

# **D4.4: Analysis of Imaging Approaches**

Hendrik de Villiers, Rick van de Zedde, Ruud Barth, Tony Pridmore | 31 December 2019



## Abstract

Plant phenotyping experiments are increasingly generating vast amounts of data. The question of how to organize this data quickly arises. However, choosing a means of organizing such data is a non-trivial matter, as many solutions are possible. This has led to a lack of standardization in the field. This hampers the use and reuse of data, as users of a dataset have to write/adapt their own code to account for the particular structure of a new dataset. Furthermore, often dataset formats do not enable the detailed recording of metadata such as provenance information, potentially leading to a lack of transparency around the activities and structures within which the dataset was collected.

In this document, we consider the question of dataset organization within the context of plant phenotyping, with a special emphasis on computer vision data. A mapping/gapping analysis is presented based on summaries of interviews with the imaging experts involved in setting up/controlling the imaging pipeline within phenotyping research infrastructure in Europe, to understand their way of working and their expectations from EMPHASIS-PREP.

Subsequently, we discuss PHIS (Phenotyping Hybrid Information System), a prominent community effort specialized in the organization of plant phenotyping datasets. The discussion includes core semantic web concepts, as well as an overview of PHIS' approach to organizing information. Annexes are included which further expand on the information presented in the main matter, allowing the interested reader to learn more and experiment with using PHIS. This includes the uploading of a case study dataset containing hyperspectral image data. In addition, we consider the question of how to best preserve PHIS' ability to capture provenance metadata while connecting the system to subsequent machine learning workflows (with a special emphasis on deep learning).

## Acknowledgements

The following individuals and groups are thanked for their contribution to the creation of this document:

- INRAE Montpellier groups for initial testing VM, scripts and running a workshop on PHIS.
- Jean-Eudes Hollebecq (INRA) for feedback on the manuscript.
- Aneesh Chauhan, Esther Hogeveen, Gerrit Polder and Don Willems as project leaders supporting case study development.
- Aneesh Chauhan for advice on the tutorial code, as well as coordination between projects.
- Esther Hogeveen for provision of the initial development dataset.
- Gerrit Polder for provision of the final case study dataset.
- Don Willems for contributing to the material on semantic web standards.
- Ruud Barth, a former WR-colleague, for contributing to the chapter on data structuring patterns in computer vision.
- Sven Warris for feedback and contributing material on compute considerations to the section on machine learning with PHIS datasets.
- Aneesh Chauhan, Sven Warris, Don Willems and Hajo Rijgersberg for participating in internal discussions on PHIS usage and deployment.
- Peter Roos for feedback on the manuscript.
- The NPEC (Netherlands Plant Eco-phenotyping Centre), KB DDHT (Data-driven and High-tech program KB38-001-003/007) and HUMISTATUS projects for supporting this case study development.

## Table of Contents

Abstract .....	2
Acknowledgements .....	3
1. Introduction.....	7
2. Mapping/ Gapping analysis.....	11
2.1. Interviews and Responses .....	11
2.2. Analysis .....	13
2.3. Expectations .....	18
3. Dataset Structuring Patterns in Computer Vision .....	20
3.1. Positioning Phenotyping in Computer Vision .....	20
3.2. General benchmark datasets .....	21
3.3. Benchmark Datasets in Agriculture and Phenotyping.....	23
3.4. Suggestions for Phenotyping Dataset Organization and Management .....	27
4. Dataset Management .....	29
4.1. Introduction .....	29
4.2. FAIR Principles for Dataset Stewardship .....	30
4.3. Conclusion.....	31
5. Representing experiments and measurements using PHIS.....	32
5.1. Semantic data representation .....	32
5.2. Data representation in PHIS.....	34
5.2.1. Ontology of Experimental Scientific Objects (OESO) .....	35
5.2.2. Ontology of Experimental Events (OEEV) .....	35
5.3. The PHIS software ecosystem.....	36
5.4. PHIS Dataset Structure .....	38
5.5. Conclusion.....	40
6. Machine Learning with PHIS Datasets .....	41
6.1. Structure of machine learning datasets.....	41

6.2.	Challenges to automating machine learning dataset assembly from PHIS .....	43
6.3.	Trace-based Tensor Assembly and Provenance Capturing .....	45
6.4.	Roadmaps towards incorporation of trace-based provenance in PHIS.....	49
6.5.	Compute considerations .....	50
6.6.	Trace-based tensor description using JSON .....	51
6.7.	Roles in data science workflows.....	55
6.8.	ML-Schema .....	57
6.9.	Trained machine learning model interchange .....	58
6.10.	Summary and conclusion .....	60
7.	Summary and Conclusion .....	61
	References .....	63
	Document information .....	64
1.	Executive Summary.....	66
	Introduction.....	66
	Summary.....	66
	Recommendations .....	68
Annex 1:	Check list .....	70
Annex 2:	Further Semantic Web Concepts .....	71
2.1.	Controlling representations .....	71
2.2.	Examples of graph-based dataset representation.....	74
2.2.1.	Tabular data .....	74
2.2.2.	Time series .....	77
2.3.	The semantic web: IRIs and shared ontologies .....	78
2.3.1.	Unique identification of entities and their properties .....	79
2.4.	RDF and Triples.....	81
2.5.	Queries on triple stores using SPARQL .....	82
2.6.	Conclusion.....	83
Annex 3:	An Introduction to PHIS .....	84

3.1.	REST services for software developers .....	84
3.1.1.	Identifying web services using URLs .....	85
3.1.2.	Kinds of HTTP request.....	85
3.1.3.	Passing Information to a Web Service .....	86
3.1.4.	Server responses to HTTP requests .....	87
3.1.5.	Examples of web requests supported by PHIS .....	87
3.2.	A First Example in Python .....	89
3.3.	Using Swagger Documentation to Explore the PHIS API .....	91
3.4.	Conclusion.....	96
Annex 4:	Dataset Capturing in PHIS: A Case Study .....	97
4.1.	Case study dataset.....	97
4.2.	Using the example code .....	98
4.3.	Projects .....	99
4.4.	Experiments.....	107
4.5.	Sensors .....	109
4.6.	Provenance.....	111
4.7.	Scientific Objects .....	113
4.8.	Uploading Data .....	115
4.8.1.	Point data .....	115
4.8.2.	Bulk data .....	117
4.8.3.	Cross-referencing between data subsystems.....	118
4.8.4.	Uploading the case study data: Ground truth labels .....	118
4.8.5.	Uploading the case study data: Image data .....	123
4.9.	Conclusion.....	124

## 1. Introduction

In the scientific community, but also in the corporate world, data management and organisation is a very splintered phenomenon. There is a wide variety of ways the information can be stored and handled. This is coming forth from the premise that data objects and their properties are actually very simple constructs for which there is no obvious, single logically intuitive way of storing and organising them. On the one hand, this leads to a multitude of implementations each specific for their application, serving as input for proprietary processes. On the other hand, usually any data format can also be transformed to others using simple scripts (given the data and their properties are similar and from the same domain, e.g. stock data cannot meaningfully be translated to image data.)

Plant phenotyping experiments often collect vast amounts of data, yet typically the format in which this data is stored is particular to the given project. In addition, metadata enabling the understanding of the data by third parties is often not recorded. This means that such data is not easily reusable by other groups, perhaps even within the same organization. While there are some common structuring patterns for datasets, easy interoperability between systems using data collected in different experiments is usually not expected. Even if the need for later reuse is discounted, the lack of standardization means that researchers have to rewrite the same kind of processing script for each project, which is both wasteful and can lead to unnecessary error.

In this document, a mapping and gapping analysis is presented based on summaries of interviews with the imaging experts involved in setting up/ controlling the imaging pipeline within phenotyping research infrastructure in Europe, to understand their way of working and their expectations from EMPHASIS-PREP. Digital images are central to plant phenotyping, for a variety of reasons. Many traits of interest are visible to the eye, suggesting that they can in principle be recovered automatically from appropriately captured and processed image data. Though images of plants present new challenges, image analysis, and the broader field of computer vision, have provided a wide variety of techniques and tools which have the potential to achieve this. Moreover, most imaging devices produce broad-field data which is open to multiple interpretations; once an image is captured it can be analysed in different ways, by techniques which focus on different areas of the visual field. An image initially captured to support e.g. wheat ear counting can later be analysed to extract leaf traits, or provide improved ear counts when new methods become available.

Subsequently, current standards for data set management for image data are investigated. From this, suggestions are made for **organizing phenotyping data**. It must be noted that the results are highly biased towards current best practices in computer vision. Given that machine learning, notably Deep Learning, is the current state-of-the-art, this forms the suggested data set organisation. Furthermore, because the field is moving extraordinary fast, changes in managing the data should be expected in the near future already. Hence this document comes from a very specific perspective in both domain and time. One take home message is that the methods applied to the data dictate the data format and organization and less so the other way around. Choosing flexible data formats can help storage approaches adapt with changing methods of data processing.

It must also be said that even within this domain, there is a large variance of type of data objects and their properties and relations. For example, some images are annotated with only bounding boxes, others on a per-pixel level, others in a time-series from frame to frame, or even with sub-components with abstract spatial links.

Recently, efforts have been made towards enabling easy reuse and interoperability of plant phenotyping datasets. One prominent effort is the ongoing PHIS (Phenotyping Hybrid Information System) project (Neveu et al., 2019). This system is currently in development and evolving through feedback from the community. While a variety of resources exist describing the principles and application of this approach to dataset creation, this information is (generally speaking) distributed across a number of sources.

A further aim of this document is to provide a guide to the most important aspects of using PHIS to record experimental design and measurements. High-level aspects of PHIS' organization are discussed in the main text, while in the annexes a more thorough overview of the practicalities of interacting with PHIS is provided. The overview is intended to be as self-contained as possible, while citing useful resources for further study. Practical examples are provided, including source code where applicable. Source code examples are in the Python language, since this is a commonly used language in data science. However, we will take care to help users transfer these concepts to their own working languages where possible. In particular, PHIS implements interaction with user code via REST services, which can be called the vast majority of programming languages. No previous experience of interacting with REST services will be assumed on the part of the reader.



Where necessary, we make proposals for encoding aspects of machine learning experiments in a way which complements existing PHIS facilities.

Chapter 2 presents a mapping and gapping analysis based upon a set of one-to-one structured interviews with platform staff identified as closely involved in image analysis and the recent, relevant trends in the image analysis and computer vision literature.

Chapter 3 relates the domain of plant phenotyping within the domain of computer vision, and gives an overview of current approaches to structure commonly used vision datasets. Furthermore, key plant phenotyping datasets which use these formats are identified and discussed.

In Chapter 4, we discuss the challenges inherent in recording experimental design and measurements in a truly reusable and interchangeable way. The FAIR guidelines for management and curation of datasets are presented as a key set of criteria to enable such reuse. We review a selection of methods attempting to address these issues, with special focus on the approach taken by PHIS and related projects.

PHIS relies heavily on representing knowledge using the approach taken by the semantic web. Therefore, Chapter 5 provides a compact overview of the most important concepts underlying the semantic web. The chapter will also introduce the various subcomponents of PHIS. While knowledge representation is a significant part of PHIS, other subcomponents enable users to access or alter this knowledge with a variety of methods. We will focus on portions of the system which are needed to begin using PHIS, and provide references for using the more specialist aspects of the system. To support the material on PHIS, annexes 2 through 4 provide additional technical information, as well as a worked example of the uploading of a case study dataset to PHIS. This includes uploading scripts in the Python language.

Chapter 6 considers how to extract information from a PHIS dataset for use in training and testing of machine learning models. PHIS attempts to provide a detailed representation of an experiment's design and measurements. This representation is meant to provide a single, definitive record of events that occur during an experiment. However, machine learning researchers would likely not do training and testing directly based on the PHIS dataset. Multiple teams may be modelling different subsets of the larger dataset. Machine learning methods, particularly deep learning, are dependent on so-called "tensors". Assembling such tensors from the appropriate subset of data in a PHIS

dataset is important, but it is also important that information about how the tensors were assembled is retained. We make proposals for representing such assembly procedures in a way which complements PHIS. In this way, reproducibility of machine learning experiments can be enhanced.

Machine learning models, particularly deep neural networks, can take a long time to train, and are themselves valuable research outputs. Such models are often reused by other groups on novel tasks. In the last portion of Chapter 6 we discuss ways in which models may be stored for reuse and possible dissemination (focusing mainly on deep neural networks).

In Chapter 7, we conclude the text with a discussion and suggestions for further reading to deepen the material covered.

## 2. Mapping/ Gapping analysis

The first EMPHASIS-PREP survey of phenotyping platform operators addressed a variety of topics spanning all of EMPHASIS' pillars and goals. This attempt to create a single, comprehensive survey provided much useful information, but inevitably also raised questions that required deeper analysis. To clarify and build upon the understanding gained of the image analysis approaches and used across the consortium, and to identify opportunities for improvements and areas needing greater attention, a series of interviews were conducted with staff members identified as closely involved with image analysis in their platforms. It is upon these interviews and observations of the recent relevant literature that this analysis is based.

### 2.1. Interviews and Responses

A set of twelve interviews were conducted by representatives of the EMPHASIS Core Group's facilities. EMPHASIS-PREP partners were asked to identify individuals they considered their imaging/image analysis experts, those who were closely involved in setting up and maintaining the image analysis component of the installations they operate. It was made clear that interviewees should have the deepest technical understanding of the methods and tools being used that was available at the platform.

Interviews were conducted by Rick van de Zedde and Tony Pridmore by telephone, skype or similar web conferencing tools over a period of some 10 months. Initial contact with the interviewees made it clear that EMPHASIS-PREP's goals in this were to understand both the image analysis facilities employed and their way of working with them, as well as seeking some knowledge of their expectations from EMPHASIS-PREP. A set of questions was drawn up around which the interviews were structured, though the varying backgrounds of the interviewees meant that individual conversations inevitably varied somewhat. The questions were organised into four clusters:

- General: These sought to clarify the respondent's background, providing useful context for the remainder of the discussion, and the nature of the platform with which he/she is associated. Specific questions asked were: *What is your role/responsibility in the facility? What part of you is a biologist/ data scientist/ technology developer (in percentages). What is your background/professional profile? Please describe your platform.*
- Imaging and Image analysis: These questions were selected to provide information on both the methods and tools in use, and the platform staff's relationship to them. The latter is

key, as EMPHASIS services must match the needs and abilities of installation operators. Specific questions asked were: *Can you describe the imaging pipeline you use, i.e. what sequence of processes is applied to the initial image? Do you know which specific algorithms are employed (e.g. Otsu or Rosin thresholding)? Which software tools and/or programming languages do you use? What degree of control do you have over the pipeline? Can you set camera positions, individual algorithms' parameters? How many cameras are involved, of what type? What relevant functionalities do you use? How/what do you calibrate, and how do you verify your system?*

- **The Team:** This section aimed to clarify the amount of support available within the platform for the installation, maintenance, operation and development of imaging and image analysis functionality. Specific questions asked were: *Can you describe your technical development team? Do you have a particular image analysis specialist who you work with? How do you gain new knowledge on new imaging developments (sensors/ software/ data analytics)? And are you able to integrate that into your systems/ imaging pipeline?*
- **Data and Computational Services (previously e-Infrastructure):** The final section of each interview focussed on the data produced, and how it was stored, managed and further analysed. Specific questions asked were: *Which tools do you use for data handling and storage? Are you using external services for data sharing? Do you use a standardized identification system? Do you use standardized protocols? Are these metadata organized with standardised languages? Have you designed a long term archiving capacity? Are you using a proprietary information system for sorting and handling data? Are data generated on your platform available through web services? Which tools are you currently using for data handling and storage (e.g. file based, hard drive, CSV file)? What is the storage capacity of your computer infrastructure?*

Each interview ended with a final, overarching question: *What are your expectation concerning EMPHASIS in relation to these components of your operation?*

Interviews were organised with colleagues from Belgium, UK, Netherlands, Germany, Austria, Italy and France. Figure 1 shows the number of interviewees from each country. These were drawn from 10 distinct institutions: ALSIA, INRA, IPK Gatersleben, JPPC, Rothamsted Research, Universite Catholique de Louvain, University of Nottingham, VBCF, VIB and Wageningen University. Most contributed a single interview, with the wider range of facilities available at INRA, Universite

Catholique de Louvain and Wageningen University leading them to contribute 3,2 and 3 interviews respectively.

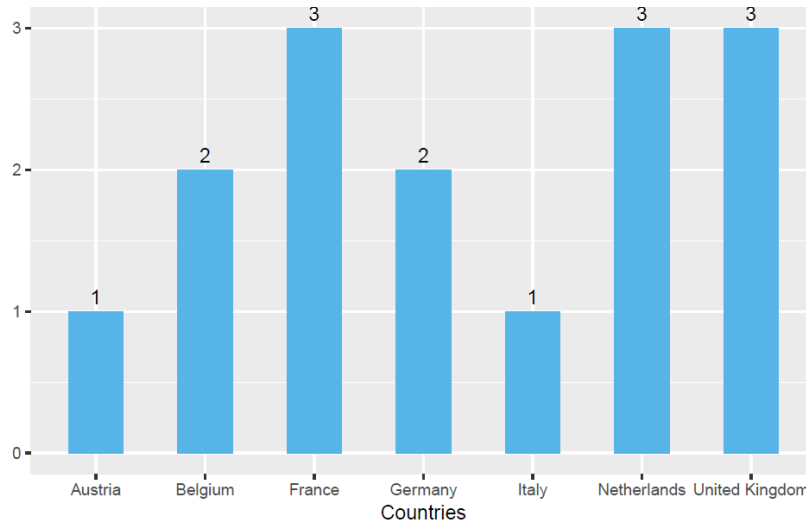


Figure 1: Number of interviewed experts in the different countries.

Comprising a total of 27 questions, each interview represented a more detailed investigation of the platform’s imaging, image analysis, data and computational services than was elicited by the initial EMPHASIS-PREP survey. Each interview lasted 40-60 minutes, with written notes being captured by the interviewer.

## 2.2. Analysis

The interviews were structured to allow a natural conversational progression, and so proceeded from general discussion of the interviewee’s background and skills, through the tasks they perform and tools they use to the human and computational support they receive. When mapping the image analysis landscape it is, however, more informative to follow a different path. In what follows we focus on the hardware and software tools and methods in use within the community and discuss platform staffs’ capabilities in regard to them.

### 2.2.1. Imaging and image analysis

**Number of cameras:** Figure 2a summarises the range of imaging devices (and so image data) found in platforms operated by the EMPHASIS core group. Standard colour cameras dominate, reflecting

their generality and low cost. Figure 2b summarises the number of cameras found in phenotyping platforms. Those with lower numbers of cameras tended to be either specialist installations relying heavily on e.g X-ray CT or Lidar. Those with 4+ devices typically employed a variety of imaging modalities, though again RGB cameras were the most common. Though the question was not explicitly asked, interview responses suggest that in most cases multiple cameras are used to provide coverage of multiple plants or growth environments: only three interviewees mentioned the use of multiple cameras to recover 3D information. A majority of image analysis performed within these installations is two dimensional; measurements are made on the image plane. Those that do recover 3D information are more likely to use specialist hardware to do so. Similarly, those employing multiple camera types tended to describe them independently; there is little evidence here that sensor fusion methods are currently being used in plant phenotyping.

**Mapping:** RGB cameras and independent 2D analysis of the resulting images dominates current practice.

**Gapping:** More might be made of the combination of multiple RGB images, and of images obtained from different types of imaging device.

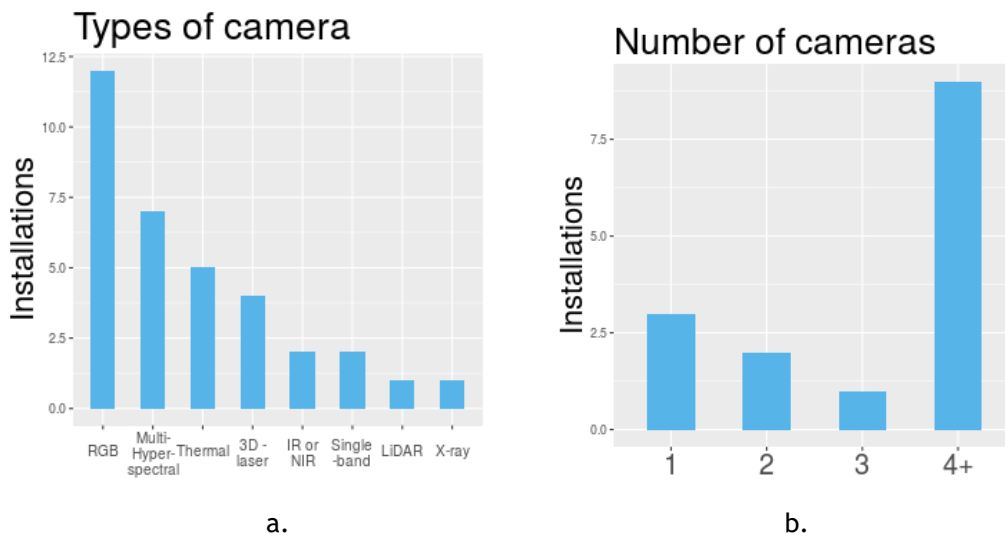


Figure 2. Histograms summarising a) the type and b) the number of cameras used per group in phenotyping platforms operated by the EMPHASIS Core Group.

**Calibration of cameras:** Two-dimensional image analysis can provide a great deal of information regarding viewed objects, but requires the camera(s) used to be calibrated. Unless the relationship

between real world dimensions (e.g. mm) and image plane measurements (made in pixels) is known, any change in camera settings or position will disrupt the data provided.

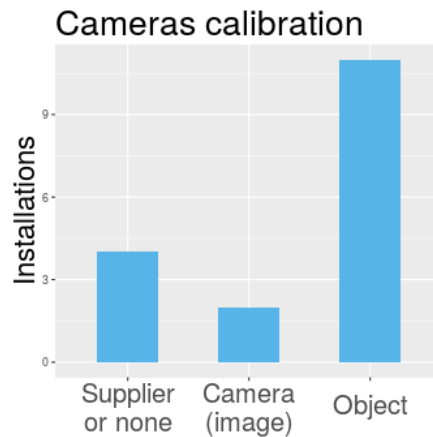


Figure 3: Calibration methods used in Core Group facilities.

Figure 3 summarises the calibration methods used in the platforms surveyed. A large proportion employed a standard camera calibration target and software tools to recover camera parameters, giving them full control over the calibration process. Two more used image-based methods, in which they manually compared pixel measurements with corresponding plant dimensions to compute a relationship. Only 4 relied upon manufacturer’s methods or did not calibrate at all, but these included installations which relied upon more specialist devices such as X-ray CT, for which this is appropriate. During the conversations only 3 platform operators expressed concerns about the calibration methods available to them. A variety of tools and targets are, however, in use, and calibration is performed with varying frequency.

**Mapping:** Appropriate calibration tools are and processes are in place in a large majority of platforms.

**Gapping:** There is an opportunity for greater standardisation of calibration methods and frequency. All the methods described are internal to the given platform; no mention was made of calibration against other platforms producing similar data.

**Imaging analysis tools:** Phenotyping installations gather images of a wide variety of plant species and organs in controlled environments, glasshouses and fields. They seek to provide phenotyping data at the cell, organ whole plant and plot scales to support the recovery of a wide variety of

traits. It is therefore to be expected that there will be significant variation in the image analysis techniques, tools, and particularly pipelines employed. These pipelines might be supplied by commercial concerns, built in-house or obtained from other platforms and phenotyping research groups. A key question, given the variation between platforms' needs, is how much knowledge and understanding of the underlying methods do platform staff have, and how much control can they exert?

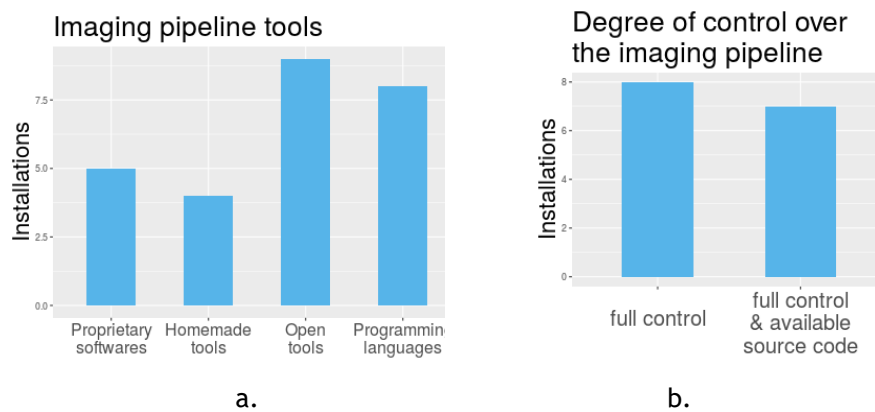


Figure 4: a) Sources of image analysis software within the Core Group and b) perceived degree of control over the image analysis pipeline by platform staff

Figure 4a gives a histogram showing the frequency with which image analysis software within the Core Group is obtained from the available sources. Figure 4b summarises interviewees' response to the question of how much control they feel they have over their pipelines. All state that they have full control, in the sense of being able to manipulate what they consider to be the key parameters. Approximately half also have access to source code, and so could make structural and detailed changes to the processes applied, as well as having full knowledge of the operations being applied.

Over 80% of those interviewed were able to describe the pipelines used in their platforms to a high level of technical detail, citing specific algorithms and showing clear understanding of the interactions between them. The pipelines described during the interviews, however, shared a common feature: all were based upon classical image analysis operations such as noise removal, thresholding, segmentation, mathematical morphology, etc. All were also straightforward linear processing chains, in which the output of one operation becomes the input of the next. The technology which has had the greatest effect on the wider computer vision community in recent years - deep machine learning - was conspicuous by its absence. Though most platforms are able to



process their images with their own software tools, we expect that flexibility in these tools towards new plants/new situations is limited and that deep learning will offer significantly increased flexibility (with explanation) without requiring significant re-design of the tools concerned.

**Mapping:** Phenotyping platform staff have a high degree of understanding of and control over the image analysis pipelines they oversee. The pipelines described rely exclusively on classic image analysis operations.

**Gapping:** The degree of control and widespread use of home-grown and open source software means that there is little/no standardisation in the software tools used, raising the possibility of the EMPHASIS community developing shared approaches. The most significant technological gap the lack of exploitation of recent advances in deep learning, which requires access to sizeable annotated data sets. We return to this below.

#### 2.2.2. Data and Computational Services (e-Infrastructure)

Interviewees' responses to the questions probing e-infrastructure suggest a community in transition. A significant majority are currently using straightforward data storage techniques, in many cases storing images and trait data in everyday formats such as csv on local machines, organising the data only by constructing an appropriate folder hierarchy. Some 15-20% reported going a little further and using standard database tools such as SQL. The systematic use of standardised identifiers for plants and other objects is comparatively rare, and very few reported use of application-specific data management software. Approximately half of the interviewees, however, described partial use of more advanced methods and/or an intention to adopt them in the future. Several were part-way through installation of PHIS at the time of the interview, and others stated an intention to use either PHIS or PIPPA.

**Mapping:** Current techniques are predominantly local, and rely on standard file structure and database methods. There is however, evidence of a move towards more systematic representation, storage and management of image and related data.

#### 2.2.3 Individuals and Teams

Within this set of platforms all but one of the interviewees reported having some training or experience of engineering, computer or data science, usually along with some biological qualifications. Those that were primarily biologists all reported having easy access to staff with the

required technical knowledge. All those interviewed either stated or implied that they work with image analysis and computer vision experts within their own facilities, rather than external groups. It is noted that the interviews only included members of the EMPHASIS-PREP Core group, who may have a greater interest in and motivation for technical development, and so employ a wider range of staff than institutions more tightly focussed on experimentation. Should this be the case, however, the Core Group staff remain a significant resource, with potential for technology development and transfer across the wider community.

When asked how they obtain new knowledge, interviewees listed standard academic mechanisms such as via journal and conference papers, workshops and tutorials. Only a few conferences (e.g. CVPPP) were mentioned by name, but again there was little standardisation or commonality in the sources used.

Mapping: Phenotyping platform staff span a range of disciplines and are a significant resource.

Gapping: There no evidence of commonality in the sources of information they use, which may limit information flow across the community.

### 2.3. Expectations

The final question of each interview addressed respondents' expectations of EMPHASIS, in the context of imaging, image analysis and the resulting data. The most common expectations were related to data; with the development and provision of open data standards the most frequent and often first response. Data sharing was the second most common topic raised. This referred to image and trait data, but it seems reasonable to assume that EMPHASIS members will come to expect the network to also support access to the annotated images needed when training deep learned solutions.

In terms of processing techniques, only shared libraries were mentioned, and not shared pipelines. This may be a result of the high level of internal development of classic pipelines currently seen, and/or the variety of tasks performed across the community: members do not think of pipelines as being transferable, but are interested in libraries from which they can construct their own. Given the emphasis placed on pipelines and workflows in other ESFR projects such as Elixir, this is surprising. Workshops, training events and other information exchange methods were also highlighted as desirable.

Mapping: Current expectations focus more on data than processes.

Gapping: There was little discussion of representation and sharing of pipelines, which should perhaps be addressed.

## 3. Dataset Structuring Patterns in Computer Vision

### 3.1. Positioning Phenotyping in Computer Vision

Before it can be suggested how to shape phenotyping data, first it needs to be roughly positioned in the computer vision domain. The scope of datasets in computer vision is very wide. Below a limited short list from different sub-domains. It shows already the diversity that can be expected in potential organizing data in each domain:

- Low-level Vision
- Optical Flow
- Video Object Segmentation
- Change Detection
- Image Super-resolutions
- Intrinsic Images
- Material Recognition
- Multi-view Reconstruction
- Saliency Detection
- Visual Tracking
- Visual Surveillance
- Visual Recognition
- Scene Understanding

To pinpoint the most relevant scope for phenotyping, first needs to be estimated where phenotyping for the most part resides. For this, it is required to get a common understanding or definition of what phenotyping with images actually entails:

#### ***Plant Phenotyping***

The observable physical characteristics of a plant.


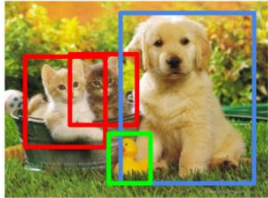

The domain of ***Visual Recognition*** is the field where from image data is recognized what is where. This highly overlaps with phenotyping, where from images it needs to be derived where the plant (parts) are and thereafter, with use of some additional processing steps, link the location of those parts to physical characteristics, for example by using 3D reference images.

Other domains, such as multi-view reconstruction or scene understanding might also be relevant for phenotyping, albeit those are more application specific.

*Visual recognition* can be further sub-divided into the fields in the following table, from a higher-level recognition, to a fine-grained lower-level recognition: image classification, object detection and semantic segmentation.

### 3.2. General benchmark datasets

For image data in each of these sub-domains, the community has several benchmark datasets that existed over a decade <sup>1</sup>. With these datasets, new methods of learning are tested and compared to the state-of-the-art. The dataset management and organization of these datasets remained relatively stable, although they got extended when new techniques required new and different input formats.

Sub-Field	Example	Benchmark Datasets
<p><b>Image Classification</b> Estimate what the image is mainly about.</p>	 <p>CAT</p>	<ul style="list-style-type: none"> <li>• The PASCAL Visual Object Classes (VOC) Dataset.</li> <li>• ImageNet Large Scale Visual Recognition Dataset</li> </ul>
<p><b>Object Detection</b> Pin-point where objects are in the image with bounding-boxes.</p>	 <p>CAT, DOG, DUCK</p>	<ul style="list-style-type: none"> <li>• The PASCAL Visual Object Classes (VOC) Dataset.</li> <li>• ImageNet Object Detection Dataset.</li> <li>• Microsoft COCO Dataset</li> </ul>
<p><b>Semantic Segmentation</b> Determine for every pixel to which object class it belongs.</p> <p>Depending on the methods may also include object instance detection.</p>	 <p>CAT, DOG, DUCK</p>	<ul style="list-style-type: none"> <li>• The PASCAL Segmentation Dataset</li> <li>• Microsoft COCO Dataset</li> </ul>

It is these datasets that define also the current standards in dataset management and encoding. For phenotyping applications, it is key that plant (parts) are segmented on a per pixel-level and moreover

<sup>1</sup> A comprehensive list of datasets can be found here: <https://www.datasetlist.com/>

different instances can be discerned. Hence the sub-field within visual recognition of semantic instance segmentation would be the most relevant scope to further expand on.

There are 2 main ways to organize semantic instance segmentation data, coming forth out of the two benchmark datasets.

### *The PASCAL Segmentation Dataset*

The ground truth label of each image consists of a pair of images, one with color information and the other with a per pixel color encoding of the class of interest, with a different color per instance.



### *Microsoft COCO Dataset*

For COCO a different approach was chosen with the benefit that storing and parsing the ground truth labels is more efficient and lightweight. Instead of storing annotations as additional images, a text based description is used, which can be extendable

The annotations are stored using JSON (JavaScript Object Notation) which is a lightweight data-interchange format. It is possible for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange format.

Each object instance annotation contains a series of fields, including the category id and segmentation mask of the object. The segmentation format depends on whether the instance represents a single object (iscrowd=0 in which case polygons are used) or a collection of objects (iscrowd=1 in which case

RLE, or run-length encoding, is used). Note that a single object (iscrowd=0) may require multiple polygons, for example if occluded. Crowd annotations (iscrowd=1) are used to label large groups of objects (e.g. a crowd of people). In addition, an enclosing bounding box is provided for each object (box coordinates are measured from the top left image corner and are 0-indexed). Finally, the categories field of the annotation structure stores the mapping of category id to category and supercategory names. See also the detection task.

Below an example is given for the encoding in the JSON file.

```
annotation{
  "id"          : int,
  "image_id"    : int,
  "category_id" : int,
  "segmentation" : [polygon], e.g. ([[x1 y1 x2 y2], [...], ...])
  "bbox"        : [x,y,width,height],
  "area"         : float,
  "hasSomething" : 0 or 1,
  "isCrowd"     : 0 or 1,
}
```

### 3.3. Benchmark Datasets in Agriculture and Phenotyping

The previously mentioned benchmark datasets have a broad general everyday life content with classes such as cars, trees, humans and pets. There also exist benchmark datasets for agriculture and phenotyping specifically.

#### 3.3.1. Leaf Segmentation Challenge Dataset

To advance the state of the art in leaf segmentation and to demonstrate the difficulty of segmenting all leaves in an image of plants, the Leaf Segmentation Challenge (LSC) was created (Minervini,

Fischbach, Scharr, & Tsaftaris, 2016). For the challenge a training set was released which contains raw images and annotations <sup>2</sup>.

The format of the challenge is similar to the PASCAL Segmentation Dataset (example data from this dataset can be seen in Figure 1. Each instance of leaves has a unique color label and a per pixel annotation is used. Furthermore, there are additional CSV data files that include physical plant parameters like leaf size, center and bounding boxes. All images were hand labelled to obtain ground truth masks for each leaf in the scene. These masks are image files encoded in PNG where each segmented leaf is identified with a unique integer value, starting from 1, where 0 is background. For the counting problem, annotations are provided in the form of a PNG image where each leaf center is denoted by a single pixel. Additionally a CSV file with image name and number of leaves is provided.

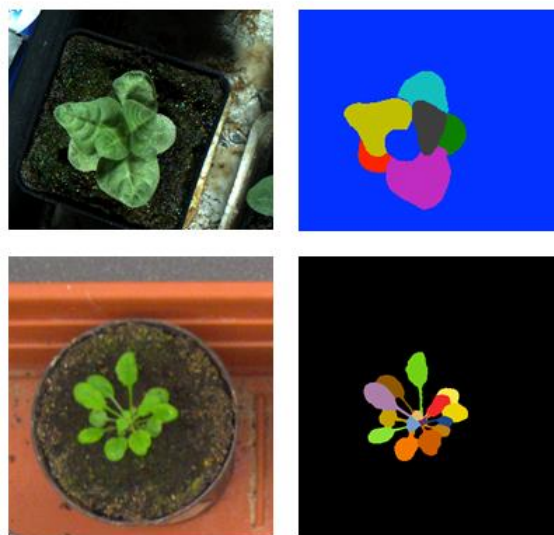


Figure 1: Leaf Segmentation Challenge Data

---

<sup>2</sup> <https://www.plant-phenotyping.org/cvppp2014-challenge>



### 3.3.2. *Sweet-Pepper Dataset*

For plant part detection and instance segmentation, a synthetic and empirical (real) dataset was created<sup>3</sup> of a commercially grown sweet-pepper crop. Samples from this dataset can be seen in Figure 2. For each plant, up to 8 classes were annotated.

The synthetic data (top two images, left-hand column of images) was based on distributions of plant part measurements encoded in a model to procedurally generate plants. These models were then rendered to obtain photorealistic scenes and corresponding ground truth (Barth, IJsselmuiden, Hemming, & Henten, 2018). The empirical data (bottom two images, left-hand column of images) was collected in a greenhouse and manually annotated. Initially the ground truth label format was based on the PASCAL Segmentation Dataset. However, instances were not included and therefore any overlapping objects could not readily be discerned. In a later approach, the dataset was expanded to include bounding boxes and instance segmentation for 3 plant part classes (right-hand square of 4 images). This format was according to the COCO dataset and included a JSON file as annotation.

---

<sup>3</sup> <https://data.4tu.nl/repository/uuid:884958f5-b868-46e1-b3d8-a0b5d91b02c0>

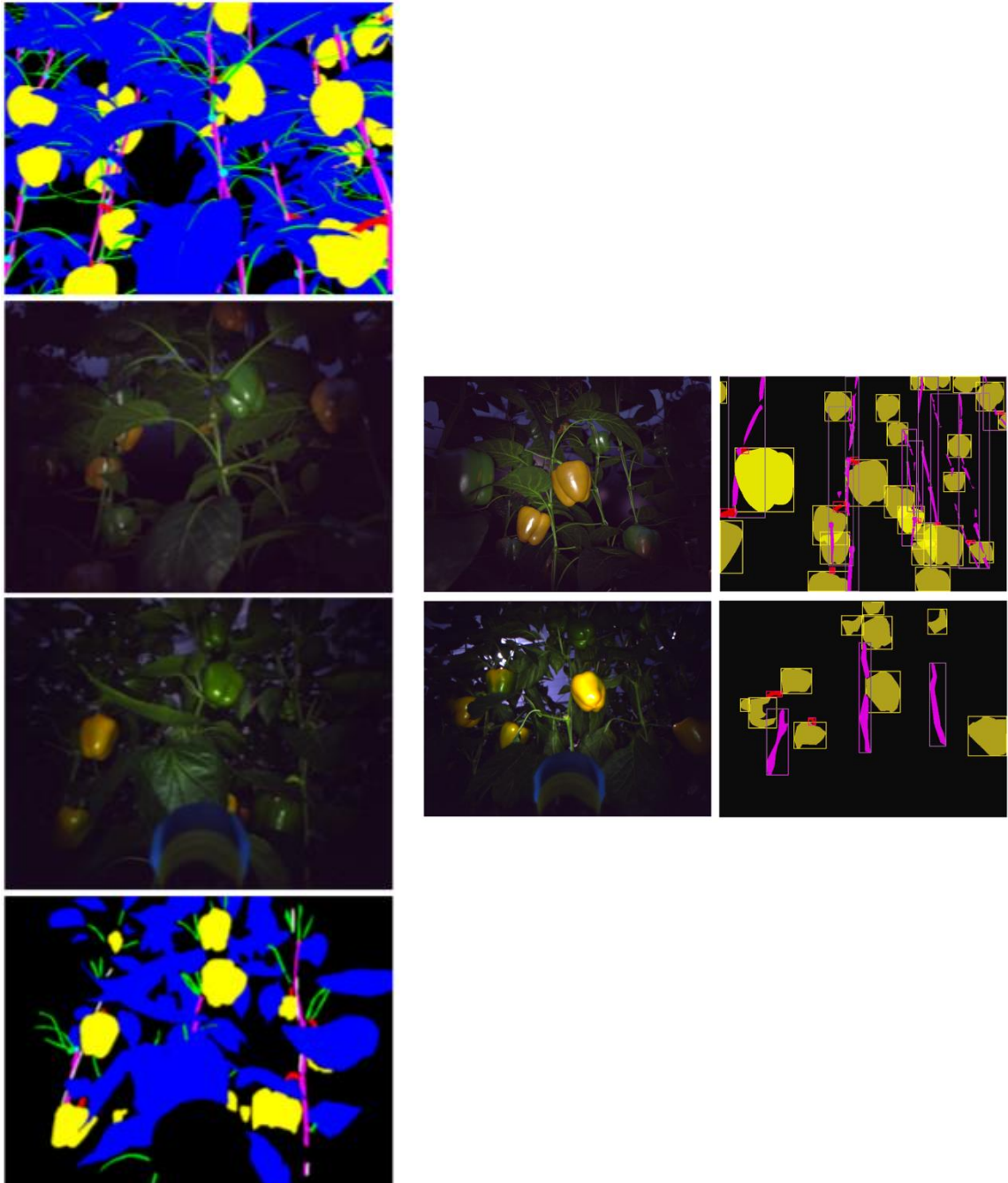
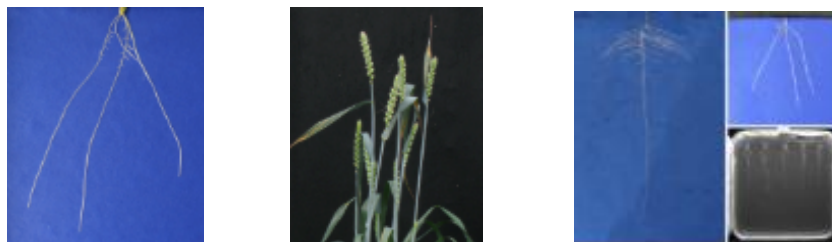


Figure 2: Sweet Pepper Data

### 3.3.3. Annotated Crop Image Database (ACID)

Initially created to manage training sets constructed during development of a series of convolutional neural net solutions to phenotyping tasks at the University of Nottingham, ACID has now been opened to the community (<https://plantimages.nottingham.ac.uk/>). ACID currently contains three datasets:

- Wheat blueprint provides images and RSML (Root System markup Language) annotations of wheat seedlings grown hydroponically. RSML provides a full structural description of the root architecture which has been used to create training images for both image segmentation and feature detection tasks.
- Wheat 2017 provides images and spike/spikelet localisation, expressed as JSON files, for use when training localisation algorithms
- Rootnav 2.0 contains images of hydroponic wheat and oilseed rape and arabidopsis seedlings grown on gel plates. These images were used to train RootNav 2.0. Images are supplied with RSML annotations.



ACID includes a search facility which allows users to select data items and form their own collections from search results. Funded by the UK plant phenomics network, PhenomUK, a new version of ACID is under construction which will support video, 3D and multi-view datasets and allow users to host their own plant datasets on ACID if they so wish.

## 3.4. Suggestions for Phenotyping Dataset Organization and Management

This chapter has provided a concise overview of how datasets and their annotations are managed for state-of-the-art methods in the computer vision and machine learning domain. These ways of organizing the data can be an important guideline for how phenotyping data should be structured, as computer vision is generally the first step in analyzing the physical characteristics of plants.

When phenotyping data is structured in a similar fashion as the mainstream computer vision datasets, then the power of state-of-the-art algorithms in that domain can become more easily deployed on phenotyping data. Currently, the COCO dataset structure that combines color images with a JSON file

description seems to be the most suited for phenotyping, as it allows for instance encoding but also extension of all kinds of physical plant parameters. For example, the center and area of leaves, GPS location or any other trait can be easily extended in the JSON file.

However, as the spectrum of phenotyping in itself is as broad as the spectrum of computer vision applications, each goal requires a tailored approach. In turn, this will dictate the dataset organization.

It is very important to keep following the main field of computer vision and continuously adapt current and older datasets to newly introduced formats (e.g. Capsule Network might require new formats in time). This requires that the old formats are already in a shape of the previous state-of-the-art, as this makes them more easily translatable to the new formats. Hence, dataset management and organization also should include tools to transfer and interpret the current dataset format. In the end, dataset organization is ambiguous and fluid, which requires maintenance. There is no golden standard. The upside is that the underlying data is usually straightforward; an image has objects that has properties. Hence, a good start, based on previously proven structures is very much suggested.

In the subsequent chapters, we discuss open approaches to representing datasets adapted specifically to phenotyping data. We consider PHIS as a means of recording and organizing raw data obtained from measured objects. Needs of machine learning researchers with respect to accessing this data are considered. Finally, assuming the use of Deep Learning, the dissemination of resulting deep neural networks is discussed.

## 4. Dataset Management

### 4.1. Introduction

With every passing year, more plant phenotyping datasets are recorded, with ever increasing size and complexity. Often, the use of a particular dataset is primarily envisaged to be tied directly to a given experiment. While ideally the reuse of such datasets would maximize their impact on research activities, the reusability of datasets is often hampered by a number of factors.

Of particular concern is the general lack of standardization through which datasets are recorded. Typically, a dataset's internal structure is unique to the project itself. In certain cases, a dataset has a similar purpose to another well-known dataset, in which case the well-known dataset's format can act as a kind of ad-hoc standard. However, there is generally not an expectation that the processing code for a given project will be interoperable with another.

The challenge of creating interoperable and reusable data formats for storing measurement data is not new, and many formats have been proposed to fill this need. However, the requirements of different problem domains have led to a diverse set of solutions which range from the highly specialized (such as image data formats) to the highly general (such as XML).

Furthermore, storing measurement data in an interoperable format is not the same as providing a dataset in an interoperable format, as there is a large amount of metadata and high-level experimental structure that needs to be encoded to make this measurement data meaningful. For example, with what hardware and with what settings was a given measurement performed? How does a measurement relate to entities being measured, such as individual plants? How do we associate different kinds of measurements performed on the same entity? The availability of such metadata improves the reproducibility and subsequent extensibility of results.

Another important example is that of data annotations, which provide information about aspects in the data that go beyond the raw measurement and its capture parameters. For example, an image dataset might contain raw images, but may in addition contain annotations showing where a particular kind of object, such as individual fruits, are positioned within a given image. In this example, we can then use the images and their annotations to train a model to predict the location of fruits within previously unseen images.

## 4.2. FAIR Principles for Dataset Stewardship

In order to facilitate transparency, reproducibility and data reuse, a set of criteria known as the FAIR principles (Wilkinson et al., 2016) were defined as guidelines for best practices in dataset curation. FAIR is an acronym for findable, accessible, interoperable and reusable, which are the four core principles. In the following, we summarize each of these principles (following the treatment of Wilkinson et al.):

- **Findable:** Researchers should be able to locate the dataset. To facilitate this, a dataset must have a unique identifier that will always remain valid. Metadata and data associated with the dataset should make clear reference to this identifier. In addition, each dataset must be provided with considerable metadata regarding the activities which resulted in the dataset. The dataset and its metadata should be stored on a suitably accessible server that provides adequate facilities to search and retrieve relevant data.
- **Accessible:** Data and metadata should be accessible by some standardized protocol which is open, free, and implementable by any party. If necessary, authorization for access to the data should be part of the system. Metadata should always remain accessible, even if the data itself has been removed.
- **Interoperable:** Data and metadata should be represented using a common knowledge representation approach which is designed with FAIR principles in mind. If necessary, data and metadata should be allowed to refer to other datasets, with this reference annotated as necessary.
- **Reusable:** The metadata should be detailed and accurate to the highest extent possible, taking into account all factors which help describe precisely how and why data was acquired. This also includes other aspects of provenance information such as the person performing the experiment. The conditions for reuse (licensing) of the data should be made clear and be easy to access. Finally, the data and metadata should conform to generally accepted standards within the relevant research community.

FAIR principles do not by themselves imply a particular design or implementation choice. Rather, they are a means by which the relative tradeoffs of different design choices can be evaluated with respect to their impact on potential users of the dataset.

Possibly the biggest choice facing organizations is at which level of generality should they represent and make available their data and metadata. Should a software ecosystem be chosen that attempts to incorporate datasets from a large variety of subject fields? Or, should a system be chosen which specializes in the kind of information relevant to the organization's particular focus area?

Choosing a system that is too general means that organizations are left to specialize the system themselves, leading to a de facto lack of standardization. By contrast, an overspecialized system is

only of interest to a small set of organizations, limiting the broader reusability of the data made available through such a system. In the next section, we consider this question and how it relates to the field of plant phenotyping specifically.

Plant phenotyping encompasses a broad set of specializations and experimental methods. There is also great diversity in the kinds of organization operating in this field. Despite these differences, organizations working in this field share similar kinds of facilities, equipment, workflows, technical language and objects of study. The combination of a large target community with substantial shared interests suggests that a knowledge representation system focused on the plant phenotyping domain is viable.

### 4.3. Conclusion

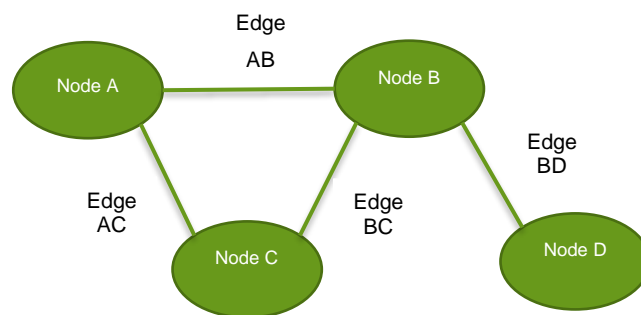
In this chapter, we discussed the high-level considerations involved in designing and adopting a system for dataset representation and sharing. The FAIR principles were outlined, which provide a framework for evaluating particular dataset management systems and the design choices they represent. As a prominent community effort in this field, PHIS is selected as the basis for the discussion in the remaining portion of the document. In the following chapters, the underlying concepts of knowledge representation in the PHIS system will be discussed, as well as providing details regarding interacting and manipulating information within a PHIS server.

## 5. Representing experiments and measurements using PHIS

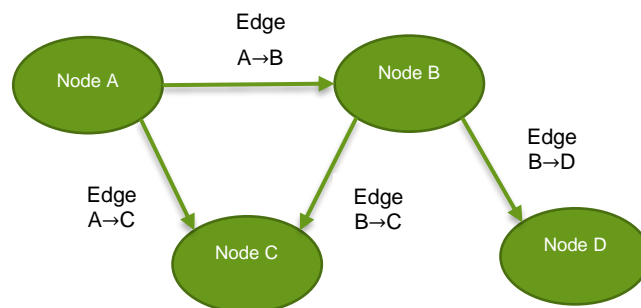
### 5.1. Semantic data representation

The essential problem of proposing a dataset standard is one of choosing an appropriate means of representing knowledge in datasets. The choice is dictated both by flexibility with respect to the kinds of data that can be represented (anticipating future improvements), but also the ability to impose structure where necessary to ensure compatibility between datasets recording similar kinds of data.

In PHIS (Neveu et al., 2019) and other systems using semantic web standards, the relationship between entities and their properties takes the form of a graph. A graph is a structure consisting of nodes and connections (edges) between nodes. Below is an example of a graph with four nodes and four edges that connect some of the nodes with each other.



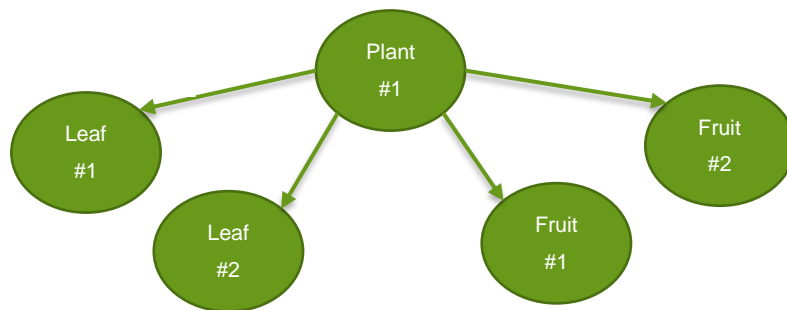
A directed graph is a graph where the connections between nodes have a direction such that each connection goes from one source node to a target node. In the following example, we take the previous example graph and make it a directed graph by adding a direction arrow to each edge.



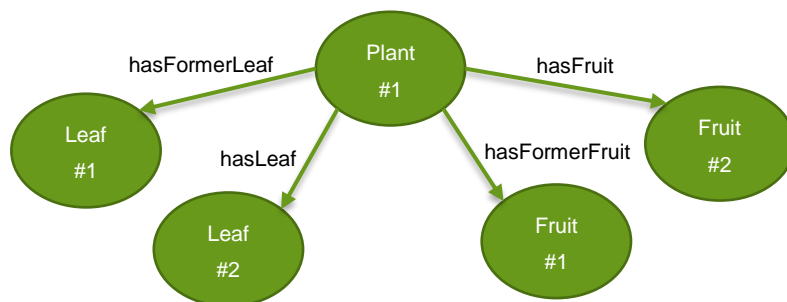


Let us say we want to record information about a number of distinct entities such as plants or fruits. We can represent each entity with an individual node in the graph, labelled with some name that uniquely specifies the entity. We can then record relationships between entities using edges, with the edges also labelled to indicate the kind of relationship we are expressing.

For example, we can represent the relationship between plants and their leaves/fruits by creating nodes for each plant, fruit and leaf. Following this we use directed edges to indicate that a given fruit or leaf comes from a particular plant. This is illustrated in the following figure.




We can illustrate other kinds of relationships between nodes, however we need a way of clarifying the kind of relationship each edge then illustrates. For example, while we intuitively think of the previous figure as showing the leaves and fruits being part of the plant, this need not necessarily have been the case. For sake of argument, we may be keeping track of the plant over multiple seasons, and we would like to make a distinction between fruits and leaves currently on the plant, and formerly on the plant. We can make this relationship explicit by labelling the edge as in the following example:



Seen from this perspective, it is clear that graphs provide a flexible means of representing entities, their properties and relationships. The question arises though how one should represent graphs themselves for easy manipulation by machines and also be readily understood by humans. Semantic web standards (RDF and OWL, discussed in annex 2) make use of an elegant approach that uses the basic insight that graphs are fully defined by the edges and their associated. We can represent directed edges from the above example as a series of “triples” as follows:

```
Plant1 hasFormerLeaf Leaf1
Plant1 hasLeaf Leaf2
Plant1 hasFormerFruit Fruit1
Plant1 hasFruit Fruit2
```

In this representation, each edge is declared in its own line using three identifiers. The first identifier is the source node. The second identifier is the edge label (property). The third identifier indicates the target node.

 Identifiers in real semantic web representations are referred to as URIs (Uniform Resource Identifiers). These often take a form similar to the URL of a web page (in fact, a URL is a special case of a URI). For example, PHIS might internally identify a project as **[http://www.opensilex.org/opensilex/tomato\\_mildew](http://www.opensilex.org/opensilex/tomato_mildew)**. A URI is meant to refer to a unique entity to ensure that its use always has the same meaning. PHIS internally names every entity uniquely in this way. We make use of this concept extensively in the following chapter.

This representation using triples underlies the core of semantic web functionality. Collections of triples are maintained in so-called “triple stores”. Triple stores can be anything from a file on a disk to a cloud server instance running a web service. Despite the differences between how triple stores function, they all fulfil the same function, that of managing a set of triples that represent knowledge in graph form.

## 5.2. Data representation in PHIS

In terms of data representation, PHIS provides two core functions. One is a triple store for storing dataset information using graphs. The second is a facility for storing files, for example images in PNG format.

The triple store is meant to be the primary means of structuring the data. Towards this, PHIS employs a set of ontologies specialized in describing experimental methods and plant phenotyping information. Storage of data as files is used mainly when dealing with data that is bulky and awkward to represent in a triple store. In addition, it may be desirable to store the original measurement files obtained from a scientific instrument for archival purposes.

In the following discussion, we will consider some of the core ontologies employed by PHIS and their respective functions. It is recommended that the reader use a software package such as Protégé (<https://protege.stanford.edu/>) to examine and explore the ontologies deeply. However, we provide links to the WebVOWL visualization tool to facilitate quick exploration of the ontologies.

### **5.2.1. Ontology of Experimental Scientific Objects (OESO)**

Perhaps most core to dataset management is the recording of the types of entities physically involved in experiments. Such entities might include individual plants or fruits, measurement devices such as cameras or physical locations such as greenhouses or plots of land. OESO provides a controlled vocabulary for expressing the relationships amongst such “scientific objects”.

For a more detailed look at the ontology, the reader can make use of the WebVOWL ontology visualizer:

<http://www.visualdataweb.de/webvowl/#iri=https://raw.githubusercontent.com/OpenSILEX/ontology-vocabularies/master/oeso.owl>

### **5.2.2. Ontology of Experimental Events (OEEV)**

While OESO describes the physical objects involved in an experiment, OEEV describes interactions that these objects have over a period of time. Such events could include treatments that plants undergo (such as potting, irrigation and sampling), physical movement of pots, calibration events of measurement devices, or undesirable events such as a sensor breakdown.

Like OESO, the OEEV ontology can be visualized using WebVOWL at:

<http://www.visualdataweb.de/webvowl/#iri=https://raw.githubusercontent.com/OpenSILEX/ontology-vocabularies/master/oeev.owl> .

### 5.3. The PHIS software ecosystem

While the core function of PHIS is to enable the representation of datasets, the larger system consists of a larger number of supporting components which enable easier access to the core functions.

Before commenting on the internals of PHIS, it is important to discuss how the system interacts with the outside world. This occurs through a layer of web services that PHIS exposes. The web services abstract complex internal operations into a relatively simple set of operations exposed to the outside world, accessible over a network. The following figure illustrates this approach.

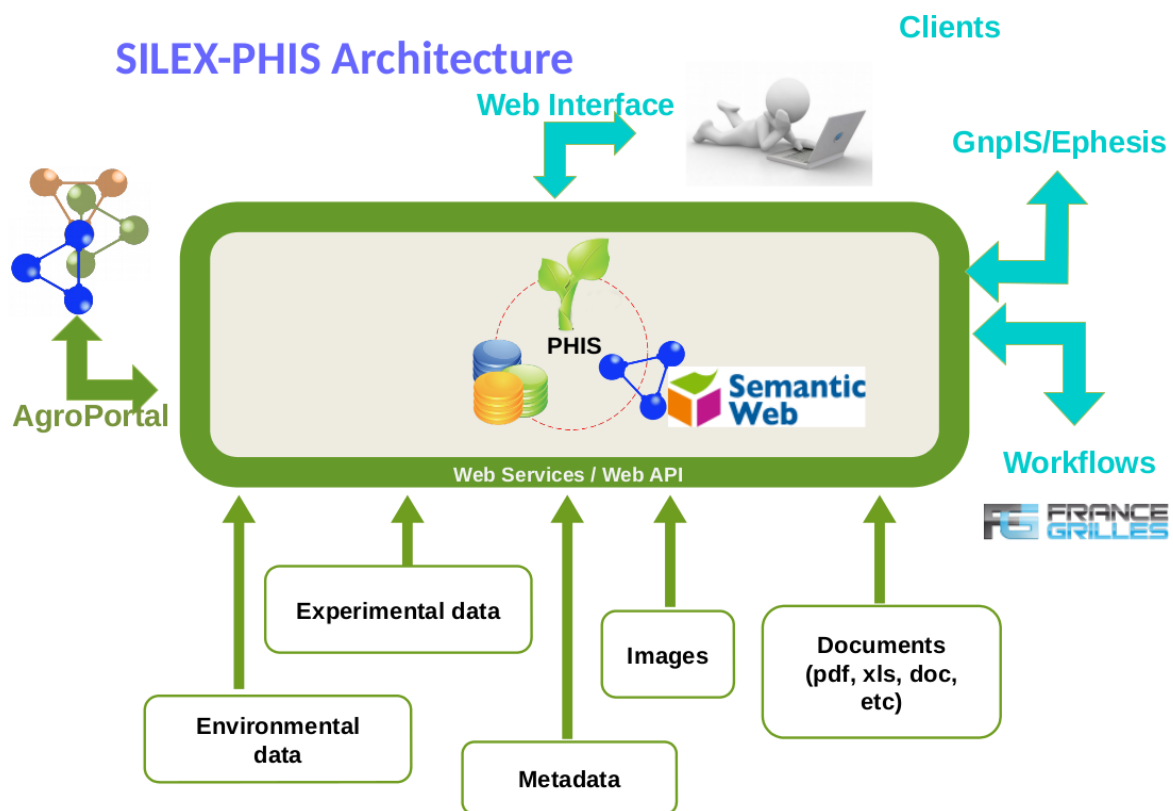


Figure 3: OpenSILEX/PHIS architecture<sup>4</sup>

It should be noted that PHIS itself is agnostic towards the kinds of external applications / services that may interoperate with it. One key example is the PHIS web interface, which we will discuss

<sup>4</sup> Source: <https://opensilex.github.io/phish-docs-community/> (Boizet et al., n.d.)

shortly. This interface enables the setup of experiments and the manual uploading of certain kinds of data (such as spreadsheets). Non-developers would typically make use of this interface.

For more complex operations such as recording sensor output automatically, software developers can write custom code that interact with PHIS using the web service functionality of their languages, making the system relatively language independent. We will discuss the basic principles of performing such operations, using Python as an example language for this purpose.

A more detailed look at PHIS can be found in the following figure:

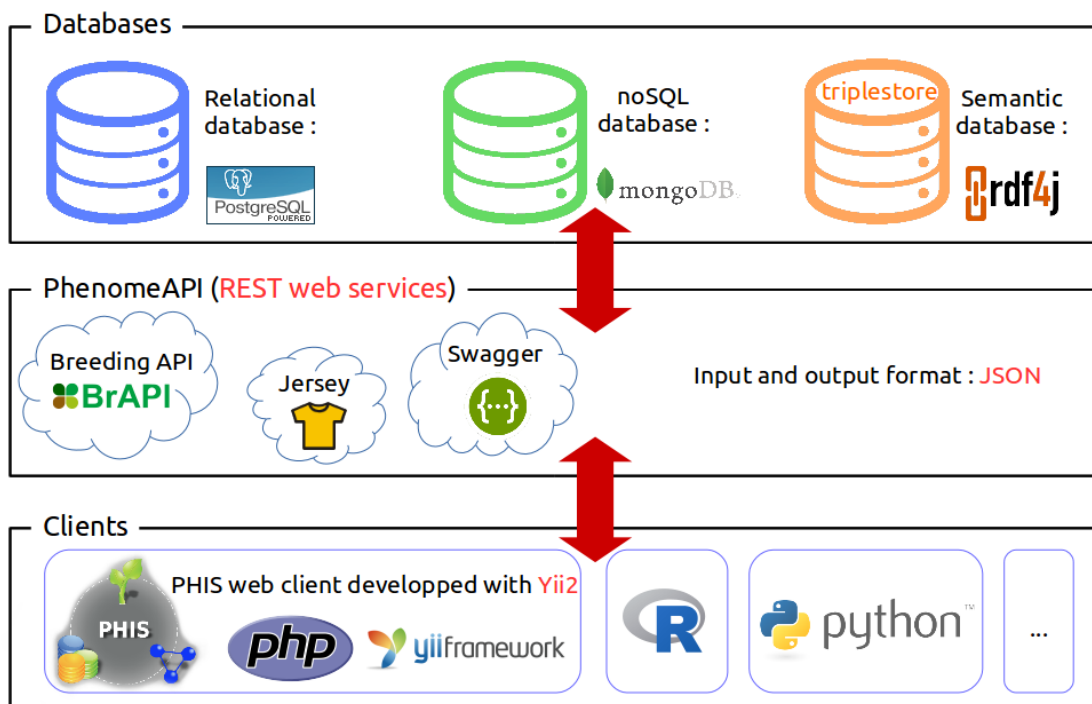


Figure 4: Division of concerns in PHIS<sup>5</sup>

<sup>5</sup> Source: <https://opensilex.github.io/phis-docs-community/> (Boizet et al., n.d.)

Notice again that clients such as the web interface and programming languages like R or Python interact with the databases using the exposed REST web services. We will return to the web services shortly, but note here the internal databases employed by PHIS within which the data is represented. Until recently, these included PostgreSQL (typically used for storing and manipulating tabular data), MongoDB (a more freeform “NoSQL” database), and rdf4j (a triple store). At the time of writing, however, developers report that PostgreSQL is phased out in new versions of PHIS, while rdf4j is being replaced by GraphDB (<http://graphdb.ontotext.com/>) as triple store.

The representation of datasets is distributed over these subcomponents, but the clients are insulated via the web services from dealing with these directly.

#### 5.4. PHIS Dataset Structure

Entering a dataset into PHIS consists of creating a hierarchy of objects, largely in the triple store, which describe increasingly fine detail of a project’s organization and its associated measurements. In the subsequent discussion, we describe each level of organization broadly.

The highest level of organization for a project is, naturally, a “project” entity. Project entities are manipulated using the “projects” web service. Project entities carry metadata such as the name of the project, whether it is a subproject of another project, its description, its financial support, running time, and so forth.

Experiments are the next level of organization (although an experiment may be part of more than one project). The “experiments” web service is used to interact with these entities. Metadata associated with an experiment includes its objective, location, running time, crop species and contact persons.

There exist a number of entities types that capture different aspects of a given experiment. These include, amongst others, “scientific objects”, “sensors”, “vectors” and “actuators”.

Scientific objects are the subjects of study and data measurement, such as plants, fruits and leaves. To help track scientific objects being measured, it is possible to describe one as being part of another. For example, a given a set of fruits might be linked to a particular parent plant. Furthermore, scientific objects can be given a location.

Sensors are objects which gather data. Within PHIS, sensor entities are associated with descriptions of their capabilities. For example, an entry for a camera might note its resolution and frame rate.

Vectors are entities which transport other items. Trolleys and sample trays are possible examples of transportation vectors. Other examples of vectors are setups which carry sensors. For example, a robot arm might move a camera around a given plant.

Actuators are items which manipulate the world in some way. Examples might include heating elements and sprinkler systems.

Ultimately, PHIS is intended to capture data. PHIS makes a distinction between two kinds of data. The simplest kind of data are files, which are simply uploaded to the database with metadata such as the item which was measured (usually a particular scientific object). This service is particularly suited to bulky data, such as images or videos.

It should be noted that, while PHIS allows one to add metadata describing the contents of the file, PHIS does not, in general, provide interpretive facilities for the contents of such files. Interpreting files downloaded from the PHIS store is left up to the client application. The web client, for example, provides some visualization capabilities with respect to image data.

Alternatively, PHIS supports capturing data with a more detailed level of imposed structure. Examples of such data might include tabular data from a spreadsheet, or continuously submitted data from a temperature sensor. The aim of this subsystem is to capture data where each measurement is some single value (a number or a string). Images, for example, are not appropriate for this subsystem.

PHIS frames this task as one of defining variables for which data may be submitted, capturing aspects such as the sensor, scientific object and time of measurement. Variables themselves can be decomposed into the trait being measured (such as weight), the method of measurement (such as a

laboratory scale) and the unit of measurement (such as grams). In this way, apart from the actual value, each measurement carries with it a detailed description of how it was obtained.<sup>6</sup>

PHIS defines a number of other entities. Users are one such entity type, used both for access control of the database, but also as metadata regarding who is responsible for a project or measurement process. Provenance entities record the high-level source of a particular aspect of a dataset. For example, an experiment might include multiple provenance entities where the description of what measurements were performed at a particular stage are recorded. Alternatively, provenance entities may indicate some other source of data, for example a dataset associated with a paper or the algorithm used to process images into refined data.

## 5.5. Conclusion

In this chapter, we provided a summary of the semantic web concepts underlying systems like PHIS. We then discussed the software ecosystem forming part of PHIS. Finally, we provided a high-level overview of how PHIS structures a given dataset. This provides an intuitive understanding regarding the nature of the services PHIS provides, as well as the mechanisms via which these services are implemented. Note that the interested reader may refer to the annexes for a detailed look at PHIS web services and how to upload a dataset, including an example using a case study dataset. So far, we have discussed capturing data within PHIS. In the subsequent chapter, we discuss the extraction of information from PHIS for machine learning purposes, and make suggestions and recommendations with respect to further expansion of this functionality.

---

<sup>6</sup> At the time of writing, developers report that recent changes to PHIS has changed the trait-method-unit model of variables to a quality-entity-method-unit model to conform with ontologies of measurement (OM or QUDT).



## 6. Machine Learning with PHIS Datasets

PHIS aims to provide a single, definitive record of all information generated during the course of experiments. This includes, for example, metadata meant only for human consumption, and potentially erroneous data due to sensor malfunction (along with annotations indicating such problems).

By contrast, machine learning tasks related to such experiments might use only a subset of the measured data. Part of a machine learning task is to distil the relevant data into a format convenient for use by machine learning libraries. Data cleaning decisions related to faulty data need to be made during the process.

For the purpose of reproducibility, it is critical that researchers have access to the original dataset employed during a machine learning experiment. However, it is clear that there exists a gap between the relatively complete representation provided by PHIS, and the narrower representation a particular machine learning task requires.

In the rest of the chapter, we make recommendations around machine learning experiment reproducibility, based on an already existing PHIS representation of the wider experiment. We begin the discussion by considering the format requirements of machine learning frameworks, with a special emphasis on deep learning approaches.

### 6.1. Structure of machine learning datasets

In this section, we consider how machine learning methods represent training and test data. It should be emphasized that this discussion is by no means complete, machine learning is enormously diverse. The approach we discuss is certainly one of the most popular, and is used in approaches ranging from “classical” machine learning to advanced deep learning models.

The core structure common to most machine learning methods is the tensor. The term tensor is used synonymously with multi-dimensional array in this context. We begin our discussion by discussing an example dataset and the role tensors play in representing it. An example from the well-known Iris dataset follows. In this example, we try to predict the species of Iris from different

measurements (in centimetres) taken of a particular flower. Each of these measurements is referred to as a feature. Each row represents an individual flower (or sample).

Sepal length	Sepal width	Petal length	Petal width	Class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica

In order to perform training (and subsequently testing), we represent this dataset in memory as a matrix and vector respectively. The inputs (which we label  $X$ ) we represent as a matrix that has  $N$  rows and  $M$  columns, where  $N$  is the number of samples and  $M$  is the number of input features.

$$X = \begin{bmatrix} 5.1 & 3.5 & 1.4 & 0.2 \\ 4.9 & 3 & 1.4 & 0.2 \\ 7 & 3.2 & 4.7 & 1.4 \\ 6.4 & 3.2 & 4.5 & 1.5 \\ 6.3 & 3.3 & 6 & 2.5 \\ 5.8 & 2.7 & 5.1 & 1.9 \end{bmatrix}$$

The outputs (or class), which we label as  $y$ , is represented as a vector. The class labels are first converted to integer labels. Which label is assigned to which integer in principle does not matter, but typically one class has label 0, and others are added in increments of one. Here we choose Iris-setosa, Iris-versicolor and Iris-virginica should have class labels 0, 1 and 2 respectively.

$$y = [0 \ 0 \ 1 \ 1 \ 2 \ 2]$$

In the preceding paragraphs, we employed two special cases of the tensor, matrices and vectors, to represent the dataset. Similarly, more complex cases such as image sets, videos and point sets can be represented using tensors.

- “Spreadsheet” data: As in the case of the Iris dataset, one- and two-dimensional tensors are commonly used to represent input features and output targets.

- Image datasets: For colour images, four-dimensional tensors are commonly used. The four dimensions correspond to image number, channel number (red / blue / green), pixel X-position and pixel Y-position. Hyperspectral image sets are similarly represented, except these have more than three channels.
- 3D voxel models: These can be represented as images, except there is an additional Z-position. There may or may not be multiple channels, so the tensor may be four- or five-dimensional.
- Video datasets: Similar to image datasets, five-dimensional tensors may be used where the frame number is an additional dimension (see the following discussion on ragged tensors).

Tensors typically have a single number describing the number of entries along each dimension. For example, an image dataset with dimensions  $5 \times 3 \times 100 \times 100$  may represent 5 images that are 100-by-100 pixels with 3 colour channels (red, green and blue).

However, particularly in cases such time series, this fixed dimensionality may not always be the case. For example, an RGB video dataset of 5 videos at resolution 100-by-100 may be stored in a structure with dimensions  $5 \times F_n \times 3 \times 100 \times 100$  where  $F_n$  is the number of frames for a particular video. Depending on the video number ( $n$ ), the number of entries along the first dimension in the video tensor, the number of frames  $F_n$  may be different. This is known as a “ragged” tensor.

Standard tensors and ragged tensors account for a vast number of internal representations used in machine learning tasks. It would therefore be useful to be able to represent how a particular tensor was derived from a PHIS dataset, since this would aid transparency and reproducibility of results. In addition, this would allow automated assembly of tensors from such a description, simplifying the task of sharing and using machine learning datasets. We therefore focus on potential ways of enabling this functionality in the following sections.

## 6.2. Challenges to automating machine learning dataset assembly from PHIS

In the creation of a machine learning dataset, workers make a number of decisions in order to adapt raw data into data ready for training and testing of models. Some aspects that need consideration include:

- Data points of interest: Which data points are included in the task dataset? How are these data points partitioned into subsets such as training and test sets?

- Data cleaning decisions: Which data points are ignored, and why? What are they replaced with?
- Correspondence between data points: If an item is measured multiple times at different dates/times using multiple sensors, how do we associate the correct measurements with each other? If multiple measurements on one sensor correspond to single measurements on another sensor, how do we combine the multiple measurements? Or, do we do the reverse by associating the single measurement to each of the multiple measurements? One example of this would be to take the average of a number of temperature measurements to obtain a single daily temperature to correspond with a daily plant height measurement.

At least the following strategies exist for dealing with these needs:

- Make available preprocessing scripts written in a general purpose scripting language such as Python.
- Store and make available the results of preprocessing in an interchange format such as HDF5.
- Employ a description language to encode aspects of the early portions of data preprocessing, allowing automation of tensor generation.

Each of the above strategies, by themselves, have disadvantages.

Preprocessing scripts in general purpose language are not subject to dataset standardization constraints, leading to the danger of non-standard script outputs. Executing scripts often have complex behaviour, which may cloud the provenance relationships between the input and output of the script. Furthermore, scripts in a particular language may be difficult to read and critique by users not familiar with the language.

Storing the results of preprocessing is convenient, but without further effort this loses provenance information, since the contents of the tensor are generally not connected back to the raw dataset. Furthermore, large datasets that use compression (such as large collections of JPEG images) may be difficult to store and disseminate as raw tensors, since tensor exchange formats do not support all possible modes of compression.

A special purpose description language is a tempting option. However, while it may be possible to develop a higher-level representation to clarify dataset assembly in procedural terms, it is still

likely that special cases will resist representation in such a scheme. In the worst case, such a description language would acquire the power of more general purpose programming languages, leading to the same disadvantages.

However, it may be possible to combine the preceding strategies to obtain a satisfactory solution. One can allow the use of general programming language scripts, but then have such scripts leave “traces” that relate the contents of the output tensors to the raw data in the PHIS database. Along with the scripts, these trace files may be made available, containing both high detail provenance information and a means to automatically assemble a tensor. In the case where tensors are small enough for this to be practical, storing the preassembled result in an interchange format such as HDF5 would be an optional extra step.

The question arises whether PHIS’ existing facilities can be employed for such a scheme. Storing of scripts and interchange files is trivial using the bulk storage facility. However, the trace information containing provenance information should ideally form part of the descriptive framework PHIS provides. In the next sections, we clarify the idea of trace descriptions and we consider options for storing trace information in a FAIR manner.

### 6.3. Trace-based Tensor Assembly and Provenance Capturing

Trace-based descriptions are essentially a set of notes describing how a computational process evolved without actually describing the entire computational process itself. Such traces can be used to capture the essentials required to map one **class** of inputs to the more general computational process’ outputs, without representing the full complexity of the original computation.

One particularly prominent example of such a scheme is the approach of the PyTorch framework to supporting the ONNX format (a format for describing deep neural networks). Although PyTorch itself allows the structure of networks to be changed based on arbitrary Python computations, certain common types of neural networks can be saved in ONNX format through PyTorch “notes” (traces) of a particular computation. The neural network computation can then be recreated without recourse to the original Python script. We discuss the ONNX format in a later section of this chapter.


The most important underlying idea is that a more general computation can be replicated by creating a list of simpler instructions as the general computation takes place. This balances the

freedom of general purpose programming languages with the need for simple provenance and reproducibility information.

In PHIS, a data point is the combination of a value, provenance, scientific object, date/time, variable. The non-ambiguity is guaranteed by an assigned URI and the traceability by the provenance. This is elaborated on in Annex 4.8.1. Recall that provenance information is simply metadata describing the circumstances of the data collection, including (amongst other considerations) information about the experiment, the person performing measurements and the procedures followed.

Consider the following example of a simple dataset captured using the PHIS data point storage facility.

URIs	Provenance	Scientific Object	Date and Time	Variable	Value
URI1	PROV1	SENSOR1	2020-09-01 01:00	Temperature (°C)	13
URI2	PROV2	PLANT1	2020-09-01 10:40	Weight (g)	220
URI3	PROV2	PLANT2	2020-09-01 10:42	Weight (g)	200
URI4	PROV1	SENSOR1	2020-09-01 13:00	Temperature (°C)	20
<b>URI5</b>	<b>PROV1</b>	<b>SENSOR1</b>	<b>2020-09-02 01:00</b>	<b>Temperature (°C)</b>	<b>-80</b>
URI6	PROV1	SENSOR1	2020-09-02 13:00	Temperature (°C)	23
URI7	CORRECTIONS	SENSOR1	2020-09-02 01:00	Temperature (°C)	21

 PHIS uses the combination of provenance, scientific object, date/time and variable to uniquely identify a data point. Data points currently have URIs, but GET services do not currently allow searching for data points with specific URIs. Nevertheless, we assume that data points can be retrieved based on their URIs, as this simplifies the discussion and would be simple to add in future versions of PHIS.

Consider creating a spreadsheet, and subsequently a tensor, based on this table. Using for the moment the temperature at 13:00 as the daily temperature, we obtain

Plant	Weight (g)	Temperature (°C)
PLANT1	220	20
PLANT2	200	20

The tensor containing the weight and temperature features would be the matrix

$$X = \begin{bmatrix} 220 & 20 \\ 200 & 20 \end{bmatrix}$$

However, obtaining this tensor would involve making a number of calls to the PHIS web services and other complex computational operations. However, recalling that a URI uniquely identifies a data point, one can easily reproduce the above matrix if we store a trace of the source datapoints for each element in the matrix.

URI2	URI4
URI3	URI4

With this table, it is trivial to automatically obtain the matrix  $X$ , since the individual elements needed to uniquely identify the source data points are given for each element of the matrix. As an added advantage, the provenance of each element in the matrix is known explicitly in its entirety, because all data points have a reference to their provenance.

If a direct one-to-one datapoint mapping is too simple for a particular situation, certain common multi-datapoint mappings can be defined. For example, taking the average of a daily temperature of the first day could be represented as

#### **AVG URI1 URI4**

A vector of the average daily temperatures could then be represented as

AVG URI1 URI4
AVG URI5 URI6

However, the temperature  $-80^{\circ}\text{C}$  (URI5) in the example is incorrect (highlighted in red). Based on a decision we replace the value with 20 (inserted as a datapoint in PHIS with URI7), we can indicate the correction as follows.

AVG URI1 URI4
AVG URI7 URI6

Because a special provenance (CORRECTIONS in this case) has been provided, a human readable accounting for the correction can also be provided attached to the provenance entity.<sup>7</sup>

This example shows that the trace approach is itself simple. The tensor creation script contains all of the “intelligence”. The script would have had to query for value corrections using the web services and finding corresponding values from different provenance entities. The simplicity of traces is intentional, as the end user should be able to easily reconstruct the tensor and trace back the provenance of values.

The idea of a trace-based description is not to provide facilities for any possible type of preprocessing, but rather to facilitate tracking of the most fundamental construction of initial data tensors. While it may seem cumbersome at first, for spreadsheet-type data the amount of extra storage required is similar to that of the PHIS representation of the data itself (each element in the trace has a corresponding data point in the PHIS representation). Spreadsheets, however, tend to be compactly represented, and so doubling of the amount of storage required is likely not a significant disadvantage.

For bulk data tensors, many of the entries can be handled implicitly by seeing a file in PHIS bulk storage as a subtensor of the larger tensor, which dramatically lowers the storage requirements for the most common types of large tensors. For example, the trace representing an image dataset may be a simple list of URIs referring to images in bulk storage.

Having discussed the essentials of trace-based tensor descriptions, we list important advantages of such frameworks:

- Trace-based annotations as contemplated above are fundamentally based on references between entities, which is compatible with semantic web representations. This may aid incorporation of such structures into future versions of PHIS.
- Because data points are explicit references, rather than implicit queries, a tensor derived from a trace-based description will never change, without recourse to administrative rights, given PHIS’ current design. This is because data points or bulk data files in PHIS cannot be changed once created. Defining tensors implicitly based on queries may hamper

---

<sup>7</sup> Another option would be to define a REPLACE trace operation, which lists a URI, an explicit replacement value and a human readable comment. However, we proceed from the position that including as much metadata within PHIS is the preferable solution.



reproducibility, because the addition of data points and other queryable entities can change the result of queries relative to past execution of the same queries. Even if the script that produced the trace description produces a different trace in future, a stored trace description can be compared with the new trace, and even subtle changes can be located exactly with full provenance information.

- After a script in one language has produced a trace-based description, generic tensor reconstruction scripts in other languages can easily assemble the tensor based on this description. This is true even if the original script performed complex operations and queries to form the description. This allows other researchers to easily reuse a tensor in their language of choice, and also critique its construction through reference to provenance information. It also avoids potential sources of error by enabling the shared maintenance of tensor assembly libraries amongst different groups.

#### 6.4. Roadmaps towards incorporation of trace-based provenance in PHIS

It would be technically possible to encode trace-based tensor provenance using existing facilities. Entities that may prove useful in such an encoding process include provenance, annotation, data point and bulk data entities. However, such approaches would be awkward given that the PHIS services have not been designed with trace-based tensor definitions in mind.

Instead, a path towards incorporating trace-based tensor descriptions into PHIS datasets would need to be adopted. A number of options towards this goal exist.

The most immediate and flexible means of describing tensors is to define a file format for recording such traces. This path to adoption has the advantage of starting from a clean slate, since direct integration into PHIS facilities is not required. The disadvantage of such an approach is that this leaves trace files opaque to PHIS facilities in the near term.

Another path to implementation may be to extend the ontologies related to PHIS. Following this, specialist web services may be implemented to help with the storage and interpretation of traces. Such web services may include facilities for serving subtensors to cloud instances working on separate portions of the training data, which would assist high-performance computing approaches to deep neural network training. This is, however, substantially more complicated because of simultaneous work on integration with other PHIS services.

Because these facilities would represent a further extension of the PHIS ontologies into the domain of describing machine learning workflows, it may be advisable to use the opportunity to bridge into existing ontologies for such workflows. A proposed ontology called ML-Schema has recently been developed to incorporate a number of separately developed ontologies for this purpose into a single framework. ML-Schema is discussed later in this chapter.

It should be noted that ML-Schema does not support trace-based tensor descriptors, but does define concepts such as datasets and feature vectors. Therefore, investigating the viability of using some of ML-Schema's facilities might still be a worthwhile avenue of exploration.

## 6.5. Compute considerations

Trace-based tensors explicitly linking data through URIs will impact the required amount of compute and storage resources in several ways. URIs are string-based data structures which require much more storage than the value it represents. In those cases additional storage needs to be allocated to store the tensor. When the URIs link to full images trace-based tensors will show a significant drop in required storage where only during computations the (part of) the image is retrieved and stored in memory.

However, retrieving these data through the PHIS API will result in this API being a bottleneck of the computations. To facilitate this, two options are available. First, PHIS itself will run on a large compute node with the underlying databases on their own compute nodes. This way data retrieval will be distributed across several nodes, making it faster than running everything on a single node. The second option is to move the required data to a separate instance of PHIS, or even use load-balancing technologies and running several instances of PHIS. This works very well for data retrieval: after collection the data and assigning them URIs, the data will not change. Hence is it irrelevant where the data come from. This load balancing approach is more complicated when we allow people to add trace-based tensors, although it would be possible to send these requests to a single instance.

When storing tensors in PHIS it will also become possible to send the tensor calculations to PHIS and either report the results back to the ML application or add them to PHIS. By following this principle of sending the computations to the data instead of the other way around much less data will go over the network. Additional, specific API calls and computational models can be added to PHIS to

further speed up the analysis. The API then accepts a ML method and trace-based tensors to start calculations. Which can be performed on high-performance hardware such as GPUs.

## 6.6. Trace-based tensor description using JSON

As mentioned earlier, one option towards trace-based tensor description in PHIS is to define a new file format for interim handling of this functionality. In this section we propose a simple approach that uses JSON to represent traces. This has the advantage of being easy to generate by scripts in multiple languages, but are also inherently friendly for future incorporation into web services.

Note that the reader may safely opt to skip this section if implementation details are not of interest to them.

JSON directly supports the representation of multi-dimensional arrays. For example, the matrix

$$X = \begin{bmatrix} 220 & 20 \\ 200 & 20 \end{bmatrix}$$

is represented using nested arrays as

**[[220, 20], [200, 20]]**

Ragged tensors can also be represented since there is no requirement that nested arrays be the same length as others on the same level of the hierarchy. The following is an example of a ragged tensor

**[[220, 20], [200, 20, 30]]**

To enable the representation of trace-based tensor descriptions, we need three additional features:

- Referring to data points or bulk files in the PHIS database, instead of using literal numeric values. These can be used to download individual data points as tensor elements, or bulk data files as subtensors.
- Allow simple operations such as averaging to be performed.
- Attach high-level metadata that applies to the whole tensor or subtensors, such as hints around tensor dimensionality.

These features can be implemented using JSON objects. A data point<sup>8</sup> may be represented as

---

<sup>8</sup> We identify point data using a combination of provenance, scientific object, date/time and variable. As an alternative, the single data point URI might be used, but recall PHIS web services currently do not support queries searching by a data point URI.

```
{ "type" : "dataPoint",  
  "provenanceURI" : "...",  
  "scientificObjectURI" : "...",  
  "datetime" : "...",  
  "variableURI" : "..." }
```

Here we specify the type of PHIS data service (point data) and the four elements necessary to uniquely identify a given data point in PHIS.

A bulk data file can be represented using

```
{ "type" : "bulkData",  
  "dataURI" : "...",  
  "loader" : "..." }
```

Here we specify the PHIS data service (bulk data) and the URI pointing to the bulk data file's entry. We also specify a loader function to be used in loading the data. A set of file loaders may be specified as part of a standard. For example, we might specify "rgblmage", and support automatic loading of common image formats such as PNG, JPEG and TIFF.

It should be noted that, with this information, the individual data points or bulk data files are trivial to download from PHIS.

Operations could be specified using a JSON object. For example, the following specifies an averaging operation over a number of data items

```
{ "type" : "operation",  
  "operation" : "average",  
  "tensorData" : [ ... ] }
```

Finally, we need to be able to specify high-level metadata. We can do this by nesting tensors within a higher-level JSON object, containing this metadata. For example, at the highest level, we might have

```
{ "type" : "metadata",  
  "scriptURI" : "URIToScript",  
  "scriptExecutionDate" : "...",  
  "shapeHint" : [3, 2, 5],  
  "tensorData" : [[...], [...], ... ] }
```

which contains a URI pointing to where the original script that generated the trace is publicly available. There is also the time at which the script was executed (this can help in situations where a script's output changes between executions). The shape hint would be an optional hint denoting what the dimensionality of the tensor should be expected (in this case  $3 \times 2 \times 5$ ). There is also a field for storing the tensor itself, where each data point or bulk data file is stored as a single element JSON object within the multi-dimensional JSON array.

Often the situation arises where one would like to apply a property to all nested objects in the hierarchy. Therefore, one could add the property `descendantsInherit` to specify that a particular set of properties is inherited by all nested JSON objects within `tensorData`.

```
{ ...
  "descendantsInherit" : [ ... ],
  ...
}
```

This allows us to specify properties such as the provenance within a high-level JSON object, and let all nested data elements inherit this provenance. Descendants may then optionally override values, but this allows the bulk of data points to share certain attributes

We now present a worked example showing a possible encoding of a simple point-data tensor. We reuse the table presented earlier in the chapter (restated here for convenience)

URIs	Provenance	Scientific Object	Date and Time	Variable	Value
URI1	PROV1	SENSOR1	2020-09-01 01:00	Temperature (°C)	13
URI2	PROV2	PLANT1	2020-09-01 10:40	Weight (g)	220
URI3	PROV2	PLANT2	2020-09-01 10:42	Weight (g)	200
URI4	PROV1	SENSOR1	2020-09-01 13:00	Temperature (°C)	20
URI5	PROV1	SENSOR1	2020-09-02 01:00	Temperature (°C)	-474
URI6	PROV1	SENSOR1	2020-09-02 13:00	Temperature (°C)	23
URI7	CORRECTIONS	SENSOR1	2020-09-02 01:00	Temperature (°C)	21

We wish to represent the following matrix

Plant	Weight (g)	Temperature (°C)
PLANT1	220	20
PLANT2	200	20

The structure of the resulting tensor was as given below

URI2	URI4
URI3	URI4

We can now represent the above tensor using a JSON object as follows

```
{ "type" : "tensorDescription",
  "scriptURI" : "http://opensilex.org/opensilex/d00001",
  "scriptExecutionDate" : "2020-10-03T13:15:33.143Z",
  "shapeHint" : [2, 2],
  "tensorData" :
    [
      [
        { "type" : "dataPoint",
          "provenanceURI" : "PROV2",
          "scientificObjectURI" : "PLANT1",
          "datetime" : "2020-09-01T10:40:00.000Z"
          "variableURI" : "WEIGHT" },
        { "type" : "dataPoint",
          "provenanceURI" : "PROV2",
          "scientificObjectURI" : "PLANT2",
          "datetime" : "2020-09-01T10:42:00.000Z"
          "variableURI" : "WEIGHT" }
      ],
      [
        { "type" : "dataPoint",
          "provenanceURI" : "PROV1",
          "scientificObjectURI" : "SENSOR1",
          "datetime" : "2020-09-01T13:00:00.000Z"
          "variableURI" : "TEMPERATURE" },
        { "type" : "dataPoint",
          "provenanceURI" : "PROV1",
          "scientificObjectURI" : "SENSOR1",
          "datetime" : "2020-09-01T13:00:00.000Z"
          "variableURI" : "TEMPERATURE" }
      ]
    ]
}
```

```
} ] ]
```

Because this matrix is small, we have not used inherited values, as they are more useful for large matrices.

Note that each data point entry can be directly queried from the PHIS server, as the information needed to retrieve a unique data point is available. This makes assembling the resultant texture a simple matter. At the same time, full provenance information is available for each value.

Bulk data datasets may be even simpler. For example, a tensor descriptor for an image dataset may consist simply as a list of **bulkData** elements.

In the next section, we consider where in the machine learning workflow the generation and usage of the trace file belongs.

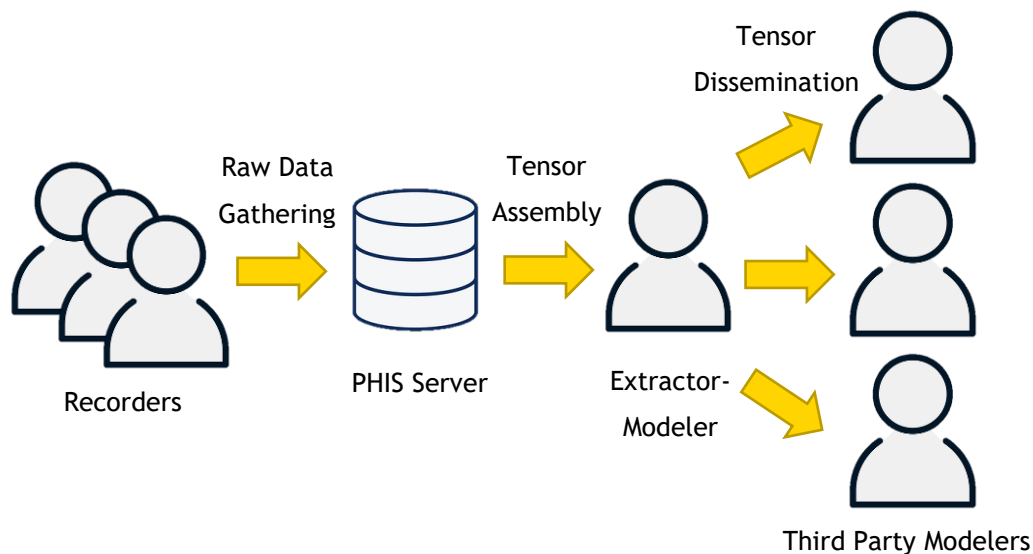
## 6.7. Roles in data science workflows

At least the following roles can be identified in the larger dataset preparation and usage workflow

- Recorders of the raw dataset
- Extractors of tensors suitable for a machine learning experiment
- Modellers using the extracted tensors for modelling purposes

An actor (person or other entity) may be responsible for more than one of these roles.

However, typically in a large experiment the recorders of the dataset are a team of observers and technical personnel performing measurements and storing them in some format (possibly in a system such as PHIS). There is usually at least one extractor-modeller, responsible for a first analysis of some subset of the larger raw dataset. The extractor-modeller writes scripts to process the raw dataset into tensors suitable for their experiments, and also performs some modelling on these tensors. Subsequently, if an extractor-modeller shares the tensors in some way (either explicitly or via scripts/data). Third-party modellers may then perform their own experiments on the pre-existing tensors, possibly comparing their results afterwards.



The FAIR principles place some constraints on how each of these actors fulfil their roles. PHIS provides an existing framework for dataset recorders, and for the purposes of this discussion, we presuppose the existence of a PHIS dataset with raw data and provenance information.

Recorders are not the appropriate entities to specify tensors for machine learning purposes, as they may not be familiar with what subset of data or preprocessing of data that is required for the machine learning experiment(s). Our discussion focusses instead on the interplay between the roles of extractor-modeller and third party modellers.

In principle, any third party modeller with access to the PHIS repository and adequate descriptions of a tensor's assembly can replicate the process. However, this may not be a simple operation, especially for large datasets. Furthermore, mistakes made either by the extractor-modeller or the third party modellers may result in potentially subtle errors in tensor assembly, leading to problems with reproducibility of experiments.

FAIR principles encourage the minimization of burden on third party modellers. An extractor-modeller could resolve this by exporting a tensor descriptor as part of their workflow. To facilitate later sharing of tensor descriptors, a modeller should separate out tensor definition and tensor usage in their workflow.



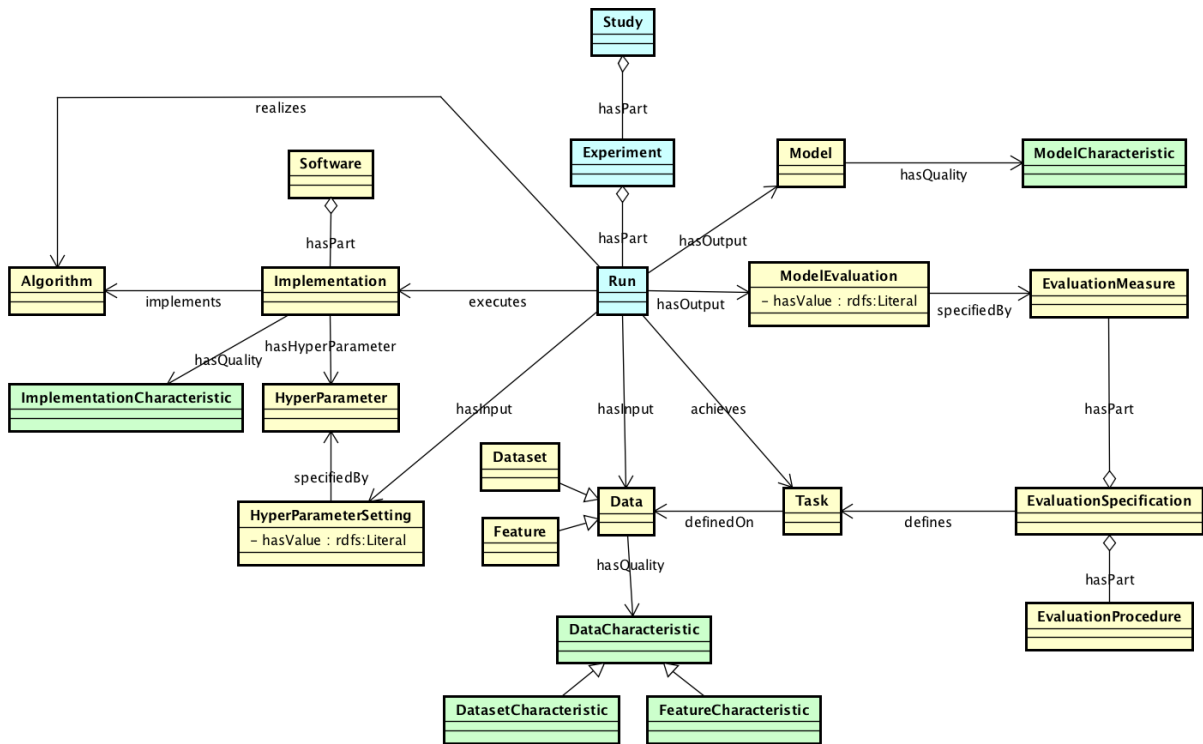
By first generating a tensor descriptor, and using that to assemble the tensors for machine learning purposes, this makes sharing of tensors with third parties simple. It also means that it is guaranteed that the extractor-modeller's training and testing scripts and those of third party modellers start with the same data. Furthermore, this is also to the extractor-modeller's advantage, as this allows easy reuse of scripts on different datasets for similar tasks.

In the preceding section, one potential representation of tensors has been discussed. This representation is not intended to be complete, but to present one potential avenue for exploration in the dissemination of tensors with full provenance information. In the following sections we consider other aspects of disseminating aspects of machine learning projects.

## 6.8. ML-Schema

The focus of the current document is on the storing, annotation and distribution of datasets in a FAIR compliant way. In the preceding discussion we did, however, note that the needs of machine learning modelling processes are a key consideration in easing the reuse of datasets. It is therefore worthwhile to note efforts to easily interchange machine learning experiments themselves.

From a semantic web point of view, ML-Schema (Publio et al., 2018) is probably the most prominent example of a machine learning ontology, combining several earlier ontologies for modelling aspects of machine learning. Figure 5 shows a graph visualizing the core concepts captured by this ontology.



**Figure 5: ML-Schema core vocabulary**

As can be seen from the figure, ML-Schema attempts to capture a broad subset of machine learning-ready data and metadata. This complements PHIS' approach, which is much more focused on the raw data and its metadata.

It is important to note that ML-Schema includes facilities for defining datasets, emphasizing the needs of machine learning experiments. This may be useful in standardizing how machine learning datasets are extracted from PHIS databases. It is therefore recommended to consider the possibility of exploiting this overlap in dataset management between PHIS and ML-Schema, to ease the automated extraction of data from PHIS for machine learning purposes.

However, the current focus of ML-Schema is on tabular data. Therefore some extensions would be needed to handle the higher-dimensional tensors common in deep learning applications.

## 6.9. Trained machine learning model interchange

While the FAIR principles mostly revolve around the availability of input data, the parameters of machine learning models (after training) are also an important kind of data which could be

disseminated. In this section, we briefly discuss ONNX or the Open Neural Network Exchange format (<https://onnx.ai/>), an interchange format for deep neural networks.

There are a number of deep neural network frameworks in use today, including PyTorch, TensorFlow, Keras and others. Each of these frameworks has its own internal representation of a neural network and its parameters. They also usually have their own means of saving these representations and parameters. In general, the format of the saved networks / parameters are not exchangeable between frameworks.

ONNX attempts to alleviate this problem by providing a standardized way of representing a large class of deep neural networks and their parameters. By doing this, it is possible for workers to exchange neural networks between frameworks. For example, a researcher might experiment in PyTorch, and transfer the resulting neural network to TensorFlow for final deployment.

Deep neural networks are highly diverse in their operation, but can largely be broken down into basic building blocks. These building blocks are individual operations on tensors that produce other tensors. ONNX provides a way of defining which tensor operations are present in a deep neural network, and how these operations connect with each other to form a network of operations. Such a network of operations is called a computational graph. A computational graph shows how information flows through a deep neural network and the operations performed on the information at each point in the network.

ONNX provides a representation for such computational graphs. It focusses on deep neural networks where information always flows forward through the network (feedforward neural networks). This does not cover all neural networks, networks where information can flow backwards (recurrent neural networks) also exist. However, feedforward neural networks represent a large portion of state of the art methods. Furthermore, recurrent neural networks can often be “unfolded” into feedforward networks for particular classes of input. Therefore ONNX can represent an important subset of deep neural networks.

ONNX has a good level of adoption in industry (including IBM, Amazon, Intel, Nvidia, Facebook, Mathworks and Microsoft), as well as support for a number of popular deep learning frameworks (including PyTorch, Caffe2, Matlab, Tensorflow and Keras). This makes ONNX attractive as a format for adoption for exchange of deep neural networks in plant phenotyping experiments.

## 6.10. Summary and conclusion

In this chapter, we considered the challenge of exchanging tensor data for machine learning experiments. Towards this, trace-based descriptors were proposed as a possible vehicle for tensor exchange, as this combines easy reconstruction of tensors with full provenance information regarding elements of these tensors. We also noted the possibility of using ML-Schema as a means of extending the PHIS ontology to handle machine learning dataset considerations in a more explicit way. Finally, we discussed ONNX as a means of exchanging trained deep neural networks.

## 7. Summary and Conclusion

In this document, we have considered the challenge of dataset management from the perspective of the plant phenotyping domain.

A mapping/gapping analysis was presented where we found that most systems are able to process images with their software tools, it is expected that flexibility in these tools towards new plants/new situations is limited and AI/ deep learning will offer that flexibility. Therefore the main gap in knowledge how to integrate these AI tools into their data pipeline, and how to prepare datasets to be able to make use of the rapidly emerging AI tools.

We discussed how plant phenotyping computer vision research is situated within the larger general computer vision field. In particular, we noted the implications of large, commonly used datasets for choices made in more specialized fields like plant phenotyping. Examples of such general datasets were provided, along with corresponding cases within the plant phenotyping field.

Subsequently, we considered foundational principles underlying good dataset management. We emphasised the FAIR (findable, accessible, interoperable and reusable) requirements as guides for such management activities.

Following this we discussed PHIS (phenotyping hybrid information system) as a prominent community effort for dataset management in the plant phenotyping domain. This discussion included a look at semantic web concepts, as well as the software ecosystem PHIS builds on these standards to represent data and metadata regarding projects and experiments.

Finally, we considered the interface between PHIS' representation of data and subsequent use in machine learning (particularly deep learning) experiments. We noted that tensors are the typical format expected by such experiments, and made suggestions regarding how one could automatically assemble tensors from PHIS datasets without losing provenance information while avoiding relying on general programming language scripts. The possibility of extending PHIS' internal representations to include information for machine learning experiments was considered, with lessons learned from representations such as ML-Schema being one potential source to draw on. Lastly, we discussed ONNX

as a means of disseminating trained deep learning models such that the resulting models of deep learning experiments remain available for reuse and peer review.

Dataset management remains, and will likely remain, an unsolved problem. It is expected that developments within computer vision and plant phenotyping in particular will continuously place new requirements on dataset management systems. However, adopting an extensible, standardized system for performing such management creates a route by which the plant phenotyping community can share the responsibility of responding to such changes in an interoperable way.

As an existing system specialized in plant phenotyping, PHIS is a particularly attractive option, given that it has already been used in large-scale experimental facilities. It can therefore be recommended for adoption and further development by other facilities.

## References

- Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist. Semantic Web for the Working Ontologist*. Elsevier Inc. <http://doi.org/10.1016/C2010-0-68657-3>
- Banakar, A., Polder, G., Ruizendaal, J., & Balendonk, J. (2019). *SpectralCam evaluation dataset and experimental report*. Wageningen University and Research.
- Barth, R., IJsselmuiden, J., Hemming, J., & Henten, E. J. V. (2018). Data synthesis methods for semantic segmentation in agriculture: A Capsicum annum dataset. *Computers and Electronics in Agriculture*. <http://doi.org/10.1016/j.compag.2017.12.001>
- Boizet, A., Garcia, A., Tireau, A., Charleroy, A., Cabrera-Bosquet, L., Vidal, M., ... Migot, V. (n.d.). PHIS user documentation. Retrieved December 1, 2019, from <https://opensilex.github.io/phis-docs-community/>
- Malounas, I. (2019). *Evaluation of multispectral and hyperspectral cameras for detection of powdery mildew on tomato plants using convolutional neural networks*. Wageningen University and Research.
- Minervini, M., Fischbach, A., Scharr, H., & Tsiftaris, S. A. (2016). Finely-grained annotated datasets for image-based plant phenotyping. *Pattern Recognition Letters*. <http://doi.org/10.1016/j.patrec.2015.10.013>
- Neveu, P., Tireau, A., Hilgert, N., Nègre, V., Mineau-Cesari, J., Bricchet, N., ... Cabrera-Bosquet, L. (2019). Dealing with multi-source and multi-scale information in plant phenomics: the ontology-driven Phenotyping Hybrid Information System. *New Phytologist*, 221(1), 588-601. <http://doi.org/10.1111/nph.15385>
- Publio, G. C., Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., ... Zafar, H. (2018). ML-Schema: Exposing the Semantics of Machine Learning with Schemas and Ontologies, (1), 1-5. Retrieved from <http://arxiv.org/abs/1807.05351>
- Vidal, M., Heinrich, G., Migot, V., & Tireau, A. (n.d.). Clients for phis2 web services. Retrieved August 1, 2019, from <https://github.com/OpenSILEX/phis-ws-clients>
- Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., ... Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 1-9. <http://doi.org/10.1038/sdata.2016.18>

## Document information

<b>EU Project N°</b>	739514	<b>Acronym</b>	EMPHASIS-PREP
<b>Full title</b>	Preparation for EMPHASIS: European Infrastructure for multi-scale Plant Phenomics and Simulation for food security in a changing climate		
<b>Project website</b>	emphasis.plant-phenotyping.eu		

<b>Deliverable</b>	<b>N°</b>	D4.4	<b>Title</b>	Analysis of imaging approaches - Mapping and gapping analysis of imaging approaches in different PPI
<b>Work Package</b>	<b>N°</b>	4	<b>Title</b>	e-INFRASTRUCTURES

<b>Date of delivery</b>	<b>Contractual</b>	31/January/2020	<b>Actual</b>	1/February/2020 (Feb 20)
<b>Dissemination level</b>	X	PU Public, fully open, e.g. web		
		CO Confidential, restricted under conditions set out in Model Grant Agreement		
		CI Classified, information as referred to in Commission Decision 2001/844/EC.		

### Authors (Partner)

<b>Responsible author</b>	<b>Name</b>	Hendrik de Villiers	<b>Email</b>	<a href="mailto:Hendrik.devilliers@wur.nl">Hendrik.devilliers@wur.nl</a>
---------------------------	-------------	---------------------	--------------	--

### Version log

Issue Date	Revision N°	Author	Change
0.9	1	Hendrik de Villiers	

This project has received funding from the European Union's Horizon 2020 Coordination and support action programme under grant agreement No 739514. This publication reflects only the view of the author, and the European Commission cannot be held responsible for any use which may be made of the information contained therein.



## Contents

Document information .....	2
----------------------------	---

### Documents used in the preparation of this deliverable:

- Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist. Semantic Web for the Working Ontologist*. Elsevier Inc. <http://doi.org/10.1016/C2010-0-68657-3>
- Banakar, A., Polder, G., Ruizendaal, J., & Balendonk, J. (2019). *SpectralCam evaluation dataset and experimental report*. Wageningen University and Research.
- Barth, R., IJsselmuiden, J., Hemming, J., & Henten, E. J. V. (2018). Data synthesis methods for semantic segmentation in agriculture: A Capsicum annum dataset. *Computers and Electronics in Agriculture*. <http://doi.org/10.1016/j.compag.2017.12.001>
- Boizet, A., Garcia, A., Tireau, A., Charleroy, A., Cabrera-Bosquet, L., Vidal, M., ... Migot, V. (n.d.). PHIS user documentation. Retrieved December 1, 2019, from <https://opensilex.github.io/phis-docs-community/>
- Malounas, I. (2019). *Evaluation of multispectral and hyperspectral cameras for detection of powdery mildew on tomato plants using convolutional neural networks*. Wageningen University and Research.
- Minervini, M., Fischbach, A., Scharr, H., & Tsiftaris, S. A. (2016). Finely-grained annotated datasets for image-based plant phenotyping. *Pattern Recognition Letters*. <http://doi.org/10.1016/j.patrec.2015.10.013>
- Neveu, P., Tireau, A., Hilgert, N., Nègre, V., Mineau-Cesari, J., Brichet, N., ... Cabrera-Bosquet, L. (2019). Dealing with multi-source and multi-scale information in plant phenomics: the ontology-driven Phenotyping Hybrid Information System. *New Phytologist*, 221(1), 588-601. <http://doi.org/10.1111/nph.15385>
- Publio, G. C., Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., ... Zafar, H. (2018). ML-Schema: Exposing the Semantics of Machine Learning with Schemas and Ontologies, (1), 1-5. Retrieved from <http://arxiv.org/abs/1807.05351>
- Vidal, M., Heinrich, G., Migot, V., & Tireau, A. (n.d.). Clients for phis2 web services. Retrieved August 1, 2019, from <https://github.com/OpenSILEX/phis-ws-clients>
- Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., ... Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 1-9. <http://doi.org/10.1038/sdata.2016.18>

## 1. Executive Summary

### Introduction

Plant phenotyping experiments are generating an ever increasing amount of data, and a choice must be made with respect to how this data is managed. This is an important consideration, because the efficiency, correctness and transparency of data scientist's work is directly impacted by such choices. Furthermore, in cases where it is desirable to share datasets (for example, for collaboration or openness requirements wrt. funding), the data management approach must account for the ease by which third parties can make use of the data.

### Summary

#### *Chapter 2: Mapping/Gapping Analysis*

- Interviews were conducted by Rick van de Zedde and Tony Pridmore by telephone, skype or similar web conferencing tools over a period of some 10 months.
- For the analysis of these interviews we have focussed on the hardware and software tools and methods in use within the community and discuss platform staffs' capabilities in regard to them.
- Mapping: RGB cameras and independent 2D analysis of the resulting images dominates current practice. Gapping: More might be made of the combination of multiple RGB images, and of images obtained from different types of imaging device.
- Mapping: Appropriate calibration tools and processes are in place in a large majority of platforms. Gapping: There is an opportunity for greater standardisation of calibration methods and frequency. All the methods described are internal to the given platform; no mention was made of calibration against other platforms producing similar data.
- Mapping: Phenotyping platform staff have a high degree of understanding of and control over the image analysis pipelines they oversee. The pipelines described rely exclusively on classic image analysis operations. Gapping: The degree of control and widespread use of home-grown and open source software means that there is little/no standardisation in the software tools used, raising the possibility of the EMPHASIS community developing shared approaches. The most significant technological gap the lack of exploitation of recent advances in deep learning, which requires access to sizeable annotated data sets. We return to this below.

- Mapping: Current techniques are predominantly local, and rely on standard file structure and database methods. There is however, evidence of a move towards more systematic representation, storage and management of image and related data.
- Mapping: Phenotyping platform staff span a range of disciplines and are a significant resource. Gapping: There no evidence of commonality in the sources of information they use, which may limit information flow across the community.
- Mapping: Current expectations focus more on data than processes. Gapping: There was little discussion of representation and sharing of pipelines, which should perhaps be addressed.

#### *Chapter 3: Dataset Structuring Patterns in Computer Vision*

- The field of *general* computer vision has produced a number of extremely popular datasets. These include datasets for tasks such as finding particular kinds of objects in images. Each such dataset has its own way of organizing information internally. The MS COCO (Microsoft Common Objects in Context) dataset is a prominent example of this.
- Because of the popularity of some of these datasets, other datasets are sometimes modelled after them. Examples include plant phenotyping datasets, as discussed in the text.
- Dataset formats in general computer vision are constantly changing, but the phenotyping subdomain needs to keep abreast with these changes, as this allows the reuse of code written for general problems (such as recognizing cats, cars or tables) on more specific phenotyping tasks (such as pepper recognition).

#### *Chapter 4: Dataset Management*

- Datasets should be Findable, Accessible, Interoperable and Reusable (the FAIR criteria). That is, researchers should be able to locate and download a dataset. The means by which the dataset is structured and made available should be standardized and open. The dataset should contain enough information about itself (metadata) such that a new user can decide whether or not the dataset is suitable to their needs.

#### *Chapter 5: Representing Experiments and their Measurements in PHIS*

- PHIS (Phenotyping Hybrid Information System) is a data management system specially designed for use in the field of plant phenotyping, and is already in use in large phenotyping facilities. The system aims to produce one definitive representation of all information recorded about and during the course of experiments.
- PHIS employs open semantic web standards in its representation of information, and aims to represent and make available information in a FAIR compliant way.

- PHIS does more than simply collect data. It allows the representation of organizations, projects, experiments and procedures followed during experiments. It enables the association of data with objects which were involved in producing the data such as individual plants measured, sensors such as cameras, and fields in which plants grew.

#### *Chapter 6: Machine Learning with PHIS Datasets*

- For state of the art deep learning methods, it is important to be able to easily extract “tensors” (arrays of numbers) from a dataset so that they can be processed rapidly in high-performance computing environments, using Graphical Processing Units (GPUs) in particular.
- Enabling easy extraction of tensors from PHIS while preserving information about where the information came from is essential for both the speed at which data scientists can begin experimenting with a dataset, but also that the work can subsequently be reproduced and critically evaluated.
- PHIS does not yet internally support description and extraction of tensors, although it does represent all the information necessary on which to build such facilities.
- A possible approach to this challenge (“trace-based” descriptors) was discussed.
- After a deep learning experiment is complete, it is important to be able to share the resulting deep neural network. We discussed the ONNX format, which allows the sharing of neural networks between a number of deep learning frameworks. ONNX enjoys the support of a number of important players in the deep learning field including NVIDIA, Amazon, Facebook, Microsoft, IBM and Intel.

### Recommendations

- We have found that most installations have their imaging pipeline well organised with commercial or openly available software tools, more emphasis should be on the data organisation resulting from these imaging data pipelines in combination with the minimally required metadata. We have shown in the report the potential of PHIS in this matter and we recommend users, facilities to analyse their current way of working and tools for data organisation such as PHIS.
- PHIS is tailored to the specific needs of the plant phenotyping community, and can be extended for future needs arising in this subdomain in particular. Having already been demonstrated in large data gathering facilities, it is suggested that PHIS be adopted for further data management tasks.

- Especially the data management of phenotypic experiments demands a common toolset when researchers want to launch multi-site/ multi-region experiment carried out at different institutes/ fields and exchange collected data and metadata through local instances of PHIS, structured similarly.
- PHIS acts as a single repository for all data pertaining to an experiment, but the workflow for selecting from this information contained in a convenient format for deep learning experiments could be improved. It is suggested that different approaches (such as “trace-based” descriptors) be considered to address this challenge.
- Because of its existing level of adoption and support by major corporations, it is suggested that deep neural networks be disseminated using the ONNX format.

## Annex 1: Check list

Deliverable Check list (to be checked by the “Deliverable leader”)

	Check list	Comments
<b>Before</b>	I have checked the due date and have planned completion in due time	<i>Please inform Management Team of any foreseen delays</i>
	The title corresponds to the title in the DOW	<i>If not please inform the Management Team with justification</i>
	The dissemination level corresponds to that indicated in the DOW	
	The contributors (authors) correspond to those indicated in the DOW	
	The Table of Contents has been validated with the Activity Leader	<i>Please validate the Table of Content with your Activity Leader before drafting the deliverable</i>
	I am using the EMPHASIS deliverable template (title page, styles etc.)	<i>Available in “New EMPHASIS Logo, Templates, CI” on the collaborative workspace</i>
<b><i>The draft is ready</i></b>		
<b>After</b>	I have written a good summary at the beginning of the Deliverable	<i>A 1-2 pages max. summary is mandatory (not formal but really informative on the content of the Deliverable)</i>
	The deliverable has been reviewed by all contributors (authors)	<i>Make sure all contributors have reviewed and approved the final version of the deliverable. You should leave sufficient time for this validation.</i>
	I have done a spell check and verified the English	
	I have sent the final version to the WP Leader and to the Project coordinator (cc to the project manager) for approval	<i>Send the final draft to your WPLleader and the coordinator with cc to the project manager on the 1<sup>st</sup> day of the due month and leave 2 weeks for feedback. Inform the reviewer of the changes (if any) you have made to address their comments. Once validated by the 2 reviewers and the coordinator, send the final version to the Project Manager who will then submit it to the EC.</i>

## Annex 2: Further Semantic Web Concepts

In the following annexes, we elaborate on the technical details of semantic web concepts, as well as providing a more detailed overview of PHIS. This overview includes examples scripts in the Python language for the submission of a case study dataset containing hyperspectral images of tomato plants. These annexes can be used as material for self-study, or as an accompanying text for a workshop on PHIS.

### 2.1. Controlling representations

The earlier chapter on semantic web contained the most essential aspects of semantic web representations. However, semantic web standards go further than this by allowing us to define a strictly controlled set of possible relationships between nodes. Such a controlled set of relationships is known as an ontology.

An ontology is, effectively, a controlled vocabulary for talking about entities we are interested in. This allows us to place sensible restrictions on what users are allowed to enter into a dataset. In some cases, it also allows us to reason about the dataset itself in a way that answers questions for which no direct answer is encoded in the database. For example, in our example graph, there is no direct relationship describing whether fruits come from the same plant, yet we can infer this indirectly by checking whether the plants connected with them are the same plant.

However, how can we impose this kind of control on relationships in the graph? As an example, consider the following incorrect graph represented as triples:

```
Plant1 hasLeaf Leaf1
Plant1 hasLeaf Leaf2
Plant1 hasLeaf Fruit1
```

A human will quickly spot that the last edge is problematic. However, without further help a machine would not be able to tell there is a problem. To illustrate, consider that the machine sees each term in this example as an arbitrary collection of letters and numbers. From this perspective, the triples could just as well have been:

```
A B C
A B D
A B E
```

Clearly there is no way of knowing that the last triple is somehow incorrect without further specification.

Fundamentally, we would like to be able to talk about different kinds of object such as plants or leaves. We would also like to be able to talk about kinds of relationship, such as hasLeaf or hasFruit. In order to do this, semantic web standards support the definition of “is-a” type relationships. Consider an updated version of the example:

```
Plant1 is-a Plant  
Leaf1 is-a Leaf  
Leaf2 is-a Leaf  
Fruit1 is-a Fruit  
hasLeaf is-a Property  
Plant1 hasLeaf Leaf1  
Plant1 hasLeaf Leaf2  
Plant1 hasLeaf Fruit1
```

Because each entity has now been assigned a type using is-a relationships, the graph representation includes a means by which we can say things about plants, leaves and fruits as a group of entities.<sup>9</sup>

Notice that we also defined hasLeaf using a special is-a relationship, asserting that it is a property. Treating properties as if they are objects in themselves allows us to create rules governing how properties are used. For example, we could add the following rules

```
hasLeaf hasDomain Plant  
hasLeaf hasRange Leaf
```

The domain of a property defines what types of objects can be left of the property in a triple. Similarly, the range of a property defines what types of objects can be right of the property in a triple. These two rules together state that only plants may have the property hasLeaf, and that property may only point to a leaf. In our earlier example, a machine would then be able to pick up that the entry

---

<sup>9</sup> For readers with software development experience, it should be noted that “is-a” relationships in semantic web standards are handled in a fully object-oriented fashion. Entities can be defined as being part of a certain class, and classes may inherit from each other.



### **Plant1 hasLeaf Fruit1**

is illegal, since the right hand side is not a leaf. In the same way, the following entry would be rejected, because hasLeaf may only have a plant as the left-hand side:

### **Fruit1 hasLeaf Leaf1**

It is through similar mechanisms that semantic web standards allow information to be recorded, but also to enforce constraints on the structure of the data being stored. This enables us to use triple stores both as a flexible means of recording data, but also enable different groups to standardize datasets by agreeing on a common set of rules.

In the next section, we will look at some concrete examples of how complex information may be represented using graphs. Following this, we will look in more detail at the semantic web standards which allow us to formalize these graphs.

## 2.2. Examples of graph-based dataset representation

In this section, we give some concrete examples of representing common types of data using directed graphs. It is important to realize that many alternative representations are possible for any particular data structure, and in this chapter we present options that do not necessarily follow PHIS' approach. In a later chapter, we will discuss PHIS' approach to dataset representation.

### 2.2.1. Tabular data

One of the most basic approaches to recording experimental results is to use a spreadsheet. While other kinds of data can be represented in spreadsheet format, the most common use case for this format is to enter data as a table. Often a sample is recorded as one row in the spreadsheet, with each column being a specific attribute of the sample. As an example, consider the following table.

Plant ID	Measurement Date	Height (cm)	Number of leaves
P1	2019-04-12	8.0	3
P2	2019-04-12	9.5	4
P1	2019-04-19	10.5	4
P2	2019-04-19	12.0	5

Tabular formats such as in this example are highly intuitive. While not appropriate for all kinds of data, many kinds of project can use this format successfully for at least some of the attributes being recorded.

One of the most basic container formats for tabular data is the comma separated value (CSV) format. These are simple text files where rows correspond to rows in the table, and values in the columns are separated by commas (although other separators such as tabs may be chosen instead). The above table rendered as a CSV file would look as follows.

```
Plant ID, Measurement Date, Height (cm), Number of leaves
1, 2019-04-12, 8.0, 3
2, 2019-04-12, 9.5, 4
1, 2019-04-19, 10.5, 4
2, 2019-04-19, 12.0, 5
```

The advantage of CSV files is that they are easily read by a human (particularly if there aren't too many columns). At the same time, it is easy to write a program to create or load such a table. This simplicity accounts for the format's popularity.

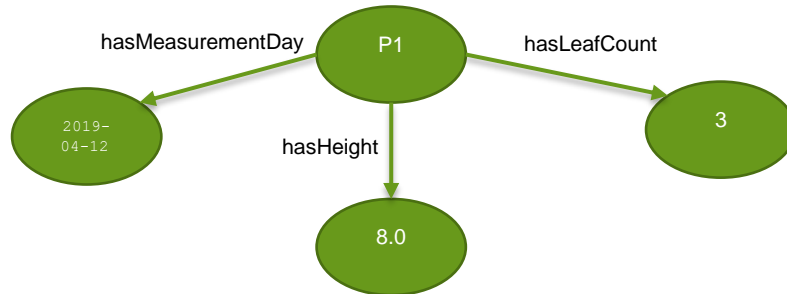
However, a key aspect not recorded by a CSV file is the type of information stored in each column. In the previous example, the CSV file doesn't specify that entries in column 2 must be dates, column 3 may contain real numbers, and column 4 only integers. Automatic inference of types is possible in some instances, but there is still the possibility of error. For example, does the date 03-04-2019 represent the 3<sup>rd</sup> of April, or the 4<sup>th</sup> of March? The answer varies according to local standards or inclination of the dataset creator. Ultimately, manual specification is required to ensure that types are correctly interpreted.

Spreadsheet applications solve this problem by allowing users to enter the type metadata for each cell. This type information is stored along with the cell entries in formats such as XLSX (open XML spreadsheet) and ODS (open document spreadsheet). However, the dataset creator should be careful to annotate the cells with this information.

Despite this capacity for capturing helpful metadata, spreadsheets still suffer from the problem that the self-consistency of the format depends heavily on care taken by the individuals adding to them or altering them. While some restrictions and checks can be encoded in a spreadsheet, relatively innocent additions like a comment on a particular sample can cause automatic processing to fail, or worse, silently corrupt part of the dataset. When multiple people are responsible for data entry, ensuring that everyone involved consistently follows the same entry conventions can be challenging.

We can overcome these difficulties by, instead, using a graph to represent tabular data, and the controlled aspect of ontologies to enforce restrictions on items of data. A graph representation for tables can be quite straightforward. If there is a unique ID of some entity given per row (and each such ID is used in exactly one row), the representation is particularly simple.

Consider the row for plant P1 in the preceding example. We may render the information in that row using the following graph:



The row can then be written as the following triples:

**P1 is-a Plant**  
**P1 hasMeasurementDay 2019-04-12**  
**P1 hasHeight 8.0**  
**P1 hasLeafCount 3**

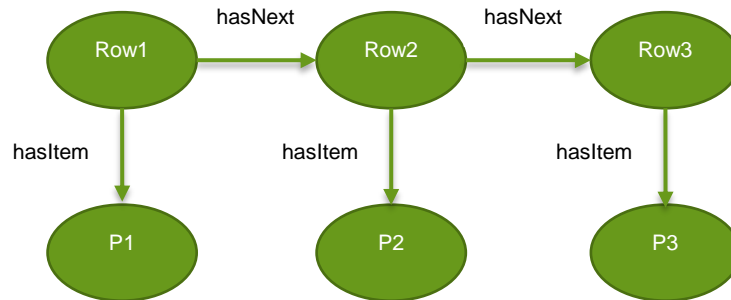
Other rows can be added by simply creating more triples. We can add constraints to limit what can be in the domain and range of each property.

**hasMeasurementDay is-a Property**  
**hasMeasurementDay hasDomain Plant**  
**hasMeasurementDay hasRange Date**  
**hasHeight is-a Property**  
**hasHeight hasDomain Plant**  
**hasHeight hasRange Float**  
**hasLeafCount is-a Property**  
**hasLeafCount hasDomain Plant**  
**hasLeafCount hasRange Integer**

These entries define each property and state that only plants may have those properties in this model. Furthermore, constraints are added such that the measurement day is a date, the leaf count is an integer, and the height is a real number. In a real semantic web representation, the format of each of these kinds of values is standardized, which helps make their meaning unambiguous, and allows certain consistency checks to be performed. For example, the formatting of dates in a particular XML representation is given at the page:

<https://www.w3.org/TR/xmlschema11-2/#date>

While it is likely not necessary in this example dataset, we could also encode the order of rows in the spreadsheet using a structure such as shown in the following figure:



Each row entity above points both to the next row in the series, as well as the plant represented on each row. The corresponding triple store representation is:

```

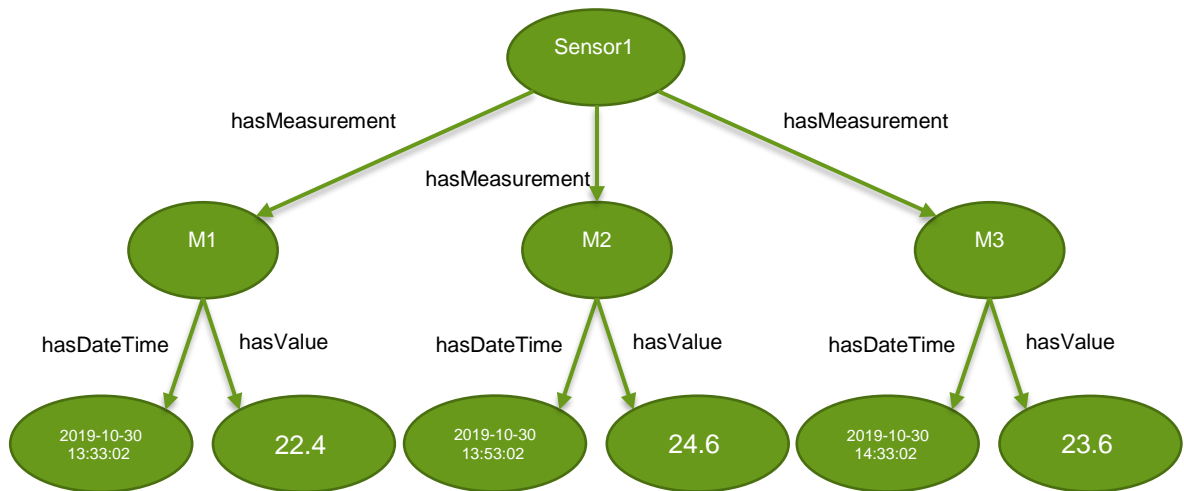
Row1 hasNext Row2
Row2 hasNext Row3
Row1 hasItem P1
Row2 hasItem P2
Row3 hasItem P3

```

Software developers will recognize this structure as a linked list. This structure can be extended for any number of rows.

### 2.2.2. Time series

Suppose that we wish to represent a series of temperature measurements returned periodically by a temperature sensor. We can represent this by creating an entity for the sensor, then connect it to individual measurements. The measurements themselves in turn have a value and a time of measurement. The following figure illustrates such a structure.



This figure can be rendered using triples as follows:

```

Sensor1 hasMeasurement M1
Sensor1 hasMeasurement M2
Sensor1 hasMeasurement M3
M1 hasDateTime 2019-10-30 13:33:02
M1 hasValue 22.4
M2 hasDateTime 2019-10-30 13:53:02
M2 hasValue 24.6
M3 hasDateTime 2019-10-30 14:33:02
M3 hasValue 23.6
  
```

PHIS uses a similar structure to the above for representing certain kinds of point data. However, in addition to the sensor, time and value of a measurement, PHIS also stores provenance information, the object being measured, and information about the value's type like its unit of measurement.

Note that, in the above structure, we could also form the measurements into a linked list from the earliest to the latest. However, since each measurement has a date and time, this ordering is already implicit in the representation.

### 2.3. The semantic web: IRIs and shared ontologies

Up until this point, we have considered the representation of knowledge as graphs without reference to systems which enable the creation and querying of such structures. Fortunately, extensive standards already exist for this purpose underlying what is known as the semantic web.

The semantic web aims to fulfill a similar function for machines what the world wide web provides for humans.

While the world wide web is meant to be read and navigated by humans, the semantic web enables the creation of interconnected knowledge bases in a format that is easily read and navigated by a machine. With each knowledge base added to the semantic web, a new potential foundation for other knowledge bases is available. Some examples of existing ontologies include:

Ontology Name	Function
<a href="#">Friend of a Friend (FOAF)</a>	Representation of people, core details about them, and relationships wrt. other individuals.
<a href="#">Dublin Core</a>	Commonly used vocabulary for recording generic metadata associated with a particular resource. Examples of such metadata include creator, description, date and publisher.
<a href="#">Simple Knowledge Organization System (SKOS)</a>	Vocabulary for describing relationships amongst concepts. For example, that the concept “elephants” is narrower than the concept “mammals”, or that the concept “number” is related to the concept “counting”.

We will now discuss the core concepts underlying semantic web standards. The goal of semantic web standards is to formalize the concept of representing knowledge represented as directed graphs. While extensive facilities exist for advanced use cases, we will focus on those aspects most relevant for the day-to-day work of data scientists.

### 2.3.1. Unique identification of entities and their properties

Firstly, we must be able to identify a given entity uniquely. For example, many datasets might contain a plant labelled PLANT1, but semantic web representations of these datasets should provide labels unique across the semantic web. Without such unique labels, for example, we may associate measurements from the wrong plant to another plant, or even associate contradictory information with these plants.

Semantic web standards deal with this issue by taking inspiration from URLs (uniform resource locators) as used in the world wide web. URLs are meant to refer to a unique web page, including address resolution information which directs a user's browser to contact the correct web server.

As an example, the World Wide Web Consortium's index page on semantic web standards can be found at the following URL:

<https://www.w3.org/standards/semanticweb/>

Note that the URL encodes a hostname [www.w3.org](http://www.w3.org) which identifies the host to be contacted, while the portion "standards/semanticweb" identifies which exact page the user is requesting. Importantly, a URL only identifies a web page uniquely, the contents (and thus "meaning") behind the URL is a separate matter. While it is advisable to form a URL in such a way that it gives a clue to the contents being accessed, this is not a requirement.

Semantic web standards generalize this idea to URIs (uniform resource identifiers). The function of a URI is to uniquely identify some sort of resource. URLs are, in fact, a special case of URIs that specialize in uniquely identifying (for example) websites. Rules for forming URIs are similar to those of URLs. We can break up our preceding example URL in the following way:

```
scheme:[//authority]path
```

The scheme of the url is `https`, indicating in this case the protocol used to contact the web server (hypertext transfer protocol secure). The authority portion (`www.w3.org`) identifies the organization responsible for the contents of the website via DNS. The path portion (`/standards/semanticweb/`) identifies the desired web page uniquely on the web host.

The crucial difference between URLs and other kinds of URIs, is that URLs contain within them all the information necessary to retrieve the unique resource to which it refers (for example a website on a particular host). URIs that are not URLs are only meant to function as a unique name for a particular entity.



IRIs are a further extension of URIs allowing the use of Unicode characters instead of only ASCII characters for URIs.

## 2.4. RDF and Triples

It was realised that knowledge can be expressed in small entities containing a set of three resources. These sets are triples and contain a subject, a predicate, and an object. A triple is the smallest irreducible representation of a binary relationship. The subject is the entity about which the triple says something. The predicate defines what kind of information is given about the subject, and the object represent the content of what is being said by the triple, which can be another resource such as the subject, or be a literal (string, number, date, etc.). A set of triples forms a directed graph in which the subjects and objects are nodes, while the predicates are the edges. The object of one triple can, of course, be the subject of another triple. The entity that is used as a predicate can itself also be used as the subject of other triples, allowing to define information about the property represented by the predicate. The properties of resources in a graph are represented as predicates in triples so that, for instance, the label of a resource is defined by creating a triple (entity “has label” label).

W3C has defined a format for triples known as the Resource Definition Framework (RDF), which can easily be extended. W3C, for instance, also defined the RDF schema ontology, extending RDF by adding annotation properties such as `rdfs:label`, and `rdfs:comment`, to define labels and descriptions of resources respectively. RDF schema also defines properties for specifying subclass relations, allowing the creation of class hierarchies such as taxonomies. Another W3C specification is the Ontology Web Language (OWL), which defines resources and properties needed to create ontologies. RDF, RDF schema, and OWL are themselves also ontologies defined in RDF (that includes RDF itself). Other parties can defined their own ontologies. One of our own examples is the Ontology of Units of Measure (OM), which defines quantitative units and quantities. Publishing these ontologies as linked open data allows for the reuse of data when they are expressed using publicly available ontologies.

## 2.5. Queries on triple stores using SPARQL

Once knowledge is represented in a triple store, the need arises to access this knowledge to make inferences. The SPARQL (SPARQL Protocol and RDF Query Language) language defines a rich set of operations which can be used to make such inferences. SPARQL fulfills approximately the same role for RDF triple stores that the SQL language fulfills in relational databases. In fact, SPARQL syntax is similar to that of SQL (by design), and so existing knowledge of SQL can be helpful when learning SPARQL.

The full range of SPARQL functionality falls outside the scope of this document. However, it is instructive to pause and consider how one might use the language to make queries on a phenotyping dataset.

One query of fundamental interest is to return all items of a particular type from the database, for example, images.

```
SELECT ?item
WHERE {
    ?item a :image .
}
```

Here we ask for all entities that have type `:image` (assuming that this type has been defined in the triple store). The keyword `a` is used to indicate the special `is-a` relationship which defines an object's type. Users of SQL will note the similarity to SQL's own `SELECT` statement.

The preceding query will return all images in the database, but we might only want all images from a particular project. We can restrict the query for this purpose as follows

```
SELECT ?item
WHERE {
    ?item a :image .
    ?item :capturedForProject :EPPN2020 .
}
```

Here we assume that each image is connected to the project which gathered it using a `capturedForProject` property. The items returned must satisfy both constraints. They must be both images and captured for the project `EPPN2020`.

We can capture even more elaborate queries by introducing additional variables inside the WHERE clause. In the following example we ask for all cameras that were used to capture images as part of the EPPN2020 project. Here we have changed `?item` to `?x` to emphasize that the select statement is returning `?cam` (cameras), and not `?x` (the images).

```
SELECT ?cam
WHERE {
    ?cam a :camera.
    ?x a :image .
    ?x :capturedForProject :EPPN2020 .
    ?x :capturedUsing ?cam.
}
```

We can see in this example that it is possible to define “wildcard” entities such as `?x` in the preceding example, and constrain them and their relationships with the entities being selected for.

More elaborate queries can be formulated using SPARQL. For example, constraining images to being from a particular plot of land, or captured within a particular date range. Time series may be returned by asking for images of a particular plant over a given date range.

PHIS itself does not directly support general SPARQL queries. Instead, PHIS provides search mechanisms that handle the most common query use cases. This has the advantage of simplifying the task of retrieving information from the PHIS server. However, it is possible (though inadvisable from a software design perspective) to query the triple store indirectly if the RDF4J server is configured to allow this.

## 2.6. Conclusion

In this chapter, an overview was provided of knowledge representation using triple stores. While the general topic of semantic web standards falls beyond the scope of this document, core concepts were discussed with reference to common patterns of data representation in phenotyping / machine learning datasets such as spreadsheets and time series. This provides perspective on the internal representation PHIS utilizes to store and organize experimental data. For a more in-depth text on semantic web standards, the reader is referred to “Semantic Web for the Working Ontologist” (Allemang & Hendler, 2011). In the following chapter, we discuss PHIS itself and the general principles used to submit and retrieve information either manually or in an automated fashion.

## Annex 3: An Introduction to PHIS

In this and the following chapter, we will provide a walkthrough of setting up an experiment and inserting data into PHIS. It is important to note that certain tasks in PHIS can be performed using its web interface. However, the focus of this text is on providing software developers the means of automating such tasks. For an in-depth look at the web interface's functionality, the reader is referred to <https://opensilex.github.io/phs-docs-community/> (Boizet et al., n.d.).

### 3.1. REST services for software developers

Apart from web interface functionality offered for manual editing by humans, PHIS also offers an interface for interacting with the database from programming languages.

Ideally, a worker writing programs would like a simple set of functions to call, packaged in library form. Instead of directly defining such a library, PHIS defines a web service that responds to a number of request types. This adds a level of indirection between the programmer and PHIS. However, most modern programming languages support such web service requests, meaning that the programmer is largely free to use their language of choice.

Web services such as provided by PHIS use the functionality of the HTTP protocol that was originally developed for transferring or posting files and other information related to web pages. The scope of these operations has broadened as web pages have become more sophisticated. Additionally, since the protocol provides operations that are useful for networked software other than web browsers, this has become a standard for providing functionality over a network.

In this section, we discuss conceptually how such web service requests are formulated, and how they function. In the subsequent section, we use this knowledge to demonstrate such requests performed in the Python language. It should be clear from this discussion that the principles can be easily translated to another programming language. Please note that readers confident in the use of web services may choose to skip the following section until the start of the PHIS API documentation.

### 3.1.1. Identifying web services using URLs

The first step in performing an HTTP request is to identify the host which will process the response. Let's suppose the PHIS server is the host `phis.mywork.com`. This server will listen for HTTP requests on a certain port number such as 8080, although the exact port number is configurable.

One component of the HTTP request is the URL itself, as one would encounter in the navigation bar of a browser. The primary function of the URL is to identify the web server as well as the particular service being requested from the server.

As an example, consider the web service PHIS provides for defining and retrieving the parameters of a particular sensor (e.g. a camera and its resolution). Accessing the service would use a URL similar to

**`http://phis.mywork.com:8080/opensilex/rest/sensors/ ...`**

This URL defines the web server name (`phis.mywork.com`), the port on which the web server is listening (8080), and path (`opensilex/rest/sensors`) which identifies the web service we want to access. Similarly, the web service for looking up or creating user accounts might be accessed using

**`http://phis.mywork.com:8080/opensilex/rest/users/ ...`**

As indicated by the ellipsis dots, the URL's path component may contain further portions (we will discuss these shortly).

Note that the address of the web server, the port number, and the path are all configurable and so depend on the choices made by the PHIS system administrator. However, the names of the web services (`sensors`, `user`, etc.) are fixed by the PHIS implementation itself.

### 3.1.2. Kinds of HTTP request

HTTP requests come in variants such as GET, POST, PUT, PATCH and DELETE. The type of request identifies the kind of information retrieval / manipulation that is desired. At the time of writing,

the PHIS web API makes use almost exclusively of GET, POST and PUT requests<sup>10</sup>. Broadly, they have the following functions:

- GET requests locate resources such as experiments or data files that already exist in the database. As a special case, when a URL is typed into the navigation bar of a web browser, the browser begins its interaction with the web server by performing a GET request for that URL.
- POST requests insert new resources into the database.
- PUT requests update/replace existing resources in the database.

As a concrete example, for the “sensors” web service, PHIS provides a POST call for defining a new sensor and its specifications, a GET call for retrieving this information and a PUT call for updating the sensor specifications, should this be necessary.

### 3.1.3. Passing Information to a Web Service

Whether an HTTP request has type GET, POST or PUT suggests in general terms what kind of operation is being requested. The URL specifies the web server and the name of the service being requested. Except in the simplest of cases, we still need to provide the web service with the information it requires to perform a particular kind of operation.

There are two key ways through which we can convey additional information to PHIS services using the URL itself. These are path parameters and query parameters.

Path parameters form part of the URL and are typically used to specify a specific resource the web service should interact with. For example, if the URI of the sensor of interest is already known for a particular GET sensors request, that URI can be specified as part of the path of the request URI.

**`http://phis.mywork.com:8080/opensilex/rest/sensors/{sensor-uri}`**

This causes the service to retrieve that particular sensor’s information (assuming such a sensor exists in the database).

Query parameters also form part of the URL, but have a special format distinct from that of the path portion of the URL. Query parameters are often used to perform search functions, although they can convey other information as well. Query parameters are encoded in the URL starting at a

---

<sup>10</sup> One DELETE request is defined, used for logging out from a given session. Developers are in the process of adding more DELETE services for use by administrators.

question mark (?) character and take the form of parameter name / value pairs separated by ampersands (&). For example, the following Google search request contains two query parameters. The first parameter, q, specifies the search term (ontologies). The last parameter, lr, specifies the language preference for the results, Dutch in this case.

**`https://www.google.com/search?q=ontologie&lr=lang_nl`**

Apart from the URL, there are two more portions of an HTTP that can be used to convey information to a web service. These are the header and body of the request. The header is the starting portion of the information exchanged between the client and server. It contains various pieces of metadata describing the kind of contents the receiver can expect in the subsequent body. In addition, it may be used to perform authentication by carrying, for example, a token identifying a particular user (this is how PHIS manages user sessions).

Following the header, the remainder of the HTTP request constitutes the body of the request. The body can take a variety of forms. If a binary file is being transferred between the client and server, this is where the file contents would be encoded. However, most web services defined by PHIS encode this region in JSON format, which is useful for concisely capturing complex structured data in a format that is easily readable by both humans and computers.

#### **3.1.4. Server responses to HTTP requests**

After a request has been processed, the server may send a response back to the client. This response contains a status code indicating success or failure. It further contains a header with additional metadata, followed by a body containing the payload of the response. In a similar fashion to the request, the response body may contain data such as a file, or structured data in a form such as JSON.

#### **3.1.5. Examples of web requests supported by PHIS**

In this section, we illustrate some illustrative cases of HTTP interactions between a client and PHIS server. We start with what is typically the first interaction between the client and server, authenticating the client using a username and password. In this example, the PHIS server is at address 192.168.56.101 and is accepting requests on port 8080. Let's begin by examining sections line-by-line as sent by the client to the server. The first two lines might look as follows

```
POST /opensilex/rest/brapi/v1/token HTTP/1.1  
Host: 192.168.56.101:8080
```

Here the client indicates that we are performing a POST request to the API service `brapi/v1/token`<sup>11</sup>. The client indicates that the protocol being used is version 1.1 of HTTP. The next line simply notes the host address and port number the client is contacting. The header metadata continues with the following four lines.

```
User-Agent: curl/7.64.0  
Content-Type: application/json  
Accept: application/json  
Content-Length: 143
```

The last four lines indicate the client software name (curl in this case). It indicates that both the request and expected response bodies should be JSON format. Finally, the number of characters in the request body is provided.

At this point, the body of the request begins with the following lines.

```
{  "grant_type": "password",  
    "username": "admin@opensilex.org",  
    "password": "21232f297a57a5a743894a0e4a801fc3",  
    "client_id": "string" }
```

This request body provides a useful introduction to basic JSON notation. Here we are providing server with a set of key / value pairs separated by commas and enclosed by a pair of curly braces. Such as set of key / value pairs are referred to as JSON “object” (programming languages often use terms such as dictionary, map or hash to refer to the same kind of structure).

As can be seen, the preceding JSON object provides both a username and password for the server to authenticate. The password is technically an MD5 hash of the real password (admin in this case).

Given this request, we now examine a possible response the server may provide. The header of the response is given below.

---

<sup>11</sup> The service name refers to the BreedingAPI (BrAPI), a plant phenotyping dataset interchange API which PHIS implements as part of its functionality. Consult <https://brapi.docs.apiary.io/#> for further information.



**HTTP/1.1 200**

**Location: http://192.168.56.101:8080/opensilex/rest/brapi/v1/token**

**Access-Control-Allow-Origin: \***

**Access-Control-Allow-Headers: origin, content-type, accept, authorization**

**Access-Control-Allow-Credentials: true**

**Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS, HEAD**

**Content-Type: application/json**

**Content-Length: 168**

**Date: Thu, 10 Oct 2019 11:03:50 GMT**

Notice in the first line the protocol (HTTP/1.1) and the response status code 200 (meaning the request was successful). Also note later in the response header that the body's content format is specified as being in JSON format. Following the header, the body might look as follows

```
{"metadata" : {"pagination":null, "status":null, "datafiles":[]},
  "userDisplayName":"Admin OpenSILEX",
  "access_token":"43852b9c2df4d096695bbda79264bb8f",
  "expires_in":"10763"}
```

This JSON object contains the access token the client needs under the “access\_token” key. In subsequent calls to PHIS services, the client can use this token to identify the user account associated with the requests. The token is valid until either the expiration time has run out, or a corresponding DELETE call is made to invalidate the token. Furthermore, notice under “metadata” that we can use JSON objects nested inside other JSON objects. Inside the “metadata” entry, we also use the special value `null` to indicate missing/empty values. The entry “datafiles” contains the empty list `[]` (we will return to lists later).

### 3.2. A First Example in Python

Modern programming languages usually have inbuilt facilities for performing web service requests. In this section we illustrate how the token request can be performed using Python. We employ a redacted and commented version of demonstration code available on the OpenSILEX Github repository at

<https://github.com/OpenSILEX/phis-ws-clients/blob/master/python/images/post.py>

Note that, in Python, comments are started by using a hash(#).

```
import requests # The requests library handles HTTP requests.
import json     # Handles the conversion to and from JSON notation.
import hashlib  # Used to hash the user password as PHIS requires.

# First we define the host, port and path to the PHIS web services.
host = 'http://192.168.56.101:8080/opensilex/rest/'

# We specify in our headers that both the request and response are
# in JSON format.
headers = { 'Content-Type': 'application/json',
            'accept' : 'application/json',
          }

# We specify the body of our HTTP request using a Python dictionary.
# Note the use of hashlib to hash the user's password (admin).
data = {
    'grant_type': 'password',
    'username': 'admin@opensilex.org',
    'password': hashlib.md5('admin'.encode('utf-8')).hexdigest(),
    'client_id': 'string'
}

# We convert the data from Python dictionary format to JSON format
# stored in a string.
json_data = json.dumps(data).encode('utf-8')

# We form the URL to the particular web service we would like to use
url = host + 'brapi/v1/token'

# Here we make the POST request to the given url, adding the headers
# and data defined earlier.
response = requests.post(url, headers = headers, data = json_data)
```

```
print(response.text + "\n") # We print the response text

# We convert the response from JSON and then get the access token
token = response.json()['access_token'];

# We print the token.
print ("token : " + token + "\n")
```

In the preceding listing, the performance of the request was a matter of:

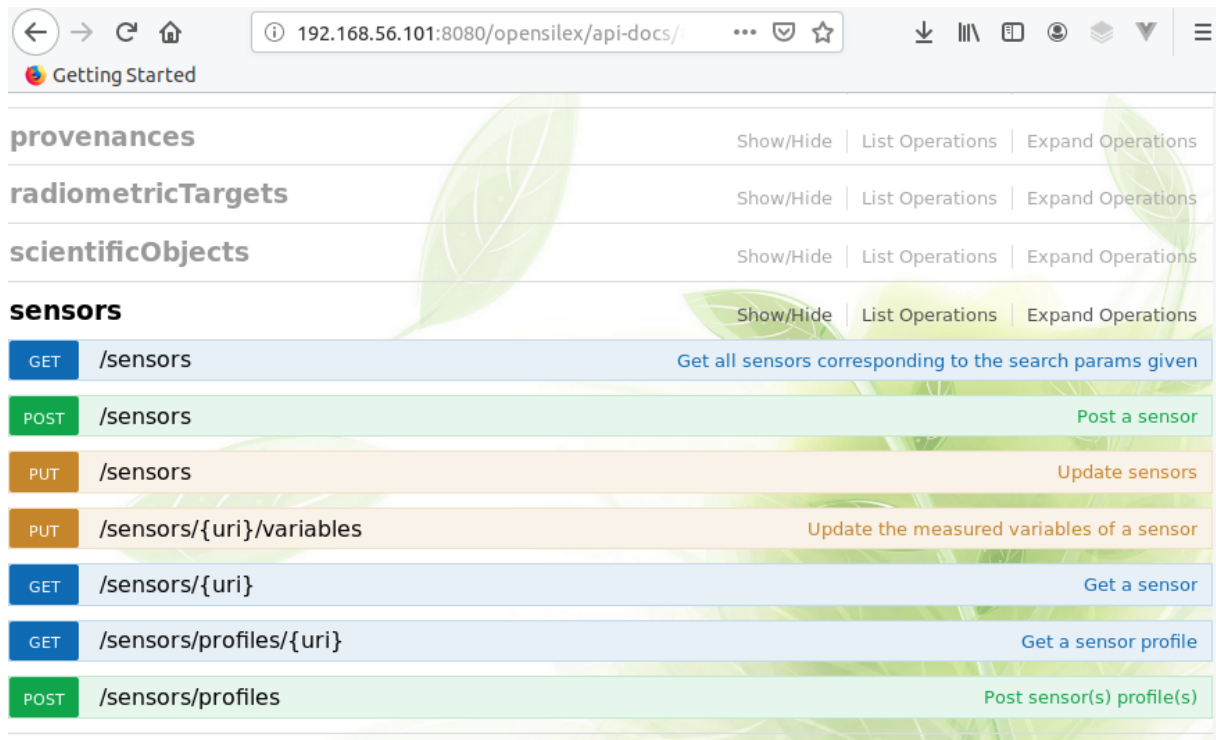
- Importing the libraries which perform HTTP requests, JSON encoding, and password hashing.
- Specifying the hostname, port number and path to the web service.
- Defining the contents of the header and body. The body (in this case) needed converting to JSON notation.
- Performing the POST request using the appropriate library call.
- Converting the response body back from JSON and look up the token value.

In general, calls to other PHIS web services follow more or less the same procedure.

### 3.3. Using Swagger Documentation to Explore the PHIS API

PHIS supports a vast array of web service calls, which are also evolving as the software matures. In order to define and document the API calls, PHIS makes use of a tool named Swagger (<https://swagger.io/>). Amongst other advantages, this enables PHIS to provide developers with a API documentation webpage. This webpage also allows the interactive testing of calls within the browser itself. This is an excellent tool to obtain a sense of what kinds of interactions are supported, as well as the style of interaction expected by the various calls.

In the following example, we show part of the Swagger documentation interface for a PHIS server located at 192.168.56.101 responding on port 8080. A partial list of services can be seen (provenances, radiometricTargets, scientificObjects, sensors). The last entry, sensors, has been expanded to show all calls available for that particular service. Note that the function of each call is briefly described on the right.



provenances		Show/Hide	List Operations	Expand Operations
radiometricTargets		Show/Hide	List Operations	Expand Operations
scientificObjects		Show/Hide	List Operations	Expand Operations
<b>sensors</b>		Show/Hide	List Operations	Expand Operations
GET	/sensors	Get all sensors corresponding to the search params given		
POST	/sensors	Post a sensor		
PUT	/sensors	Update sensors		
PUT	/sensors/{uri}/variables	Update the measured variables of a sensor		
GET	/sensors/{uri}	Get a sensor		
GET	/sensors/profiles/{uri}	Get a sensor profile		
POST	/sensors/profiles	Post sensor(s) profile(s)		

Clicking on one of the calls will expand the entry to reveal call documentation and an interactive example, as illustrated in the following screenshot.

GET /sensors Get all sensors corresponding to the search params given

**Implementation Notes**  
Retrieve all sensors authorized for the user corresponding to the searched params given

**Response Class (Status 200)**  
Retrieve all sensors

Model | Example Value

```
[
  {
    "uri": "http://www.opensilex.org/demo/2018/s18001",
    "rdfType": "http://www.opensilex.org/vocabulary/oeso#Sensor",
    "label": "par03_p",
    "brand": "Skye Instruments",
    "model": "m001",
    "serialNumber": "A1E345F32",
    "inServiceDate": "2017-06-15",
    "dateOfPurchase": "2017-06-15",
```

Response Content Type

The entry begins with a description of the call, an example of an expected result, as well as the content type of the response (with other options if applicable).

Following this, a list appears showing the possible parameters which can be provided to the call. Each parameter name is shown, along with example values, a description of what the parameter is for, the type of parameter (path, query, header or body), as well as the type of the data.

Parameters				
Parameter	Value	Description	Parameter Type	Data Type
pageSize	<input type="text" value="20"/>	Number of elements per page (limited to 150000)	query	integer
page	<input type="text" value="0"/>	Current page number	query	integer
uri	<input type="text" value="http://www.opensilex.org/demo/2018/s1800"/>	Search by uri	query	string
rdfType	<input type="text" value="http://www.opensilex.org/vocabulary/oeso#"/>	Search by type uri	query	string
label	<input type="text" value="par03_p"/>	Search by label	query	string
brand	<input type="text" value="Skye Instruments"/>	Search by brand	query	string
serialNumber	<input type="text" value="A1E345F32"/>	Search by serial number	query	string

The parameter list doubles as a set of entry fields for performing test calls using the next part of the interface. Test calls can be performed by pressing the “Try it out!” button. Responses to the test call and other information about the exchange are then displayed as in the following screenshot.

Try it out! [Hide Response](#)

Curl


```
curl -X GET --header 'Accept: application/json' --header 'Authorization: Bearer 43852b9c2df4d096695bbda79264bb8f' 'h
```

Request URL

```
http://localhost:8080/opensilex/rest/sensors?pageSize=20&page=0
```

Response Body

```
{
  "metadata": {
    "pagination": {
      "pageSize": 20,
      "currentPage": 0,
      "totalCount": 3,
      "totalPages": 1
    },
    "status": [],
    "datafiles": []
  },
  "result": {
    "data": [
      {
        "uri": "http://www.opensilex.org/opensilex/2019/s19003",
        "rdfType": "http://www.opensilex.org/vocabulary/oeso#Camera",
        "label": "Hemispherical Camera",
        "brand": "Foo",
        "model": null,
        "serialNumber": null
      }
    ]
  }
}
```

 Important: Before any other example calls are made, a POST call must first be made to the `/brapi/v1/token` service. This can be done using the same web interface. Once a valid token has been retrieved with this call, it will be used automatically within all other API example calls.

Finally, it is worth mentioning that tools exist to automatically generate wrappers of API calls in a number of languages. For example, Swagger Codegen (<https://github.com/swagger-api/swagger-codegen>) supports languages such as (amongst others) C, C++, C#, Java, Perl, Python, Ruby, R. The reader could consider using such a tool if a wrapper for the PHIS API is not available in their language of choice. However, automatically generated wrappers may not handle the *implicit* expectations of the API correctly (that is, those requirements that do not form part of the formal Swagger specification). Therefore, generated wrappers should not be assumed to be correct without further testing.

### 3.4. Conclusion

In this chapter, we took an introductory look at making calls to PHIS web services, and how to make use of Swagger documentation in order to find more information about specific calls. In the subsequent chapter, we present a case study example where a hyperspectral image dataset is uploaded to the PHIS database.



## Annex 4: Dataset Capturing in PHIS: A Case Study

In this chapter and associated tutorial scripts<sup>12</sup>, we demonstrate the entry of a pre-existing dataset into PHIS. The dataset consists of a mixture of tabular and image data, which together help illustrate some of the most important use cases and provides an overview of key aspects of the API. We also discuss common pitfalls that users should be aware of.

Having discussed the kind of entities stored in PHIS databases in the main matter of this document, we will demonstrate the practical aspects of submitting a dataset. Each section will be devoted to a particular kind of entity or closely related group of entities. In the next section, we begin by discussing the case study dataset to be entered into PHIS.

### 4.1. Case study dataset

The case study dataset (Banakar, Polder, Ruizendaal, & Balendonk, 2019; Malounas, 2019) contains a collection of RGB and hyperspectral images of tomato plants. Certain of the tomato plants are infected by powdery mildew.

For illustration, consider the following two RGB images taken of a healthy plant and an infected plant respectively.

---

<sup>12</sup> Note that accompanying scripts are partly based on examples from the OpenSILEX Github repositories at <https://github.com/OpenSILEX/phis-ws-clients/tree/master/python> (Vidal, Heinrich, Migot, & Tireau, n.d.).



(a)



(b)

Healthy plant (a) Powdery mildew plant (b)

The larger project intended to compare different hyperspectral cameras and their ability to help classify between healthy and infected plants. For simplicity, we limit the case study dataset to 4 images collected with a Specim FX10. Of these images, 2 are of control (healthy) plants, while the other 2 are of diseased plants.

In the subsequent sections, we discuss how to create project metadata, track individual plants and upload the dataset itself.

#### 4.2. Using the example code

To accompany this document, a set of submission scripts written in Python have been created to illustrate how to access the various PHIS entities and upload the captured data. These are intended to be explored in conjunction with reading the following sections of this document. These scripts, along with the demonstration dataset, can be found at:

<https://git.wur.nl/villi003/phs-demo-scripts>

In each section, we will indicate which files in the collection of scripts are relevant to the present portion of the text, and also discuss what each of these scripts contains.

In the following sections, the discussion will focus on the general aspects of accessing the various PHIS web services, and what they are used for. Inside the corresponding scripts, concrete examples are provided and the user will find comments which explain in a step-wise fashion which operations are being performed and why.

### 4.3. Projects

Projects are typically the highest level of organization that a project member may administer on an existing PHIS system. As described previously, project entities capture information such as the project description, contact person, financial support, running time, and so forth.

Projects may be created in one of two ways. One way is to use the web application, the other is to use the web service API. In this chapter, we will mostly focus on using the web service API, partly because it is useful to be able to automate the creation of all entities for a particular project. However, it is instructive to see how to use the web application for this purpose as well, which we will now consider.

To create a project using the web application, select “Experimental Organization” and from the dropdown “Projects”. This will bring up the search page for existing projects. However, just below the “Projects” heading, there is a “Create Project” button. Pushing this button brings up the project creation form. This is illustrated in the following screenshot:

## Create Project

**Acronym \***

**Name \***

---

**Subproject of (optional)**

**Subproject Of**

**Subproject Type**

**Financial Support**

**Financial Name**

**Date Start \***

**Date End \***

As can be seen, in the case of projects, using the web interface for project creation is relatively straightforward, and only needs to be used once during the project lifecycle.

In the following part of the section, we consider adding projects using PHIS web services, and an example showing the creation of the case study project entity is given in the script `create_project.py`.

PHIS defines four web services for interacting with projects:

Web service	Description
<b>GET /projects</b>	Find projects matching certain criteria and return their attributes.
<b>POST /projects</b>	Create new projects with attributes as specified.
<b>PUT /projects</b>	Update the attributes of already existing projects.
<b>GET /projects/{URI}</b>	Return the attributes of the project with URI as specified in path.

We start our discussion with the **GET /projects** web service. Typical of other GET services without a URI path parameter in PHIS, this service is used to search for projects matching particular criteria. It is also possible to use this service to return all projects entries in the database by not specifying any criteria. In addition, the criteria are usually specified using query parameters (recall that query parameters are specified in the URL after a question mark).

For example, to search for a project with acronym SCE, a GET request is made with a URL of the form:

**`http://phisserver:port/opensilex/rest/projects?acronyme=SCE`**

Notice here we provide one query parameter, “acronyme”, with the value SCE. If such a project exists, a possible response from the server may be:

```
{
  "metadata": {
    "pagination": null,
    "status": [],
    "datafiles": []
  },
  "result": {
    "data": [
      {
        "uri": "http://www.opensilex.org/opensilex/SCE",
        "name": "Spectral Camera Evaluation",
        "acronyme": "SCE",
        "subprojectType": "",
        "financialSupport": "WUR",
        "financialName": "None",
        "dateStart": "2017-12-01",
        "dateEnd": "2018-12-31",
        "keywords": "",
        "description": "The Spectral Camera Evaluation project...",
        "objective": "Establishing the performance of ...",
        "parentProject": "",
        "website": null,
        "contacts": [
          {
```

```
        "email": "gerrit.polder@wur.nl",
        "firstName": "Gerrit",
        "familyName": "Polder",
        "type":
"http://www.opensilex.org/vocabulary/oeso/#ScientificContact"
    }
  ]
}
}
```

Let us take a closer look at the response structure. At the highest level, the response consists of a JSON object with two key values, “metadata” and “result”, as shown in the following snippet.

```
{
  "metadata": { ... },
  "result": { ... }
}
```

We are primarily interested in the JSON object associated with the “result” key. This JSON object has a single key, “data”, like the following

```
{
  "data": [ ... ]
}
```

Notice that the value associated with “data” is a list ([ ]) rather than an object ({}). This is because, in general, GET queries such as these may return more than one match. If there is only one match (as in the current case), we obtain the following structure:

```
{
  "data": [ {...} ]
}
```

However, if there were three matches, the structure would have been as follows:

```
{
```

```
"data": [ {...}, {...}, {...} ]  
}
```

So, the query is able to return multiple matches, each as a JSON object listed under the “data” entry. This format is typical of similar GET services in PHIS where the possibility of multiple results exists.

Finally, we turn our attention to the actual project entry with the structure:


```
{  
  "uri": "http://www.opensilex.org/opensilex/SCE",  
  "name": "Spectral Camera Evaluation",  
  "acronyme": "SCE",  
  ...  
}
```

Here we see each attribute of the given project, along with the value recorded in the database. In this case, most of the entries are string values such as “SCE”. However, one entry, “contacts”, actually consists of a list of contact persons (notice the square brackets [ ] surrounding the contact details). In this case, there is only one contact person listed.

For completeness, the complete list of query parameters accepted by GET `/projects` can be found in the following Swagger documentation.

Parameter	Value	Description	Parameter Type	Data Type
pageSize	<input type="text" value="20"/>	Number of elements per page (limited to 150000)	query	integer
page	<input type="text" value="0"/>	Current page number	query	integer
uri	<input type="text" value="http://www.opensilex.org/demo/projectTest"/>	Search by URI	query	string
name	<input type="text" value="projectTest"/>	Search by name	query	string
acronyme	<input type="text" value="PT"/>	Search by acronyme	query	string
subprojectType	<input type="text" value="subproject type"/>	Search by subproject type	query	string
financialSupport	<input type="text" value="financial support"/>	Search by financial support	query	string
financialName	<input type="text" value="financial name"/>	Search by financial name	query	string
dateStart	<input type="text" value="2015-07-07"/>	Search by date start	query	string
dateEnd	<input type="text" value="2016-07-07"/>	Search by date end	query	string
keywords	<input type="text" value="keywords"/>	Search by keywords	query	string
parentProject	<input type="text" value="parent project"/>	Search by parent project	query	string
website	<input type="text" value="http://example.com"/>	Search by website	query	string

Since many of the other GET services act almost identically to **GET /projects**, we will discuss them in less detail. However, the query parameters and returned object keys do vary according to the kind of entity we are searching for. Therefore, in subsequent discussion we will focus on these aspects primarily.

 Important note: Often PHIS services perform partial matches when string values are provided. In such cases, the developer needs to bear in mind that, if they wish to find one particular matching string, naively searching for that string may return all items that at least partially contain that string. For example, a query for a project with name “project\_1” may return “project\_1”, but also “project\_10” (if it exists).

The service **POST /projects** allows the client to create a new project entry in the PHIS database. Unlike the corresponding GET service, information is passed to the POST service using a body parameter. This takes the form of a list of JSON objects. Each JSON object contains the attributes



for a new project to be created. An example of such a body parameter is given in the following snippet:

```
[
  {
    "name": "projectTest",
    "acronym": "PT",
    "subprojectType": "subproject type",
    "financialSupport": "European",
    "financialName": "Grant No 382732",
    "dateStart": "2015-07-07",
    "dateEnd": "2016-07-07",
    "keywords": "keywords",
    "description": "description",
    "objective": "objective",
    "parentProject": "http://www.opensilex.org/opensilex/parentpj",
    "website": "http://example.com",
    "contacts": [
      {
        "email": "admin@opensilex.org",
        "firstName": "Ad",
        "familyName": "Ministrator",
        "type":
"http://www.opensilex.org/vocabulary/oeso/#ScientificContact"
      }
    ]
  }
]
```

If the POST request is successful, the client might get a response like the following:


```
{
  "metadata": {
    "pagination": null,
    "status": [
      {
        "message": "Data inserted",
        "exception": {
          "type": "Info",
          "href": null,

```

```
    "details": "1 projects inserted"
  }
},
"datafiles": [
  "http://www.opensilex.org/opensilex/PT"
]
}
```

In the above response, there are two particularly important items of interest. One is the value of “status”, indicating the insertion was a success. The other is the contents of “datafiles”. If the operation was a success, this list contains the URIs of all the newly inserted projects (in this case, only one project was created). Note that PHIS generates its own URI for the project, although it uses the project acronym as a template in this case.

In general, PHIS generates URIs for new database entries itself. In the case of projects, the URI is informative in that it uses the project acronym. However, for many other entity types, PHIS generates a URI which is not based on the entity metadata. For example, the first variable created might have the URI <http://www.opensilex.org/opensilex/id/variables/v001>. While in these cases this has the advantage of ensuring that URIs are unique, it does have the disadvantage of not conveying any information about the entity without looking at the triple store.


 Important note: When a successful POST call is made, PHIS will always create a new object, even if there is already an identical entry in the database. The new object will simply have a different generated URI. New developers should keep this behaviour in mind, because it can lead to unexpected results. For example, one half of a dataset might be associated with the first project created, and the other with the second instance.

Because of the aforementioned behaviour, it is important to first check whether a particular entity has already been created in the database before proceeding with a POST call. Therefore, one should first perform a GET call to check for existing instances. If the GET call indicates no match, then a POST call may be performed. If there is a match, the developer may opt to perform a PUT call instead, which will update the attributes of the existing object in the database. This behaviour is demonstrated in the project creation script.

A call to PUT /projects is identical to a call to POST /projects, except that the PUT call must specify the URI of an existing project for each entry being updated. Therefore, the request body parameters should look similar to the following

```
[
  {
    "uri" : "http://www.opensilex.org/opensilex/PT",
    "name": "projectTest",
    "acronyme": "PT",
    "subprojectType": "subproject type",
    ...
  }
]
```

Because the URI is provided, PHIS can find the unique object it identifies, and can then update its attributes with the newly provided values (assuming the particular service allows changes to the relevant attributes).

 Important note: Apart from logging out of a session, PHIS services do not support DELETE requests (although developers are currently implementing new services accessible by the system administrator). It is therefore important to carefully avoid polluting the database with invalid entries. While an entry's data may be updated, the entry itself cannot be deleted by a normal user using the web services. So, it is best to perform testing for a new project's workflow on a test PHIS server, rather than a production server.

#### 4.4. Experiments

Experiments represent units of work in one or more projects. While the exact division of projects into such units of work is left to the project manager, PHIS provides an internal structure for capturing different aspects of a given experiment. Amongst others, these include:

- Experiment description and keywords.
- The sensors employed during the experiment.
- The variables measured during the experiment.
- The place where the experiment is performed.
- The crop species under investigation.
- Administrative metadata such as start and end dates, containing projects, contact persons.

PHIS defines a set of web services under `/experiments` that are conceptually similar to those under `/projects`. The major difference lies in which types of descriptive data sent or received to the web services. Furthermore, there are two additional specialist web services for updating the variables and sensors associated with the experiments.

Web service	Description
<b>GET /experiments</b>	Find experiments matching certain criteria and return their attributes.
<b>POST /experiments</b>	Create new experiments with attributes as specified.
<b>PUT /experiments</b>	Update the attributes of already existing experiments.
<b>GET /experiments/{URI}</b>	Return the attributes of the experiment with URI as specified in path.
<b>PUT /experiments/{URI}/variables</b>	Update the list of variables that are measured during the experiment with the given URI.
<b>PUT /experiments/{URI}/sensors</b>	Update the list of sensors which are used by the experiment with the given URI.

In the script `create_experiment.py`, the creation of “experiment” entities is demonstrated. The process is almost identical to that of creating “project” entities, although the reader will note that the experiment script is significantly less complex than project submission script. This is because the logic of checking for existing entities and choosing whether to POST or PUT based on this has been abstracted by defining

```

post_or_put_experiment =
    phis.define_post_or_put(    getter=phis.get_experiments,
                              poster=phis.post_experiments,
                              putter=phis.put_experiments)

```

This block of code defines a function `post_or_put_experiment` that calls the GET, POST and PUT methods as needed to ensure creation of the object without making a duplicate. The reader will notice this method being reused in subsequent scripts, and it is meant to indicate that the calling of these services are similar to invocations in earlier scripts. The reader may always refer back to `create_project.py` for an example of following this entire process explicitly. The abstracted version is implemented in the function `define_post_or_put` contained in the file `phis.py`.

## 4.5. Sensors

Sensor hardware and its specification are important aspects in capturing essential information about an experimental procedure. The following web service calls are supported:

Web service	Description
<b>GET /sensors</b>	Find sensors matching certain criteria and return their attributes.
<b>POST /sensors</b>	Create new sensors with attributes as specified.
<b>PUT /sensors</b>	Update the attributes of already existing sensors.
<b>PUT /sensors/{URI}/variables</b>	Update which variables are measured by the sensor with the specified URI.
<b>GET /sensors/{URI}</b>	Get general device information regarding the sensor with the specified URI.
<b>GET /sensors/profiles/{URI}</b>	Get the measurement specifications of the sensor with the given URI.
<b>POST /sensors/profiles</b>	Update the specifications (sensor profiles) of the sensors specified in the body parameters.

The PHIS web services divide information about sensor hardware into “sensors” and their “sensor profiles”. This distinction corresponds to the representation of “Sensing Device” in the OESO ontology.

In OESO, the parent class of “Sensing Device” is the more general “Device”, which has generic associated properties such as a human readable description, manufacturer and serial number. These properties are set using either **POST** or **PUT /experiments**. Because all sensors inherit directly or indirectly from “Device”, all sensors may have these properties.

By contrast, sensor profiles refer to properties associated with descendant class of “Sensing Device” in OESO to which the particular sensor belongs. These encode specific information which characterizing the measurement process. Examples include aspects such as camera resolution and frame rate. Because these are properties of descendants of “Sensing Device”, not all sensors may be assigned these properties.

In the script `create_sensors.py`, an example is provided for defining a sensor object corresponding to the FX10 hyperspectral camera. This is followed by the attachment of a sensor profile to this object, which specifies the resolution of the sensor.

## 4.6. Provenance

An important aspect of dataset curation is to record the source of information, or provenance, with as much detail as possible. This increases transparency and supports critical engagement with the information contained in the dataset.

However, sources of information are diverse in their nature. They may include, amongst many others, published documents, information from sensors and visual observations. This is compounded by the fact that the result of a workflow processing other information may, in turn, be seen as a data source.

Because of this potential diversity, PHIS treats data provenance generically. Developers can create provenance entities which consist of four components:

Property	Description
<b>uri</b>	Unique URI identifying the provenance entity, generated by PHIS automatically.
<b>label</b>	A short identifier to act as an alias for the object.
<b>comment</b>	A human readable comment describing the source of data this provenance entity describes.
<b>metadata</b>	JSON object storing an arbitrary list of key / value pairs.

Notable here is the relative lack of prescribed structure. However, in the context of specifying information sources, this makes sense. These entries are primarily intended for human consumption, although the “metadata” field allows some structuring using JSON notation.

What is most important is that the provenance entry conveys to a database user clearly how the information was obtained. This may involve referring to URIs of related resources, which can be easily stored in the metadata object. PHIS allows searches based on the metadata object’s contents, which increases its utility in automated workflows.

It is helpful to consider a few use cases for provenance entities:

- Defining a particular procedure employed during an experiment: Experiments are not necessarily monolithic, they may comprise of a number of procedures which together gather the experimental data. A provenance entity may be used to note down which researchers

were performing the measurements, what equipment they were using, how the researchers were using the equipment to perform measurements and any manual procedures that were necessary.

- Indicating an external source of data: Sometimes it is useful to make an externally obtained dataset available alongside locally gathered data. In this case, provenance objects may refer to published papers detailing the dataset, contact persons involved in data gathering and links to where the dataset was obtained from (a website, for example).
- Record the data processing workflow by which derived data was obtained: In certain cases, it may be useful to upload the results of a data processing pipeline alongside the original raw data. This is especially true if the workflow is either complex, is computationally expensive or involved expert judgements. To aid reproducibility, a provenance entity can record the method employed to process the data, references to documents which help to detail these methods, links to source code used for automatic processing of the raw data, and the details of developers or expert judges.

To define and manage data provenance, PHIS provides the following web services:

Web service	Description
<b>GET /provenances</b>	Find provenances matching certain criteria and return their attributes.
<b>POST /provenances</b>	Create new provenances with attributes as specified.
<b>PUT /provenances</b>	Update the attributes of already existing provenances.

The operation of these web services is a relatively straightforward adaptation of the conventions established by previously discussed web services. An example showing the creation of a provenance entity is provided in the script `create_provenance.py`.



## 4.7. Scientific Objects

Tracking the actual test subjects is an important part of data curation. PHIS allows developers to represent test subjects by creating “scientific objects”. This class is defined in the OESO ontology, and can represent objects such as plants, roots, leaves and seeds. It is also possible to represent relationships between scientific objects. For example, one can create three leaves and indicate that they are part of a particular plant.

After having represented scientific objects relevant to an experiment, it is possible to indicate during data uploads which scientific objects were measured in obtaining the data. This allows developers to later query the database to find all measurements related to a particular scientific object. For example, such a query might return a series of photos taken of a particular plant at regular intervals.

PHIS defines the following web services for managing scientific objects:

Web service	Description
<b>GET /scientificObjects</b>	Find scientific objects matching certain criteria and return their attributes.
<b>POST /scientificObjects</b>	Create new scientific objects with attributes as specified.
<b>PUT /scientificObjects/{URI}/{experiment}</b>	Update a scientific object and specify which experiment it belongs to.

These web services behave similarly to previous cases. The only exception is the **PUT** service, which can only update one scientific object at a time. This service also requires the scientific object and experiment IRIs to be passed as path parameters, rather than being amongst the body parameters of the request.

A scientific object maintains the attributes:

Property	Description
<b>uri</b>	Unique URI identifying the scientific object, generated by PHIS automatically.

<b>label</b>	A short identifier to act as an alias for the object.
<b>rdfType</b>	The IRI of the class of scientific object, which identifies the object as, for example, a plant or a seed.
<b>geometry</b>	Indicates the location and extents of the scientific object.
<b>experiment</b>	JSON object storing an arbitrary list of key / value pairs.
<b>isPartOf</b>	Indicates the IRI of another scientific object that this object forms a part of.
<b>properties</b>	A list of properties which further describe the object.

Properties take the form of triples, each consisting of an `rdfType`, a relation and a value. The `rdfType` and relation are both IRIs, representing the type of the value, and the type of relationship the value has to the scientific object.

Taking an example from the documentation, a contact person may be specified by giving the IRI <http://xmlns.com/foaf/0.1/Agent> as the `rdfType`, while the relation may be specified as <http://www.opensilex.org/vocabulary/2018#hasContact>. In this case, the value may be an IRI pointing to the contact person, such as [http://www.opensilex.org/demo/id/agent/marie\\_dupond](http://www.opensilex.org/demo/id/agent/marie_dupond).

In the case study, the scientific objects are individual tomato plants. Each plant has a label of the form “Cn” or “Dn”, where n is a number from 1 to 2, and C/D indicate either control (C) or diseased (D). The script `create_scientific_objects.py` demonstrates the creation of these scientific object entities. Note that, because of the difference in how PUT operations are performed, `define_post_or_put` cannot be used to simplify the script. The scientific object submission script instead explicitly performs the sequence of GET, POST and PUT operations to ensure duplicates are not created.

Once all scientific objects have been created, there is enough structure in the PHIS database to begin uploading data related to these objects, which we discuss in the following sections.

## 4.8. Uploading Data

As mentioned previously, PHIS provides two separate mechanisms for uploading measurement data. These are handled by the services `/data` and `/data/file(s)` services respectively. We discuss these services in turn in the following subsections.

### 4.8.1. Point data

Data managed through the `/data` services can be thought of as collections of individual data points. This subsystem is particularly well suited to storing discrete individual measurements, primarily single numbers or string labels. The subsystem is also well suited to storing spreadsheet data, as each cell in a spreadsheet can be seen as a single data point and uploaded as such. The web app supports uploading of spreadsheet data in `csv` format, although it is currently relatively strict regarding how this spreadsheet is formatted (downloading a template `csv` file to assist data entry is possible via the same interface).

Each data point is characterized by the following attributes:

Property	Description
<code>uri</code>	Unique URI identifying the data point, generated by PHIS automatically.
<code>provenanceUri</code>	URI indicating the provenance of the data point.
<code>objectUri</code>	URI indicating the scientific object being measured.
<code>variableUri</code>	URI indicating the variable being measured (such as weight or temperature).
<code>date</code>	The date and time of measurement. Examples: 2017-06-15 or 2017-06-15T10:51:00+0200
<code>value</code>	The actual measurement

From these properties, we see that each data point includes its source, the object measured, the variable being measured, the time of measurement, and the actual measurement value. Provenance and scientific objects have been discussed previously, but variables have not been discussed so far. We leave a more detailed discussion to a later section, but essentially variables definitions capture three aspects: the trait being measured, the method of measurement, and the units of

measurement. For example, a variable might represent the trait “ambient temperature”, with method “thermocouple” and with units “kelvin”.

The web services of the data point subsystem include:

Web service	Description
<b>GET /data</b>	Find data points by date range, or the URIs of the variable, object or provenance.
<b>POST /data</b>	Create a new point data entry.
<b>GET /data/search</b>	Similar to GET /data, but additionally allows queries based on the object and provenance labels (not only their URIs).

The operation of these web services is similar to the operation of earlier web services. It is worth noting, however, that PHIS does not currently support a PUT service for point data. This means that, once created, a data point’s value cannot be changed. While from one perspective, this is inconvenient, from another this ensures reproducibility and transparency by preventing changes that would invalidate any prior analyses based on the older data.

There are at least two strategies by which a dataset curator can incorporate changes to already submitted data. One strategy makes use of provenance entities as a means of creating new versions of a given dataset without disturbing the previous versions. Another strategy is to use annotation entities to attach comments and replacement values to data points.

Since each data point has a unique URI, it is possible to annotate a data point with a replacement value using the **/annotations** service. The annotation can also contain an explanation for the substitution, possibly referring to events such as sensor failure (created using the **/events** service). In case the replacement value needs to be replaced, the first annotation can itself be annotated. This does mean that users should be aware of the possibility that corrections may have been made in this fashion. It is advisable to add comments about such annotations to either the experiment, project or provenance metadata, as these can be updated.

The main advantage of using the **/data** services is that PHIS provides a more structured way of capturing values and the metadata describing what they mean. Because the structure of the data is now managed by PHIS, it becomes independent of the original data storage format. The service is

also convenient for time series measurements of qualities such as temperature, which can be autonomously submitted to the web services at each measurement point.

The main disadvantage of these point data services is that they are awkward for storing bulky data. Furthermore, only one value may be stored for a given combination of provenance, object, variable and date. In some cases, this can be remedied by encoding multiple values as a JSON object stored in a string. For example, a list of the numbers 1 through 4 can be encoded as "[1, 2, 3, 4]".

However, while it is technically possible to store bulky data encoded in the value as a string, this isn't the intended use of the subsystem. For inherently large units of data, PHIS provides a file storage service which is discussed in the following section.

#### **4.8.2. Bulk data**

Some information is too bulky for submission as point data, typical examples being images or videos. For these cases, PHIS provides the facility to upload individual files to the database using the `/data/file(s)` services.

The advantage of using this subsystem is that data can be uploaded in well-known file formats, which can make subsequent use easier. Metadata can be attached to each file, describing the origin and contents. Files may be associated with scientific objects, such as individual plants being measured. The sensor employed may also be recorded in the file metadata.

The disadvantage of this subsystem is that file formats do not necessarily stay supported for lengthy periods of time. While metadata may be attached to files, PHIS web services treat the contents in a black box fashion. Dataset curators should therefore think carefully about the long-term implications of the file format choices, and attempt to capture as much file metadata as possible. As an important special case, array data may be stored in specially designed interchange formats such as HDF5 (<https://www.hdfgroup.org/>).

PHIS provides the following web services to deal with uploading files:

Web service	Description
GET /data/file/{URI}	Retrieves a file with the given URI.
GET /data/file/{URI}/description	Retrieves the description of the file with the given URI.
GET /data/file/search	Finds all files matching the criteria specified in the body parameters.
POST /data/file	Uploads a single file and its description.
POST /data/files	Uploads multiple files and their descriptions.

Again it is worth noting that PHIS does not support PUT operations for the file storage subsystem. This means that, once a file is uploaded, it cannot be changed or removed from the system. This design choice has an identical rationale as seen in the point data subsystem, and changes to files may be dealt with in exactly the same way (through provenance or annotation entities).

#### 4.8.3. Cross-referencing between data subsystems

Finally, it should be noted that it is possible to combine the point data and bulk data subsystems. One possibility is to post a bulk data file and then post a data point with the file's URI as its value. Typically developers may opt to simply use either one of the subsystems separately, but it may be useful for users to have records be available in both subsystems. For example, this approach allows a web app user to search for data points associated with an object and also get references to images, downloadable in spreadsheet format.

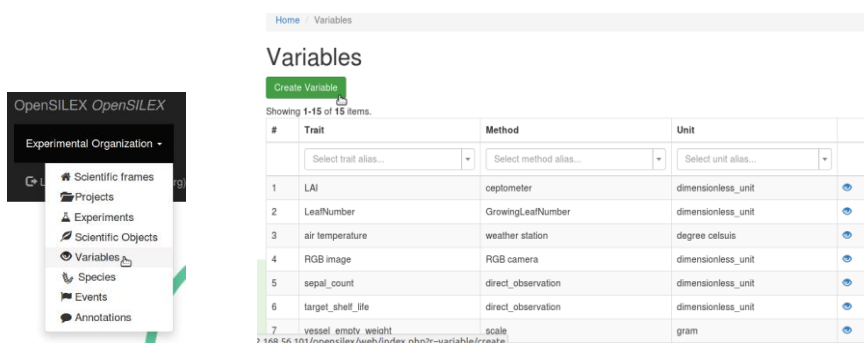
#### 4.8.4. Uploading the case study data: Ground truth labels

In the case study data, each plant is labelled as either control or diseased. The point data subsystem can be used for this purpose in order to store these labels. In this section, we will discuss this process. We will create a variable called "Diseased" in PHIS, which will store a class label for each tomato plant. This will either be the value "diseased" or "healthy", depending on whether or not the plant is infected. This process can be easily extended to storage of other variables, especially for the storage of tabular data associated with experimental samples.

As discussed earlier, each value uploaded to the point data subsystem needs a provenance URI, a concerned object URI, a variable URI, a date/time of measurement and a value to record. We have already discussed the creation of provenance and scientific object entities, but we still have to describe the creation of variables.

Variables are defined by the trait being measured, the method by which the measurement is performed, and the unit of measurement.<sup>13</sup> It is possible to define new traits, methods, units and variables using either the web app or the PHIS web services. The web app method is simple and convenient, although the web services allows for the process to be automated.

To use the web app for creating variables and related entities, select “Variables” under “Experimental Organization”. This brings up a list of existing variables, as shown in the following figures.



Clicking on “create variable” opens the interface for defining traits, methods, units and variables, as shown in the following screenshot.

<sup>13</sup> At the time of writing, developers report that recent changes to PHIS has changed the trait-method-unit model of variables to a quality-entity-method-unit model to conform with ontologies of measurement (OM or QUDT).

## Create Variable

**Variable Label \***

NA\_NA\_NA

---

**Trait \***

Trait label  +

**Method \***

Method label  +

**Unit \***

Unit label  +

Warning

You cannot modify already existing traits, methods and units.

**Ontologies References**

In order to fill ontological references (URI) you can go to these ontologies :

- [AGROPORAL](#) ?
- [AGROVOC](#) ?
- [PLANT ONTOLOGY](#) ?
- [PLANTEOME](#) ?
- [CROP ONTOLOGY](#) ?
- [UNIT ONTOLOGY](#) ?

**Related References**

From here, it is possible to combine existing traits, methods and units into a new variable. If a new trait, method or unit is needed, these can be defined using the plus buttons, bringing up the following interface:

**Trait \***

Trait label  -

**Internal Label**

diseased

**Comment**

Concerned object shows signs of disease.

**Method \***

Method label  -

**Internal Label**

direct\_observation

**Comment**

Values are derived from human observation of the concerned object.

**Unit \***

Unit label  -

**Internal Label**

dimensionless\_unit

**Comment**

Values are dimensionless.

Warning

You cannot modify already existing traits, methods and units.

Note carefully that any entities created cannot be modified via the web interface. Therefore the user should make sure not only that all the details are correct, but also that they are not creating what is essentially a duplicate entity.



Alternatively, variables and related entities may be created using the PHIS web services:

Web service	Description
<b>Trait services:</b>	
<b>GET /traits</b>	Retrieves all traits matching search parameters.
<b>POST /traits</b>	Create new traits as specified.
<b>PUT /traits</b>	Update already existing traits.
<b>GET /traits/{URI}</b>	Get information about trait with specified URI.
<b>Method services:</b>	
<b>GET /methods</b>	Retrieves all methods matching search parameters.
<b>POST /methods</b>	Create new methods as specified.
<b>PUT /methods</b>	Update already existing methods.
<b>GET /methods/{URI}</b>	Get information about method with specified URI.
<b>Unit services:</b>	
<b>GET /units</b>	Retrieves all units matching search parameters.
<b>POST /units</b>	Create new units as specified.
<b>PUT /units</b>	Update already existing units.
<b>GET /units/{URI}</b>	Get information about unit with specified URI.
<b>Variable services:</b>	
<b>GET /variables</b>	Retrieves all variables matching search parameters. No trait, method and unit information is returned.
<b>POST /variables</b>	Create new variables as specified.
<b>PUT /variables</b>	Update already existing variables.
<b>GET /variables/details</b>	Retrieves all variables matching search parameters. Trait, method and unit information is returned.
<b>GET /variables/{URI}</b>	Get information about variable with specified URI.

The script `upload_tabular.py` demonstrates how to utilize these four services to create a trait, method, unit and encompassing variable for a class label. The script also utilises the `POST /data` service to upload either a “diseased” or “healthy” label for each of the plants.

The creation of trait, method and unit entities is similar to creating other PHIS entities, requiring only body parameter containing a label and a comment. The label is a name for the trait, method and unit, and the comment is a human-readable description of what the label means. For example, the case study defines the following labels and comments:

```
trait_data = {
  "label": "Diseased",
  "comment": "Whether or not the subject in question is diseased."
}

method_data = {
  "label": "direct_observation",
  "comment": "Value is assigned by a human through direct
observation."
}

unit_data = {
  "label": "dimensionless",
  "comment": "arbitrary units"
}
```

The case study method and unit is particularly simple, since these values are determined by human observation (given the label “direct\_observation”) and do not have a unit of measurement (described in with the label “dimensionless”). However, numerical values can be similarly specified and furnished with the correct methods and units. For example, (trait) fruit weight might be measured by means of a scale (mentioning the mechanism by which the scale is measuring the weight) with grams as units.

Having created the trait, method and unit, the script then creates a variable. The body parameter is similar to those used to define traits, methods and units, except that there are URIs for the variable’s trait, method and unit as well.

```
variable_data = {
  "label": "Diseased",
  "comment": "Does the scientific object carry a disease?",
  "trait": the_trait["uri"],
  "method": the_method["uri"],
  "unit": the_unit["uri"]
}
```

Finally the data itself is posted for each scientific object using the **POST /data** service. This service requires URIs for provenance, scientific object and the variable. Further including the date and time uniquely identifies the data point. Finally, the value being saved is specified. The body parameters have the form

```
{
    "provenanceUri": provenance_uri,
    "objectUri": object_uri,
    "variableUri": variable_uri,
    "date": date_and_time,
    "value": value
}
```

In the case study, value is either “diseased” or “healthy”, referring to label of a specific plant specified using **objectUri**.

Having demonstrated the submission of point data to record ground truth data in the case study, we now show the uploading of bulk data files for handling the hyperspectral data portion of the dataset.

#### 4.8.5. Uploading the case study data: Image data

The case study data consists of a set of hyperspectral images obtained from an FX10 camera. The bulk data subsystem is most appropriate for these hyperspectral images. We will discuss uploading these images to the PHIS database.

The dataset is composed of hyperspectral scans where each scan is stored as an individual *directory* of files serving different purposes. To ease uploading of the dataset, individual directories are provided as ZIP files. We will upload these files to bulk storage, along with their metadata.

The uploading of individual images is performed by the script named **create\_hyperspectral\_images.py** . This script steps through each file in the directory containing ZIP files and submits the file using the **POST /data/file** service. Note that details of interaction with this service are implemented in the **post\_file\_object** function contained in **phis.py** file. As a body parameter, one provides:

```
{
```

```
'rdfType': rdf_type,  
'date': date,  
'provenanceUri': provenance_uri,  
'concernedItems': concerned_items,  
'metadata': metadata  
}
```

The `rdfType` identifies the type of file we are submitting using a URI. For a VNIR image, this is:

**<http://www.opensilex.org/vocabulary/oeso#VNIRImage>**

Furthermore a date and provenance URI is presented. A list of “concerned items” can be provided, which may for example include plant(s) present in the image.

Finally, the metadata entry provides a flexible means of specifying metadata relevant to the file type. In the example, we provide the sensor, project and experiment URIs, as well as the name of the original file as follows

```
{  
  "sensor" : sensor_uri,  
  "project" : project_uri,  
  "experiment" : experiment_uri,  
  "filename" : os.path.basename(file_path)  
}
```

This identifies the sensor used to upload the file, from where the sensor profile can be obtained.

The relationship between the file and containing project and experiment are also detailed. Finally, the original filename of the file is stored along with this metadata.

#### 4.9. Conclusion

In this chapter, we have considered uploading of a case study dataset to a PHIS server. This included core metadata such as the project description, as well as the actual measured data and ground truth. Ground truth data was recorded using PHIS’ point data facility, while image data was recorded using PHIS’ bulk data facility. At this point, the dataset is now fully represented in the PHIS server.