start	2023	1	8	2024-03-14 17:57:41.958112
NULL_VALUE	145	vertical_beam_width	2023	1	8	2024-03-14 17:57:41.958112
OUT_OF_RANGE	77	vertical_beam_width	2023	1	8	2024-03-14 17:57:41.958112

I.25 EVENT DATA AT DEVICE LEVEL SEMANTIC QUALITY WARNINGS BAR PLOT DATA

NAME	SILVER EVENT SEMANTIC QUALITY WARNINGS BAR PLOT DATA
Description	The object is meant to store the data needed for bar plots that show the daily evolution of the number of occurrences and percentage of each type of error flag in the semantic checks of the MNO Event Data at Device Level.
Object/Unit/Record	Quality Warnings
Contents	**Mandatory fields**: • **date:** o Type: Date o Description: date that the "daily_value" in this row refers to. • **type_of_error:** o Type: String o Description: name of the type of error this row refers to. • **value:** o Type: Float o Description: Count or percentage of this type of error for the variable this row refers to. • **variable**: o Type: String o Description: Variable that the data in this row refers to. Can be either Percentage or Number of occurrences. Partition column. • **year**: o Type: Integer 16 o Description: year of the study date of the execution of the quality warnings component. Partition column. • **month**: o Type: Integer 8 o Description: month of the study date of the execution of the quality warnings component. Partition column. • **day**: o Type: Integer 8 o Description: day of the study date of the execution of the quality warnings component. Partition column. • **timestamp:** o Type: Timestamp o Description: timestamp of the execution of the quality warnings component that produced this data object, serving as a execution ID. Partition column.

\ EXAMPLE

date	type_of_error	value	variable	year	month	day	timestamp
2023-01-03	Error 3	2.0	Number of occurrences	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	Error 4	10.0	Number of occurrences	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	No Error	11.0	Number of occurrences	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	Error 2	1.0	Number of occurrences	2023	1	3	2024-03-15 10:25:58.734726

date	type_of_error	value	variable	year	month	day	timestamp
2023-01-03	Error 1	1.0	Number of occurrences	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	Error 3	8.0	Percentage	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	Error 4	40.0	Percentage	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	No Error	44.0	Percentage	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	Error 2	4.0	Percentage	2023	1	3	2024-03-15 10:25:58.734726
2023-01-03	Error 1	4.0	Percentage	2023	1	3	2024-03-15 10:25:58.734726

I.26 MNO NETWORK TOPOLOGY TOP FREQUENT ERROS

NAME	SILVERNETWORKDATATOPFREQUENTERRORS
Description	Most frequent errors found in the MNO network topology data syntactic cleaning, together with their absolute frequency and their contribution to the total number of errors found. Data is sorted from most to least frequent error.
Object/Unit/Record	Value of the error found, together with its absolute frequency and accumulated error percentage.
Contents	**Mandatory fields**: • **result_timestamp**: o Type: Timestamp o Description: Timestamp of the start of the process when the metrics were produced. One process can generate multiple metrics. • **field_name**: o Type: String o Requirements: Either null or same as the name of a column present in input data o Description: Name of the field to which the metric refers to. Value is null if the metric refers to multiple fields. • **type_code**: o Type: Integer o Requirements: One value from the type codes (see table below). o Description: Numeric code indicating the type of the metric. See table below. • **error_value:** o Type: String o Description: erroneous value found in the field in question during syntactic cleaning. The values can be either null, if the invalid value was null or if it refers to an error concerning more than one field, or a string parsing of the erroneous value found. • **error_count:** o Type: Integer o Description: number of times that this error_value was found in the raw data. • **accumulated_percentage:** o Type: Float o Description: Accumulated percentage with respect to the total number of invalid values, accumulated from the most frequent error up to this one, included. • **year:** o Type: Integer 16 o Description: Year the event took place. • **month:** o Type: Integer 8 o Description: Month the event took place. • **day:** o Type: Integer 8 o Description: Day the event took place.

CODE TYPES

CODE	SHORT DESCRIPTION	DESCRIPTION
0	no errors	
1	value is null	
2	value is not within the set of accepted values	
3	unsupported input data type	
4	unable to parse correctly	
100	total rows at the start of method	
101	total rows at the end of method	

EXAMPLE

result_timestamp	field_name	type_code	error_value	error_count	accumulated_percentage	year	month	day
2024-01-07 10:00:00	cell_id	2	000000	400	40.0	2023	1	1
2024-01-07 10:00:00	cell_id	1	null	300	70.0	2023	1	1
2024-01-07 10:00:00	cell_id	2	123456789	200	90.0	2023	1	1
2024-01-07 10:00:00	cell_id	2	xxx123	50	95.0	2023	1	1
2024-01-07 10:00:00	cell_id	2	AVSADD	50	100.0	2023	1	1

I.27 MNO NETWORK TOPOLOGY ROW ERROR METRICS

NAME	SILVERNETWORKROWERRORMETRICS
Description	Metrics regarding the number of rows that are deleted during the syntactic cleaning process and the number of rows that had any erroneous field, be it in a mandatory one (so it is deleted) or in an optional one (it is replaced by the null value).
Object/Unit/Record	Number of rows.
Contents	**Mandatory fields**: • **result_timestamp**: o Type: Timestamp o Description: Timestamp of the start of the process when the metrics were produced. One process can generate multiple metrics. • **variable**: o Type: String o Requirements: "rows_with_some_error" or "rows_deleted". o Description: What the count of rows of this record refers to: either to rows that had any erroneous value, or rows that were deleted because of an unavoidable erroneous value in a mandatory field. • **value**: o Type: Integer o Description: Number of rows. • **year:** o Type: Integer 16 o Description: Year the event took place. • **month:** o Type: Integer 8 o Description: Month the event took place. • **day:** o Type: Integer 8 o Description: Day the event took place.

\ EXAMPLE

result_timestamp	variable	value	year	month	day
2024-01-07 10:00:00	rows_with_some_error	400	2023	1	1
2024-01-07 10:00:00	rows_deleted	300	2023	1	1
2024-01-07 10:00:00	rows_with_some_error	200	2023	1	2
2024-01-07 10:00:00	rows_deleted	50	2023	1	2

I.28 INSPIRE GRID

NAME	SILVERGRIDDATAOBJECT
Description	INSPIRE grid geometry
Object/Unit/Record	grid centroid geometry with additional information
Contents	**Mandatory fields**: • **grid_id**: ○ Type: String ○ Requirements: string following INSPIRE specification format ○ Description: Code uniquely identifying one grid tile. • **geometry:** ○ Type: Binary ○ Requirements: ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035) ○ Description: grid centroids point geometry • **quadkey:** ○ Type: String ○ Requirements: String of integers of fixed length ○ Description: Quadkey of a fixed length to which grid centroid belongs to. Used for explicit spatial partitioning

\ EXAMPLE

grid_id	geometry	quadkey
100mN4056000E5275300	POINT()	1201303
100mN4056000E5275400	POINT()	1201304

I.29 COUNTRIES

NAME	BRONZECOUNTRIESDATAOBJECT
Description	Dataset with countries polygons
Object/Unit/Record	A country polygon with additional information
Contents	**Mandatory fields**: ● **iso2**: ○ Type: String ○ Requirements: 2 capital characters string ○ Description: ISO2 code of a country ● **name**: ○ Type: Integer ○ Requirements: ○ Description: Name of a country ● **geometry:** ○ Type: Binary ○ Requirements: ■ ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035) ■ Has to be topologically valid (polygons without self-intersections, polygons of the same level don't overlap) ■ Hierarchical administrative units have to have one-to-one child to parent relationships ○ Description: polygon geometry which represents country

\ EXAMPLE

iso2	name	geometry
ES	Spain	POLYGON (24.36...
ES	Spain	POLYGON (24.37...
ES	Spain	POLYGON (24.37...

I.30 SYNTHETIC DIARIES

NAME	BRONZESYNTHETICDIARIESDATAOBJECT
Description	Contains user_id based diaries that describe the movement or stays over a period of time for a given set of users.
Object/Unit/Record	Description of a movement diary for a specific subscriber and given time interval.
Contents	**Mandatory fields**: • **year:** o Type: Integer 16 o Requirements: Integer of 16 bits. o Description: Year of the time for which to generate events for. Partition key. • **month:** o Type: Integer 8 o Requirements: Integer of 8 bits. o Description: Month of the time for which to generate events for. Partition key. • **day:** o Type: Integer 8 o Requirements: Integer of 8 bits. o Description: Day of the time for which to generate events for. Partition key. • **user_id:** o Type: Binary o Requirements: 32 bytes (256 bits) field. o Description: Unique pseudonymized identifier of the device, generated by hashing the user's IMSI using the SHA-256 function. • **activity_type:** o Type: String o Requirements: either “stay” or “move” o Description: Labels the row of either a movement or stay diary description. • **stay_type:** o Type: String o Requirement: one of “home, “work”, “other” o Description: The type of stay, signifying that the user is either in their home location, work location or some other location. • **longitude**: o Type: Float o Requirements: Longitude value in WGS84 system. Value has to be within WGS84 bounds. Optional if “cell_id” is not null. o Description: Longitude value of the location of the event. • **latitude:** o Type: Float o Requirements: Latitude value in WGS84 system. Value has to be within WGS84 bounds. Optional if “cell_id” is not null. o Description: Latitude value of the location of the event. • **initial_timestamp** o Type: String o Requirements: String with date and time following ISO:8601 format: YYYY-MM-DDThh:mm.ss o Description: Start of time for which to generate events with the current stay and activity types for the given user. • **final_timestamp** o Type: String o Requirements: String with date and time following ISO:8601 format: YYYY-MM-DDThh:mm.ss

NAME	BRONZESYNTHETICDIARIESDATAOBJECT
	○ Description: End of time for which to generate events with the current stay and activity types for the given user.

\ EXAMPLE

user_id	activity_type	stay_type	initial_timestamp	final_timestamp	longitude	latitude	year	month	day
1	stay	'home'	2024-01-01 00:00:00	2024-01-01 10:00:00	40.41740	-3.69303	2024	1	1
1	move	null	2024-01-01 10:00:00	2024-01-01 10:26:20	null	null	2024	1	1
1	stay	'work'	2024-01-01 10:26:20	2023-01-01 16:26:20	40.44566	-3.62655	2024	1	1
1	move	null	2023-01-01 16:26:20	2023-01-01 16:34:17	null	null	2024	1	1
1	stay	'other'	2023-01-01 16:34:17	2023-01-01 18:34:17	40.44325	-3.70723	2024	1	1
1	move	null	2023-01-01 18:34:17	2023-01-01 19:00:17	null	null	2024	1	1
1	stay	'home'	2023-01-01 19:00:17	2023-01-01 23:59:59	40.41740	-3.69303	2024	1	1

I.31 ENRICHED GRID

NAME	ENRICHED GRID
Description	INSPIRE grid geometry with additional information
Object/Unit/Record	Grid centroid geometry with additional information
Contents	**Mandatory fields**: • **grid_id**: o Type: String o Requirements: string following INSPIRE specification format o Description: Code uniquely identifying one grid tile. • **geometry:** o Type: Binary o Requirements: ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035) o Description: grid centroids point geometry • **elevation:** o Type: Float o Requirements: o Description: Elevation of a grid centroids • **prior_probability** o Type: float o Requirements: o Description: Prior probability value. Sum of weighted landuse ratios normalized to 1 over the whole grid • **environment_ple_coefficient** o Type: Float o Requirements: o Description: Sum of weighted landuse ratios • **quadkey:** o Type: String o Requirements: String of integers of fixed length o Description: Quadkey of a fixed length to which grid centroid belongs to. Used for explicit spatial partitioning

\ EXAMPLE

grid_id	geometry	elevation	prior_probability	environment_ple_coefficient	quadkey
100mN4056000E5275300	POINT()	12.1	0.00001	0.01	1201303
100mN4056000E5275400	POINT()	11.9	0.00034	0.05	1201304

I.32 LANDUSE

NAME	BRONZELANDUSEDATAOBJECT
Description	Dataset with landuse information.
Object/Unit/Record	Landuse polygons categorized into predefined set of classes
Contents	**Mandatory fields**: • **category**: ○ Type: String ○ Requirements: class name from the predefined list ○ Description: Name of a class which represents high-level landuse type. • **geometry:** ○ Type: Binary ○ Requirements: ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035) ○ Description: polygon geometry representing landuse class • **year:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Year of dataset extraction. • **month:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Month of dataset extraction. • **day:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Day of dataset extraction. **Optional fields**: • **quadkey:** ○ Type: String ○ Requirements: String of integers of fixed length ○ Description: Quadkey of a fixed length to which a geometry centroid belongs to. Used for explicit spatial partitioning

\ CATEGORY NAMES

CATEGORY	SHORT DESCRIPTION
residential_builtup	Built-up areas mostly occupied by residential buildings
other_builtup	Built-up areas occupied by non-residential buildings
open_area	Open areas with minimal human activities (agriculture, parks, golf fields etc)
forest	Forests
water	Water bodies and wetlands

\ EXAMPLE

category	geometry	year	month	day	quadkey
residential_builtup	POLYGON (24.36...	2024	04	01	033111001
forest	POLYGON (24.37...	2024	04	01	033111001

I.33 TRANSPORTATION

NAME	BRONZETRANSPORTATIONDATAOBJECT
Description	Dataset with roads and railroads.
Object/Unit/Record	Transportation segments categorized into predefined set of classes
Contents	**Mandatory fields**: • **category**: ○ Type: String ○ Requirements: class name from the predefined list ○ Description: Name of a class which represents hierarchy of a road. • **geometry:** ○ Type: Binary ○ Requirements: ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035) ○ Description: linestring geometry representing roads or railroads • **year:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Year of dataset extraction. • **month:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Month of dataset extraction. • **day:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Day of dataset extraction. **Optional fields**: • **quadkey:** ○ Type: String ○ Requirements: String of integers of fixed length ○ Description: Quadkey of a fixed length to which a geometry centroid belongs to. Used for explicit spatial partitioning

\ CATEGORY NAMES

CATEGORY	SHORT DESCRIPTION
primary	Major highways
secondary	Main streets in cities, towns and minor highways
tertiary	Minor streets in cities, towns, villages
pedestrian	Cycling paths, footpaths and other ways not accessible to motorized vehicles
rail	Railroads

\ EXAMPLE

category	geometry	year	month	day	quadkey
primary	LINESTRING (24.36...	2024	04	01	033111001
secondary	LINESTRING (24.37...	2024	04	01	033111001

I.34 ADMINISTRATIVE UNITS

<table>
<tr><th>NAME</th><th>BRONZEADMINUNITSDATAOBJECT</th></tr>
<tr><td><b>Description</b></td><td>Dataset with administrative units</td></tr>
<tr><td><b>Object/Unit/Record</b></td><td>An administrative unit polygon with additional information</td></tr>
<tr><td><b>Contents</b></td><td><b>Mandatory fields</b>:
<ul>
<li><b>id</b>:
<ul>
<li>Type: String</li>
<li>Requirements: has to be unique for the whole dataset</li>
<li>Description: unique identifier of an administrative unit</li>
</ul></li>
<li><b>name</b>:
<ul>
<li>Type: String</li>
<li>Requirements: English transliteration</li>
<li>Description: Name of administrative unit</li>
</ul></li>
<li><b>level</b>:
<ul>
<li>Type: Integer</li>
<li>Requirements: Starting from 0 representing the whole country. Can be null if administrative unit dataset not hierarchical</li>
<li>Description: Level of the administrative unit. Example: 0 - whole country, 1 - municipalities, 2 - districts, and so on</li>
</ul></li>
<li><b>parent_id</b>
<ul>
<li>Type: String</li>
<li>Requirements: can be null if administrative units dataset is not hierarchical</li>
<li>Description: id of the parent administrative unit</li>
</ul></li>
<li><b>counry_iso2_code</b>
<ul>
<li>Type: String</li>
<li>Requirements: 2 capital characters string</li>
<li>Description: ISO2 code of a country</li>
</ul></li>
<li><b>geometry:</b>
<ul>
<li>Type: Binary</li>
<li>Requirements:
<ul>
<li>ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035)</li>
<li>Must be topologically valid (polygons without self-intersections, polygons of the same level don't overlap)</li>
<li>Hierarchical administrative units have to have one-to-one child to parent relationships</li>
</ul></li>
<li>Description: polygon geometry which represents administrative unit</li>
</ul></li>
<li><b>dataset_id:</b>
<ul>
<li>Type: String</li>
<li>Requirements: Must be unique for each administrative units dataset</li>
<li>Description: Unique dataset identifier</li>
</ul></li>
<li><b>year:</b>
<ul>
<li>Type: Integer 16</li>
<li>Requirements: Integer of 16 bits.</li>
<li>Description: Year of the datasets used.</li>
</ul></li>
<li><b>month:</b>
<ul>
<li>Type: Integer 8</li>
<li>Requirements: Integer of 8 bits.</li>
<li>Description: Month of the datasets used.</li>
</ul></li>
<li><b>day:</b>
<ul>
<li>Type: Integer 8</li>
<li>Requirements: Integer of 8 bits.</li>
<li>Description: Day of the datasets used.</li>
</ul></li>
</ul></td></tr>
</table>

EXAMPLE

id	name	level	parent_id	counry_iso2_code	geometry	dataset_id	year	month	day
01	Estonia	0	null	EE	POLYGON (24.36...	ETAK	2024	04	01
0103	Tartu Maakond	1	01	EE	POLYGON (24.37...	ETAK	2024	04	01
010302	Tartu Vald	2	0103	EE	POLYGON (24.37...	ETAK	2024	04	01

I.35 GEOGRAPHIC ZONES

NAME	BRONZEGEOGRAPHICZONESDATAOBJECT
Description	Dataset with geographical zones. Can be any geographic divisions.
Object/Unit/Record	A geographic zone polygon with additional information
Contents	**Mandatory fields**: • **zone_id**: ○ Type: String ○ Requirements: must be unique for the whole dataset ○ Description: unique identifier of a zone • **name**: ○ Type: String ○ Requirements: English transliteration ○ Description: Name of administrative unit • **level**: ○ Type: Integer ○ Requirements: Starting from 0 representing the whole country. Can be null if zoning dataset is not hierarchical ○ Description: Level of the administrative unit. Example: 0 - whole country, 1 - municipalities, 2 - districts, and so on • **parent_id** ○ Type: String ○ Requirements: can be null if administrative units dataset is not hierarchical ○ Description: id of the parent administrative unit • **iso2** ○ Type: String ○ Requirements: 2 capital characters string ○ Description: ISO2 code of a country • **geometry:** ○ Type: Binary ○ Requirements: ▪ ETRS89 Lambert Azimuthal Equal Area coordinate reference system (EPSG:3035) ▪ Has to be topologically valid (polygons without self-intersections, polygons of the same level don't overlap) ▪ Hierarchical administrative units have to have one-to-one child to parent relationships ○ Description: polygon geometry which represents zonning unit • **dataset_id:** ○ Type: String ○ Requirements: Has to be unique for each geographic zones dataset ○ Description: Unique dataset identifier • **year:** ○ Type: Integer 16 ○ Requirements: Integer of 16 bits. ○ Description: Year of the datasets used. • **month:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Month of the datasets used. • **day:** ○ Type: Integer 8 ○ Requirements: Integer of 8 bits. ○ Description: Day of the datasets used.

\ EXAMPLE

id	name	level	parent_id	iso2	geometry	dataset_id	year	month	day
ES53	Illes Balears	2	ES5	ES	POLYGON (24.36...	nuts	2024	04	01
ES5	Este	1	ES	ES	POLYGON (24.37...	nuts	2024	04	01
ES532	Mallorca	3	ES53	ES	POLYGON (24.37...	nuts	2024	04	01

I.36 ZONES – GRID MAP

NAME	SILVERGEOZONESGRIDMAPDATAOBJECT
Description	Dataset with geographical zones ids to grid ids mapping
Object/Unit/Record	zoning unit id to grid id map
Contents	**Mandatory fields**: • **grid_id**: o Type: String o Requirements: String following INSPIRE specification o Description: Code uniquely identifying one grid tile • **zone_id**: o Type: String o Requirements: Unique identifier. If dataset hierarchical has to be the lowest level o Description: Unique identifier of a zoning unit. • **hierarchical_id**: o Type: String o Requirements: Combination of identifiers in hierarchy. Each zone level id separated by |. o Description: Unique identifiers of a zoning units in hierarchy. If zoning dataset hierarchical, id will be combined from ids of all zones in a hierarchy, each level id separated by |. If zoning dataset is not hierarchical, id will be same as zone_id • **dataset_id:** o Type: String o Requirements: Has to be unique for each zoning dataset o Description: Unique dataset identifier • **year:** o Type: Integer 16 o Requirements: Integer of 16 bits. o Description: Year of the mapped dataset • **month:** o Type: Integer 8 o Requirements: Integer of 8 bits. o Description: Month of the mapped dataset. • **day:** o Type: Integer 8 o Requirements: Integer of 8 bits. o Description: Day of the mapped dataset.

\ EXAMPLE

grid_id	zone_id	hierarchical_id	dataset_id	year	month	day
100mN4056000E5275300	ES532	ES5|ES53|ES532	nuts	2024	4	1
100mN4056000E5275400	ES532	ES5|ES53|ES532	nuts	2024	4	1
100mN4056000E5275500	ES532	ES5|ES53|ES532	nuts	2024	4	1

I.37 UE LABELS

NAME	UE LABELS
Description	Grid tiles that have been determined to be part of the usual environment of a given device / user with meaningful location labeling
Object/Unit/Record	Usual Environment grid tile.
Contents	**Mandatory fields:** ● **user_id:** ○ Type: Binary ○ Description: Unique pseudonymized identifier of the device. ● **grid_id:** ○ Type: String ○ Description: ID of grid tile ● **label:** ○ Type: String ○ Description: Label that has been inferred for this usual environment. Options are: home, second_home, work, and no_label. ● **ue_label_rule:** ○ Type: String ○ Description: Code of a rule based on which UE label was assigned. **Has to be predefined list of rules with unique codes.** ● **location_label_rule:** ○ Type: String ○ Description: Code of a rule based on which meaningful location label was assigned. **Has to be predefined list of rules with unique codes.** ● **start_date** ○ Type: date ○ Description: Start date (inclusive) of the label period. Partition key. ● **end_date** ○ Type: date ○ Description: End date (inclusive) of the label period. Partition key. ● **user_id_modulo:** ○ Type: Integer 16 ○ Description: Partition key

UE LABEL RULE CODES

CODE	RULE DESCRIPTION
ue_1	If the device was observed all days: all intervals period in top tiles at least ue_ps_threshold (default is 70% of total_assigned_ps) such tiles are labeled as UE tiles.
ue_2	For tiles which have not got UE label in previous step perform the same check for all other combinations of day types and periods. If condition is met for any of the combinations, label tiles as UE tiles.
ue_na	No label UE label assigned

LOCATION LABEL RULE CODES

CODE	RULE DESCRIPTION
h_1	If the device was observed in all days: all intervals period in top tiles at least home_ps_threshold (default is 80% of total_assigned_ps). Such tiles are labeled as Home tiles.
h_2	If no home label being assigned, repeat this condition check for all days: night-time period. Tiles that fulfilled this condition are labeled as Home tiles.

CODE	RULE DESCRIPTION
h_3	If no home label being assigned, check if the device was in the tiles at least home_ndays_threshold (default value is 80% of total_observed_days). Tiles that fulfilled this condition are labeled as Home tiles.
w_1	If the device was observed in working_days: daytime period in top tiles at least work_ps_threshold (default is 70% of total_assigned_ps). Such tiles are labeled as Work tiles.
w_2	If no work label being assigned, check if the device in working_days: daytime period was in the tiles at least work_ndays_threshold (default value is 70% of total_observed_days). Tiles that fulfilled this condition are labeled as Work tiles.
loc_na	No location label assigned

EXAMPLE

user_id	grid_id	label	ue_label_rule	location_label_rule	end_date	end_date	user_id_modulo
000000000000..01	100mN4056000E5275300	home	ue_1	h_1	2024-02-01	2024-07-31	1
000000000000..01	100mN4056000E5275301	home	ue_2	h_1	2024-02-01	2024-07-31	1
000000000000..01	100mN4056050E5275300	work	ue_1	w_2	2024-02-01	2024-07-31	1
000000000000..01	100mN4056030E5275390	no_label	ue_1	na	2024-02-01	2024-07-31	1

I.38 MID-TERM PERMANENCE METRICS

NAME	MID-TERM PERMANENCE METRICS
Description	Mid-term permanence score for grid tiles over a predefined subset of day types (sub-monthly time intervals) (weekdays, weekends, holidays) and a predefined subset of sub-daily time intervals (nighttime, daytime, working hours, …)
Object/Unit/Record	Mid-term permanence metrics per user_id, grid_id, sub_monthly and sub_daily period.
Contents	**Mandatory fields:** • **user_id:** o Type: Binary o Description: Unique pseudonymized identifier of the device. • **grid_id:** o Type: String o Description: Unique ID of grid tile. Takes a valid grid ID value whenever "id_type" is equal to grid; and values unknown or device_observation when the indicators refer to the unknown location or the global device observation, respectively. • **mps**: o Type: Integer o Description: midterm permanence score, the result of adding up the daily permanence scores of this grid_id and user_id over the specified day_type and time_interval. • **frequency**: o Type: Integer o Description: absolute count of the number of days of this day_type for which the daily permanence score was not null in the specified time_interval. • **regularity_mean**: o Type: Float o Description: mean of the number of days between two consecutive non-null daily permanence scores in the specified day_type and time_interval. • **regularity_std:** o Type: Float o Description: standard deviation of the number of days between two consecutive non-null daily permanence scores in the specified day_type and time_interval. • **day_type**: o Type: String o Requirements: A value from predefined set: all, workdays, weekends, holidays, Mondays, Tuesdays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays. o Description: Name of the type of days over which the midterm permanence metrics are computed. Partition key. • **time_interval:** o Type: String o Requirements: A value from predefined set: all, working_hours, night_time, evening_time. o Description: Name of the sub-daily interval over which the midterm permanence metrics are computed. Partition key. • **year:** o Type: Integer 16 o Description: year of the month for which this midterm permanence score was computed. Partition key. • **month:** o Type: Integer 8 o Description: month for which this midterm permanence score was computed. Partition key.

NAME	MID-TERM PERMANENCE METRICS
	- **user_id_modulo:** ○ Type: Integer ○ Requirements: Integer of 8 bits. ○ Description: Modulo division result, as applied to the integer part of the user_id column. Partition key. - **id_type:** ○ Type: String ○ Description: Partition key that takes one of three values: grid whenever the “grid_id” field contains an actual grid ID of the INSPIRE 100x100m grid; unknown when the "grid_id" field contains the value unknown; or device_observation when the “grid_id” field contains the value device_observation.

\ EXAMPLE

user_id	grid_id	day_type	time_interval	mps	frequency	regularity_mean	regularity_std	year	month	user_id_modulo	id_type
000000000000..01	100mN4056000E5275300	all	all	896	26	1.071428571428571	1	2024	2	23	grid
000000000000..01	100mN4056000E5275300	weekend	all	256	7	5.45	3.123213	2024	2	23	grid
000000000000..01	100mN4056000E5275300	all	night_time	880	24	1.142857142857143	1	2024	2	23	grid
000000000000..01	100mN4056000E5275300	weekend	night_time	233	7	6.5	4.17347	2024	2	23	grid

I.39 LONG-TERM PERMANENCE METRICS

NAME	LONG-TERM PERMANENCE METRICS
Description	Long-term permanence score for grid tiles over a predefined subset of seasons (sub-yearly time intervals: spring, summer…), day types (sub-monthly time intervals: weekdays, weekends, holidays) and a predefined subset of sub-daily time intervals (nighttime, daytime, working hours, …)
Object/Unit/Record	Long-term permanence metrics per user_id, grid_id, sub_yearly, sub_monthly and sub_daily period.
Contents	**Mandatory fields:** ● **user_id:** ○ Type: Binary ○ Description: Unique pseudonymized identifier of the device. ● **grid_id:** ○ Type: String ○ Description: Unique ID of grid tile ● **lps**: ○ Type: Integer ○ Description: long term permanence score, the result of adding up the monthly permanence scores of this grid_id and user_id over the specified season, day_type and time_interval. ● **total_frequency**: ○ Type: Integer ○ Description: absolute count of the number of days of this day_type during specified season for which the monthly permanence score was not null in the specified time_interval. ● **frequency_mean** ○ Type: Float ○ Description: mean of monthly frequency of this day_type during specified season for the specified time_interval. ● **frequency_std** ○ Type: Float ○ Description: standard deviation of monthly frequency of day_type during specified season for the specified time_interval. ● **regularity_mean**: ○ Type: Float ○ Description: mean of the monthly regularity_mean in the specified season for day_type and time_interval. ● **regularity_std:** ○ Type: Float ○ Description: standard deviation of the monthly regularity_mean in the specified season for day_type and time_interval. ● **season** ○ Type: String ○ Requirements: A value from predefined set: all, summer, autumn, winter, spring ○ Description: Name of the type of season over which the long-term permanence metrics are computed. Partition key. ● **day_type**: ○ Type: String ○ Requirements: A value from predefined set: all, workdays, weekends, holidays, Mondays, Tuesdays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays. ○ Description: Name of the type of days over which the long-term permanence metrics are computed. Partition key. ● **time_interval:** ○ Type: String

NAME	LONG-TERM PERMANENCE METRICS
	- Requirements: A value from predefined set: all, working_hours, night_time, evening_time. - Description: Name of the sub-daily interval over which the long-term permanence metrics are computed. Partition key.

- **start_date**
  - Type: date
  - Description: Start date (inclusive) of the period for which long-term permanence metrics were computed. Partition key.
- **end_date**
  - Type: date
  - Description: End date (inclusive) of the period for which long-term permanence metrics were computed. Partition key.
- **user_id_modulo:**
  - Type: Integer
  - Requirements: Integer of 8 bits.
  - Description: Modulo division result, as applied to the integer part of the user_id column. Partition key.
- **id_type:**
  - Type: String
  - Description: Partition key that takes one of three values: grid whenever the “grid_id” field contains an actual grid ID of the INSPIRE 100x100m grid; unknown when the "grid_id" field contains the value unknown; or device_observation when the “grid_id” field contains the value device_observation.

\ EXAMPLE

user_id	grid_id	season	day_type	time_interval	lps	total_frequency	frequency_mean	frequency_std	regularity_mean	regularity_std	start_date	end_date	user_id_modulo	id_type
000000000000..01	100mN4056000E5275300	all	all	all	896	26	20	8	1.0714285714285771	1	2024-02-01	2024-07-31	10	grid
000000000000..01	100mN4056000E5275300	summer	weekends	all	256	7	7	1	??	??	2024-02-01	2024-07-31	10	grid
000000000000..01	100mN4056000E5275300	winter	all	night_time	880	24	18	2	1.1428571428571443	1	2024-02-01	2024-07-31	10	grid
000000000000..01	100mN4056000E5275300	summer	weekends	night_time	233	7	6	1	??	??	2024-02-01	2024-07-31	10	grid

I.40 HOLIDAY DATES CALENDAR

NAME	HOLIDAY DATES CALENDAR
Description	Contains a row of type Date for each day that is a holiday for a country and its name.
Object/Unit/Record	Date (a date that is considered a holiday).
Contents	**Mandatory fields**: • **iso_a2:** o Type: String o Description: ISO A2 country code. • **date:** o Type: Date o Description: Date that is a holiday. • **name:** o Type: String o Description: Holiday name.

\ EXAMPLE

iso_a2	date	name
IT	2024-05-16	Holiday name
ES	2024-12-24	Holiday name

I.41 PRESENT POPULATION ZONE LEVEL

NAME	PRESENTPOPULATIONZONEDATAOBJECT
Description	Estimation of the population present at a given time at the level of some zoning system.
Object/Unit/Record	Number of people present in a given zone at a given time
Contents	**Mandatory fields**: ● **zone_id:** ○ Type: String ○ Description: Unique ID of the zone ● **population:** ○ Type: Float ○ Description: Number of estimated present population for this grid tile at this time of day. ● **timestamp:** ○ Type: Time ○ Description: Time for which the present population is estimated. ● **year:** ○ Type: Integer 16 ○ Description: Year of the present population estimation. ● **month:** ○ Type: Integer 8 ○ Description: Month of the present population estimation. ● **day:** ○ Type: Integer 8 ○ Description: Day of the present population estimation.

\ EXAMPLE

zone	population	timestamp	year	month	day
CityX_DistrictA	1232131.3	12:05:03	2024	01	01
CityX_DistrictB	65645.0	12:05:03	2024	01	01
CityX_DistrictC	628357.4	12:05:03	2024	01	01

I.42 PRESENT POPULATION

NAME	PRESENTPOPULATIONDATAOBJECT
Description	Estimation of the population present at a given time at the grid tile level.
Object/Unit/Record	Number of people present in a given tile at a given time
Contents	**Mandatory fields**: • **grid_id:** o Type: String o Description: Unique ID of grid tile • **population:** o Type: Float o Description: Number of estimated present population for this grid tile at this time of day. • **timestamp:** o Type: Time o Description: Time for which the present population is estimated. • **year:** o Type: Integer 16 o Description: Year of the present population estimation. • **month:** o Type: Integer 8 o Description: Month of the present population estimation. • **day:** o Type: Integer 8 o Description: Day of the present population estimation.

\ EXAMPLE

grid_id	population	timestamp	year	month	day
100mN4056000E5275300	156.3	12:05:03	2024	01	01
100mN4056000E5275301	2.3	12:05:03	2024	01	01
100mN4056000E5275302	123.4	12:05:03	2024	01	01

I.43 LABELING QUALITY METRICS

NAME	LABEL QUALITY METRICS
Description	Quality metrics for UE and meaningful locations labeling process
Object/Unit/Record	A metric, number of devices/tiles and time period
Contents	**Mandatory fields**: • **metric** ○ Type: String ○ Requirements: A string value from predefine list of metrics. ○ Description: Metric name. • **count** ○ Type: Integer ○ Description: Counts of devices/labels assigned. • **start_date** ○ Type: date ○ Description: Start date (inclusive) of the label period. Partition key. • **end_date** ○ Type: date ○ Description: End date (inclusive) of the label period. Partition key.

\ POSSIBLE METRICS

CODE	DESCRIPTION
ue_1_rule	Number of tiles with assigned labels based on ue_1 rule
ue_2_rule	Number of tiles with assigned labels based on ue_2 rule
h_1_rule	Number of tiles with assigned labels based on h_1 rule
h_2_rule	Number of tiles with assigned labels based on h_2 rule
h_3_rule	Number of tiles with assigned labels based on h_3 rule
w_1_rule	Number of tiles with assigned labels based on w_1 rule
w_2_rule	Number of tiles with assigned labels based on w_2 rule
ue_na_rule	Number of tiles without UE label assigned
loc_na_rule	Number of tiles without any location label assigned
h_non_ue	Number of tiles which are labeled as home, but are not part of UE
w_non_ue	Number of tiles which are labeled as work, but are not part of UE
device_filter_1_rule	Number of devices which were filtered out as rarely observed based on device_fitler_1 rule
device_filter_2_rule	Number of devices which were filtered out as rarely observed based on device_fitler_2 rule

\ EXAMPLE

metric	count	start_date	end_date
device_filter_1_rule	5000	2023-06-01	2023-11-31
ue_1_rule	1000	2023-06-01	2023-11-31

I.44 AGGREGATED USUAL ENVIRONMENTS

NAME	AGGREGATEDUSUALENVIRONMENTS
Description	Number of weighed devices that have usual environments in grid tiles
Object/Unit/Record	Weighed device counts that have usual environment per grid tile.
Contents	**Mandatory fields**: • **grid_id** o Type: String o Description: Unique ID of grid tile. • **weighted_device_count** o Type: Float o Description: Count of weighed devices with usual environment assigned to this grid tile. • **label** o Type: String o Description: type of aggregate - ue, home, work. Partition key. • **start_date** o Type: date o Description: Start date (inclusive) of the period for which the usual environment was computed. Partition key. • **end_date** o Type: date o Description: End date (inclusive) of the period for which the usual environment was computed. Partition key. • **season** o Type: String o Description: season of the period for which the usual environment was computed. Partition key.

\ EXAMPLE

grid_id	weighted_device_count	label	start_date	end_date	season
100mN4052000E5271300	50.95	ue	2024-01-01	2024-06-30	winter
100mN4056500E5270500	120.33	ue	2024-01-01	2024-06-30	winter
100mN4053100E5275200	60.09	home	2024-01-01	2024-06-30	winter
100mN4056400E5274400	20.65	work	2024-01-01	2024-06-30	winter
…	…	…	…	…	…

ANNEX II – NOTES FOR FUTURE REVISION

The next version of **D4.2** (version v2 - planned for October 2024) will include:

- Technical documentation of the pending software modules for the generation of the output indicators for *Present Population* and *Usual Environment* use cases (i.e. aggregation, k-anonimity and merging of different MNOs at aggregated level).
- For *Present Population* and *Usual Environment* use cases, a complete pipeline schema of the process representing the orchestation of different software modules and the temporal scale of the calculation (i.e. daily, mid-term, long-term, etc.).
- Possible modifications based on the re-factoring analysis being performed in a local laboratory before the deployment in the MNO premises.
- In general terms, D4.2 (v2) will contain the software status before the first testing phase in MNO premises.

The following version of **D4.3** (version v1 - planned for December 2024) will include:

- Technical documentation of new software modules (at least, the ones needed to cover a total of 6 UCs).
- Quality metrics for the different software modules.
- Evaluation of incorporting intra-day cell footprint variations.
- Functionality to stop the execution based on quality warning rules.