SMIS DATABASE ARCHITECTURE TECHNICAL DOCUMENTATION

V1.0

TABLE OF CONTENTS

REVISION HISTORY

Version	Description	Date	Author
1.0	Original database architecture description. Documentation is largely exhaustive for grower data and literature data and reflects work done. For experimental data, the documentation remains a general development specification in this version.	22/08/2017	Tomasz Kurowski

ABBREVIATIONS

API	Application Programming Interface
DBMS	Database Management System
HTTP(S)	Hypertext Transfer Protocol (Secure)
JSON	JavaScript Object Notation
MEAN	MongoDB, Express.js, AngularJS, and Node.js
NG	National Grid
OS	Ordnance Survey
REST	Representational State Transfer
SMIS	Soil Management Information System
SubVESS	Subsoil Visual Evaluation of Soil Structure
VESS	Visual Evaluation of Soil Structure
VSA	Visual Soil Assessment
XML	Extensible Markup Language

1. INTRODUCTION

1.1. PURPOSE

The following document describes the design of the database back-end, which forms a vital element of the Analytics Toolkit developed as part of the Soil Management Information System (SMIS) project. The purpose of the document is to provide an overview of the design, intended to serve as an implementation guide for the developer and as an accurate description of the technical details of the system accessible to the end user.

It should be noted that as SMIS remains in active development, this document will be updated as the implementation progresses until the hand-over of the system (November 2018) in order to ensure its thoroughness and accuracy.

1.2. SCOPE

The collection, storage, manipulation and interrogation of information obtained from diverse data sources related to the effects of soil management practices on horticultural crop productivity and environmental protection are central objectives of the SMIS project. Therefore, the specific database management solutions to be used are of primary importance to the project's success.

The document presents the specific database design decisions taken, alongside the technical considerations which guided them. In particular, these include how and by whom the system is to be managed and maintained, how the database will be populated throughout the course of this project, and how the data will be accessed.

Thus, alongside a technical description of the database implementation in the narrow sense, the document also discusses the system context. This includes both the immediate “upstream” interface of the database, that is the types of data used and the ways in which they are parsed (i.e. processed and inserted into the database), as well as the immediate “downstream” interface of the database, that is the Application Programming Interface (API) which will provide the means of interrogating and manipulating the database by the SMIS Analytics Toolkit.

Details of the data collection and SMIS Analytics Toolkit functionality are not discussed here, except in how they directly impact the designs of the parsing tools and API respectively.

2. SYSTEM OVERVIEW

The SMIS database back-end uses the MongoDB database management system (DBMS) and forms a part of a larger structure. It is responsible for storing and allowing the use of three primary types of data:

a) Grower data

b) Experimental data

c) Literature data.

The end users interact with the data exclusively through a web application (the SMIS Analytics Toolkit) which in turn accesses the database through a representational state transfer (REST) API implemented using Mongoose, Express.js and Node.js.

The database is populated with curated data, processed by parsing scripts implemented in Python. For performance reasons, these scripts access the database directly (using the *pymongo* library) rather than through the REST API. The scripts are considered part of the SMIS system although they are not accessible through the web application. This is discussed more in-depth in section 3.1.

As adding new datasets is intended to be the task of a curator / administrator rather than a standard web application functionality, the data volatility is assumed to be low for most use cases.

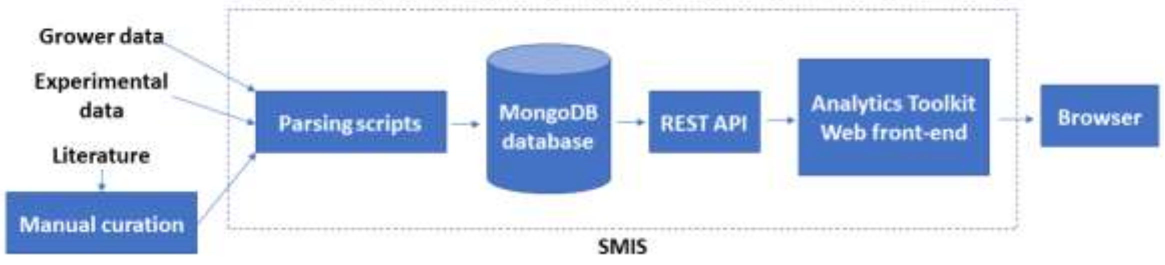


FIGURE 1: INFORMATION FLOW IN SMIS

SMIS is intended for internal, local use by an organisation (i.e AHDB) rather than Internet-wide public access. This means the traffic and demand on concurrent access are assumed to be relatively low.

2.1. CHOICE OF DBMS

MongoDB 3.4 was chosen as the database management system (DBMS) used in the SMIS back-end. The primary reason for this choice over a relational database was the schema-less nature of MongoDB. This is a crucial advantage as SMIS integrates data from disparate sources and the number of supported data formats is intended to be expandable beyond those represented among the datasets available during development. Any added formats may include new data fields and types of data, which are trivial to add to a document-oriented database like MongoDB. In a relational database expanding the capabilities in an equivalent manner could require significant changes to the database schema and possibly the rest of the application.

Relational databases hold a significant advantage in how they strictly enforce data integrity and validity. However, in SMIS the main threat to data integrity comes from inconsistencies caused by human error (e.g. typos) or differences in data entry practices between data providers (grower groups). This shifts the onus of ensuring data integrity and validity to the parsing stage, outside the control of the database management system, and largely negates the advantage of relational databases.

An additional advantage of MongoDB is its ease of integration with web applications through its inclusion as part of the MEAN stack, a popular bundle of technologies combining the database management system with the Express.js and AngularJS frameworks running on a Node.js server.

2.2. SYSTEM REQUIREMENTS

- Dedicated[1] PC running a modern Linux, macOS, or Windows Operating System
- At least 10 GiB of storage space
- Network access (HTTP/HTTPS port)[2]

While the SMIS development environment is Unix-based, all the technologies used are also available on Windows. The application will be tested in that environment to ensure it remains fully functional. System-specific installation scripts will be prepared. Alternatively, a containerised/virtualised solution, such as a Virtual Machine including a working system, could also be provided.

The database back-end in itself does not require a network connection, as it is only intended to be accessed by the SMIS Analysis Toolkit through a local port (on some systems this may require

[1] As the SMIS Analytics Toolkit is meant to be accessed over a local network, the server machine must run continuously.

[2] This requirement concerns the SMIS Analytics Toolkit as a whole rather than the database back-end itself.

modifying firewall settings). However, the SMIS web application which depends on the database does require network access so that it can be accessed from other machines.

The currently gathered data (as stored in the largely unnormalised and highly indexed database) take up less than 5 GiB of storage space and are not expected to grow much larger than that threshold before the end of the project (November 2018). Storage is therefore unlikely to be a problem, although further data gathering beyond the end of the project could raise the data volume to arbitrarily large sizes.

While traffic requirements are assumed to be relatively low, the technologies used are known to scale well. A series of full-scale performance tests will be carried out before the end of the project. These will allow for an accurate assessment of the scalability and requirements of the SMIS system if the database was to grow much larger or if the traffic was to increase significantly.

2.3. GROWER DATA

GATEKEEPER FORMAT

GateKeeper is an agronomic record-keeping solution developed by Farmplan and used by some growers in the United Kingdom. All grower datasets used in the development of SMIS are GateKeeper datasets, manually exported from the application according to export instructions provided to growers (see accompanying document: Protocol for extracting data from GateKeeper).

The default data export format used by GateKeeper is a “GateKeeper XML” file, though the same data can also be saved as an Excel spreadsheet. The two formats are in fact largely equivalent, as the XML output is a “flat” XML file, with unnormalised data stored between non-nesting tags representing individual rows of a spreadsheet.

GATEKEEPER SYNCHRONISATION

Besides its use as a stand-alone desktop application, GateKeeper also supports a client-server paradigm in which a user can synchronise the contents of their local copy of the application with a centralised server, or even share it with others (e.g. an agronomist) by providing a special access key.

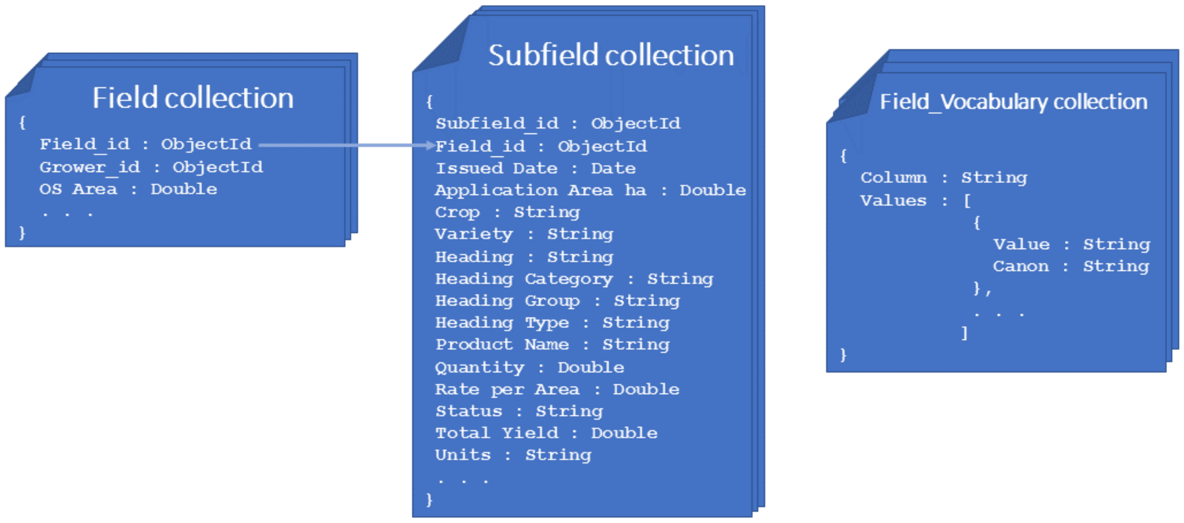
While this suggests the possibility of directly integrating SMIS with GateKeeper without the need for intermediary XML / spreadsheet files, this was not attempted, as no public API for emulating this client-server integration is available. Additionally, such close integration with a proprietary software product would be subject to obsolescence as the product changes, presenting a difficult maintenance challenge.

OTHER FORMATS

Agronomic record-keeping solutions other than GateKeeper (such as Muddy Boots) also exist and maintain a significant share of users among growers, but they are not represented among the datasets used during SMIS development and initial testing as all growers' data collected to date are in GateKeeper format. As SMIS is intended to be extendible and capable of being adapted to handle disparate sources of grower data, avoiding overly tight integration with the GateKeeper format and its conventions is among the objectives of the back-end design.

SCHEMA

Information derived from grower datasets is stored across three collections as shown in Figure 2.



```
{
  Field_id : ObjectId
  Grower_id : ObjectId
  OS Area : Double
  . . .
}
```

```
{
 Subfield_id : ObjectId
 Field_id : ObjectId
 Issued Date : Date
 Application Area ha : Double
 Crop : String
 Variety : String
 Heading : String
 Heading Category : String
 Heading Group : String
 Heading Type : String
 Product Name : String
 Quantity : Double
 Rate per Area : Double
 Status : String
 Total Yield : Double
 Units : String
 . . .
}
```

```
{
  Column : String
  Values : [
            {
              Value : String
              Canon : String
            },
            . . .
          ]
}
```

FIGURE 2: GROWER DATA COLLECTIONS DIAGRAM

The **Field** collection stores farm field–level information such as a farm field's unique identifier; the unique identifier associated with the grower who supplied the dataset; the Ordnance Survey (OS) area of the farm field; and potentially field data originating from external databases. It should be noted that grower data in SMIS is subject to anonymisation as described in section 3.1.2, so both the farm field and grower identifiers are generated at the time of parsing by (respectively) hashing and tokenisation.

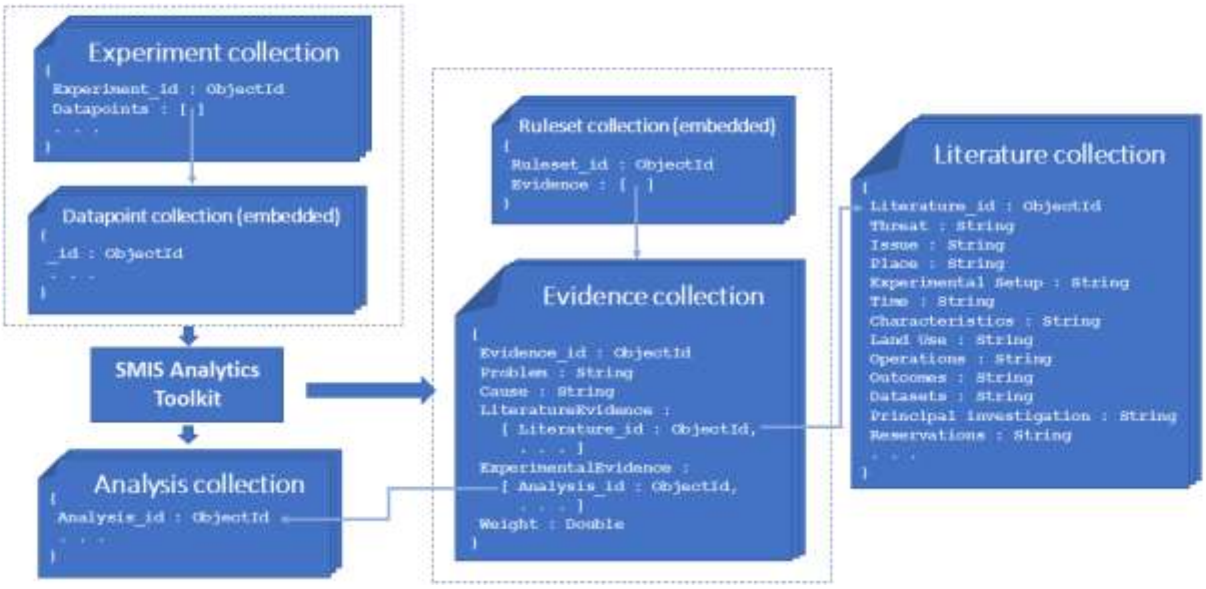
The **Subfield** collection stores information about any farm field operations, cropping, yields associated with a specific area within a farm field identified by its unique identifier (corresponding to that stored in the Field collection). This is a necessary construct as sometimes farm field operations and crop rotations are not applied to entire farm fields (as defined in grower datasets), but to smaller subdivisions. An explanation of how this information is captured is provided in section 3.1.2. The

Subfield collection is indexed on every single data field to enable for efficient execution of complex query and server-side pagination of the results for the intended use downstream (i.e. in the SMIS Analytics Toolkit).

The third collection derived from grower datasets is the **Field_Vocabulary** collection, which serves an auxiliary function in parsing grower datasets as described in section 3.1.2. It contains updateable arrays of "known" values for each data field (column) in the Subfield collection, which make it possible to detect unexpected (erroneous or novel) values in datasets during supervised parsing. Those known values are paired with "Canon" values, allowing for normalisation (or canonicalization) of data field values.

2.4. EXPERIMENTAL DATA

The primary experimental dataset used during SMIS development came from the "CP107c – Soils Programme: The application of precision farming technologies to drive sustainable intensification in horticulture cropping systems" project. The data are collected in an Excel spreadsheet, with the results of different soil structural analyses spread over multiple sheets (one per analysis type) and identifiable by shared farm field identifiers and data collection dates, in effect partially normalised.



```
{
Experiment_id : ObjectId
Datapoints : [ ]
. . .
}
```

```
{
_id : ObjectId
. . .
}
```

```
{
Analysis_id : ObjectId
. . .
}
```

```
{
Ruleset_id : ObjectId
Evidence : [ ]
}
```

```
{
Evidence_id : ObjectId
Problem : String
Cause : String
LiteratureEvidence :
  [ Literature_id : ObjectId,
    . . . ]
ExperimentalEvidence :
  [ Analysis_id : ObjectId,
    . . . ]
Weight : Double
}
```

```
{
Literature_id : ObjectId
Threat : String
Issue : String
Place : String
Experimental Setup : String
Time : String
Characteristics : String
Land Use : String
Operations : String
Outcomes : String
Datasets : String
Principal investigation : String
Reservations : String
. . .
}
```

FIGURE 3: DIAGRAM OF THE DATABASE SCHEMA AND RELATIONSHIPS FOR EXPERIMENTAL, LITERATURE, ANALYSIS AND EVIDENCE DATA

A general overview of the Experiment collection schema is shown in Figure 3. Experimental datasets are uniquely identified by a generated token and contain an embedded collection of schema-less data points which store measurements and other metadata. Each experiment entry may also be annotated with metadata. This very generic structure contains few constraints and thus allows for future inclusion of experimental datasets other than the ones used in development.

Within SMIS, all the CP107c data are grouped in a single experimental dataset. Results from different types of analyses are stored together in the embedded Datapoint collection, grouped (denormalised) by the farm field and date of analysis. The types of results stored include:

- Soil bulk density measurements
- Soil penetrometer scores
- Subsoil Visual Evaluation of Soil Structure (SubVESS) scores
- Topsoil analysis
- Visual Evaluation of Soil Structure (VESS) scores
- Visual Soil Assessment (VSA) scores

The interpretation of experimental datasets is undertaken in the SMIS Analytics Toolkit downstream.

2.5. LITERATURE DATA

The literature data used in the SMIS project was manually curated based on available sources to establish connections between soil degradation threats, soil management practices and other factors, which together would comprise a unique and useful overview of the current state of the art on those relationships. Summaries of the literature sources were manually generated and are stored in spreadsheets, which are then imported into the SMIS database and viewable in the Analytics Toolkit. The specific data fields for each item of data are listed in the "Literature collection" part of the schema diagram in Figure 3.

2.6. ANALYSIS DATA

The Analysis collection is a simple schema-less document collection used to store the results of analyses and visualisations generated through the SMIS Analysis Toolkit web application. As the results of these analyses and visualisations can be very disparate and the specific analysis use cases and tools are still under development, the collection remains schema-less other than associating a title and unique identifier with the arbitrary analysis output and possible metadata.

2.7. EVIDENCE DATA

The Evidence dataset is a derivative collection based on Literature and Experiment data (via Analysis results). Rows of the collection represent "rule bases" linking causes and effects identified either by manual literature curation (see section 2.5) or by the results of analyses based on experimental or grower data (see section 2.6).

3. SYSTEM CONTEXT

3.1. DATA PARSING

This section is intended to provide a general overview of the SMIS functionalities immediately upstream of the database, i.e. the parsing pipelines which populate the MongoDB collections based on the datasets gathered during the project. The pipelines are all implemented in Python and rely on the Mongoose API to connect to the MongoDB database, as this allows for more efficient bulk insertion of data than what could be achieved using the REST API used downstream by the SMIS Analysis Toolkit to access the database contents.

3.1.1. VOCABULARY NORMALISATION

As noted in section 2.1, inconsistencies in the input datasets used by SMIS are a major issue which needs to be addressed at the parsing stage. This is particularly true for the Grower data, which is generally entered manually and managed by individual growers. Inconsistencies can stem from human error (e.g. typos) or from data entry conventions (e.g. particular nomenclature or abbreviations) which may be fully consistent within a single organisation, but often vary between them. For example, one grower dataset might use the term “Vining Peas” while another uses “V. Peas” to refer to the same crop. For SMIS to interpret the data correctly, these conventions need to be reconciled. The same problems can also be encountered in the experimental datasets.

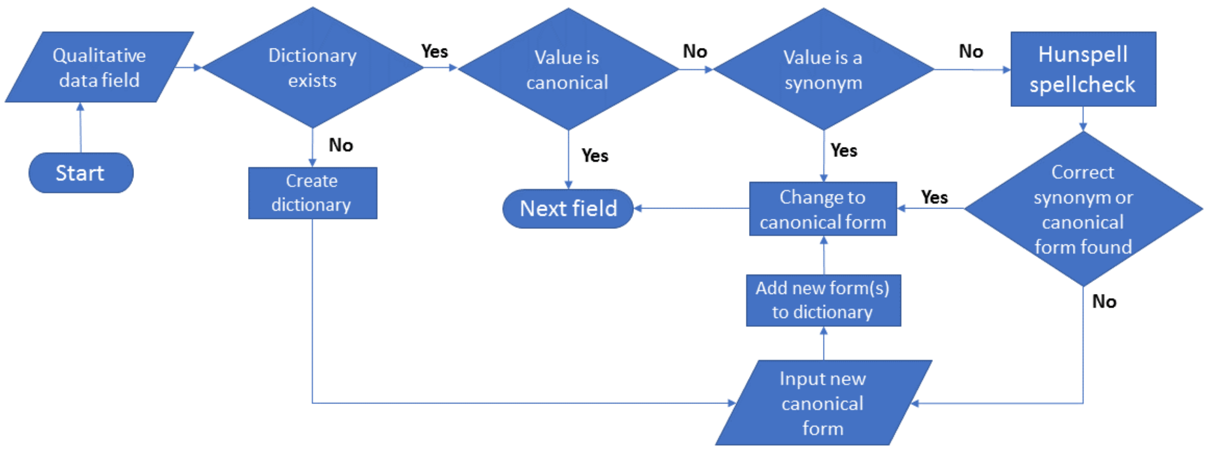


FIGURE 4: VOCABULARY NORMALISATION PROCESS FOR QUALITATIVE DATA FIELDS

The Vocabulary Normalisation parser module is designed to resolve both issues by storing "canonical" values which represent the "correct" version of each term used by SMIS. Whenever a novel value is encountered, the parsing tool attempts to either: a) if it represents a known entity, correct it to a pre-existing (or new) canon value or b) if it represents a genuinely new entity, add it to the known data field vocabulary as a canon value or a synonym to a canon value. An overview of this process is shown in Figure 4. It should be noted that vocabulary normalisation is only applied to data fields containing qualitative (textual) information. Data fields containing numerical values do not undergo normalisation.

Using the stored vocabulary as a dictionary, the widely used, open-source Hunspell spell-checking library is used to propose alternative (canonical) spellings or forms to the user who can then either select any of the proposed versions, view the entire vocabulary list for a data field to choose a different one, or enter a new canonical form into the database.

It is also possible to export and import vocabularies in tabular form. This functionality was developed primarily to speed up grower and experimental data parsing during development and testing, but the import option can also be used to effectively "front-load" the SMIS database, populating it with known values (e.g. herbicide names) which would ensure the validity of the terms used, while simplifying the work of a human curator during parsing.

3.1.2. Grower data

The parsing pipeline for grower data can be seen in Figure 5. The individual steps are discussed in sections below.

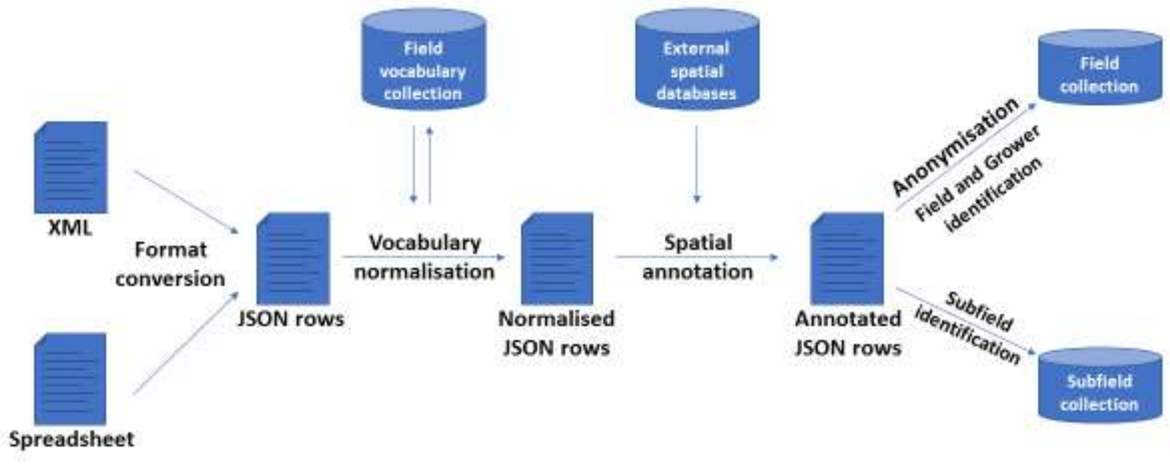


FIGURE 5: GROWER DATA PARSING PIPELINE

FORMAT CONVERSION

As mentioned in section 2.3, the currently handled formats are the GateKeeper XML format and Excel/CSV spreadsheets. Both formats are unnormalised and are thus straightforwardly convertible into sets of equally unnormalised JSON objects, each representing a single row of data. These JSON objects form a single canonical entry point for pipeline input, and therefore expanding SMIS to handle a novel grower data format requires implementing a script to convert that format into the JSON representation. For normalised datasets, this would require explicit denormalisation.

VOCABULARY NORMALISATION

JSON-formatted Grower data undergoes vocabulary normalisation as described in section 3.1.1.

SPATIAL ANNOTATION

As data stored in the database is intended to be anonymised, identifiable spatial information such as Ordnance Survey Map Sheet identifiers and National Grid (NG) codes is not stored. Any data from external databases which depends on knowing this spatial information, such as LandIS-derived soil information, need to be fetched and stored at this stage of the pipeline, before anonymisation. Database-specific import scripts can be included to pull such data on a farm field by farm field basis.

ANONYMISATION

Before being discarded to anonymise grower data, the Map Sheet identifiers, NG codes, and farm field names are concatenated and processed by a SHA-1 hash function to generate unique identifiers for each farm field.

An alternative to this approach would have been tokenisation, i.e. the generation of novel, random identifiers. While this could have been more secure, it would also prevent the possibility of importing updates to existing datasets (newer or previously missing data field information), as a relationship between the old and new data could not be established. It should be noted that tokenisation is used for identifying sub-fields and growers as described in the sections that follow.

GROWER AND FARM FIELD IDENTIFICATION

Each imported dataset is assumed to come from a single grower. However, successive (or otherwise separate) datasets can originate from a grower whose data are already in the database. While any duplicate entries can be discarded, novel data (e.g. covering later dates, or including data fields which were not previously available) has to be connected with existing data in order to give a full picture of factors such as the rotational context, effectively updating the database.

After the farm field anonymisation step described in the previous section, the identifier of each farm field in the dataset being imported is compared with those already present in the Field collection. One of the following three options is taken depending on that comparison:

a) If any of the farm fields being imported are already present in the database, the whole dataset is interpreted as coming from the same grower and any new farm fields are inserted into the Field collection alongside farm field–level information (e.g. “OS Area” and the unique grower identifier).

b) If none of the farm fields in the dataset being imported are present in the database, all of them are inserted into the Field collection alongside farm field–level information including a new, randomly generated grower identifier.

c) If the dataset being imported includes farm fields marked as originating from different growers (which is not the case in any of the data used in development, but is theoretically possible), the parser requires the user to choose which novel farm fields should be grouped with which group of existing farm fields. The new farm fields are then inserted into the database alongside farm field–level information.

Note that individual growers themselves are not represented by a separate collection in the database and are only represented by the randomly generated identifiers used in the Field collection.

SUBFIELD IDENTIFICATION/INFERENCE

In practice and in most grower datasets, farm fields are not indivisible entities, and multiple crops (accompanied by multiple corresponding sets of operations) may be grown concurrently on a single farm field, creating separate rotational contexts, separate histories of field operations, etc. For the purposes of SMIS, this separation of farm fields into virtual “sub-fields” needs to be captured.

This information may be contained in GateKeeper data (“Part Field Reference” data field), but it is missing from many real-life datasets, including some of those used in SMIS development. It is therefore necessary for the parsing pipeline to attempt to *infer* the subfields by their shared area.

A customisable hectarage margin of error (10% by default) is used in subfield inference, as the recorded areas of operations are rarely exactly the same.

Data rows for each subfield (identified either by the GateKeeper “Part Field Reference” or by the subfield inference functionality) are inserted into the Subfields collection together with a farm field identifier (allowing for aggregation with the Fields collection) and a new, randomly-generated unique subfield identifier. Individual operations covering the entire farm field rather than the area of a subfield are duplicated for each of the subfields.

3.1.3. EXPERIMENTAL DATA

The parsing pipeline for experimental data can be seen in Figure 6. The individual steps are discussed in sections below.

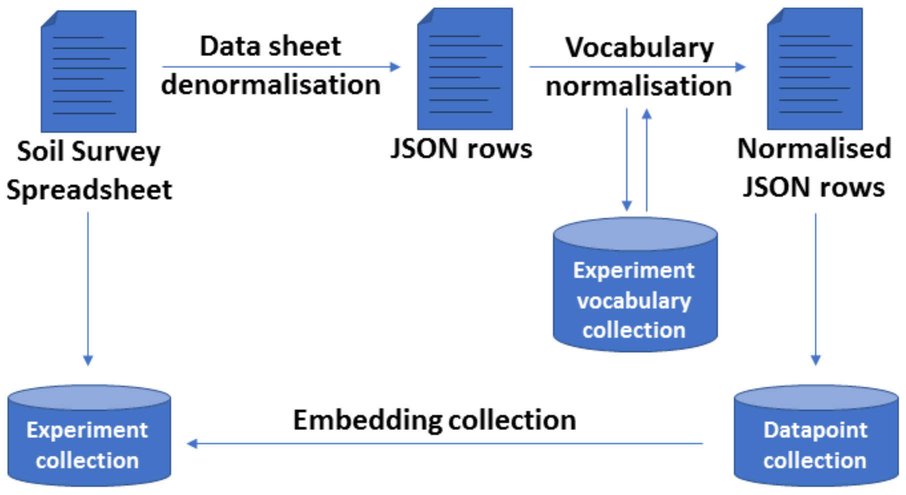


FIGURE 6: EXPERIMENTAL DATA PARSING PIPELINE

DATA SHEET DENORMALISATION

The "CP107c – Soils Programme: The application of precision farming technologies to drive sustainable intensification in horticulture cropping systems" Excel spreadsheet data points collected for the same farm fields on the same dates are split over multiple sheets, with one sheet per method. A Python script is used to bring the data from those sheets together, based on the collection dates and farm field identifiers. The collected data rows are converted into a JSON format, ready for insertion into a MongoDB collection.

When other experimental datasets (such as data from the "FV 447 - Carrots & Parsnips - Developing a strategy to control Free Living Nematodes" and "FV 380 - Identifying critical soil P in vining pea crops" projects) will be added to the SMIS database, novel scripts tailored to those datasets will have to be implemented for this stage of the data parsing pipeline.

VOCABULARY NORMALISATION

JSON-formatted experimental data undergoes vocabulary normalisation as described in section 3.1.1. In contrast with grower data, inconsistencies resulting in differences in terminology are not a major issue, but other sources of inconsistency such as typos do require correction.

DATA STORAGE

Finally, the collection of denormalised datapoint rows, with all of their quantitative data fields converted to canonical forms, are embedded in an element stored in the Experiment collection, identified by a randomly generated identifier.

3.1.4. LITERATURE DATA

Literature data is subject to manual curation as part of the SMIS project and summaries for all sources are prepared in the form of spreadsheets in a format directly mirroring the one shown in section 2.5. As such, parsing the data consists only of a single step which converts the spreadsheet into JSON format and persists the rows into the Literature collection. As with the experimental data, this conversion process is conducted using a specialised Python script specific to the formatting used in data preparation rather than through a more generic tool.