Supporting the Import of Sensor Data into the Sensor Web

Study Project

Raimund Schnürer

M.Sc. Geoinformatics Student

Supervisors:

Dipl.-Geoinf. Simon Jirka

Dipl.-Geoinf. Eike Hinderk Jürrens

Examiners:

Prof. Dr. Werner Kuhn

Dr. Albert Remke

Contents

Abbreviations

CSV	Comma-Separated Values
EPSG	European Petroleum Survey Group
O&M	Observation & Measurements
OGC	Open Geospatial Consortium
SensorML	Sensor Modeling Language
SOS	Sensor Observation Service
SWE	Sensor Web Enablement
UTC	Coordinated Universal Time
XML	Extensible Markup Language

Abstract

The OGC Sensor Web Enablement architecture provides a standards framework for integrating sensor measurements into the World Wide Web. However, sensor data is available in various formats and people might not be entirely familiar with OGC standards. This project aims to close these gaps by designing and implementing a tool which helps to publish observations, stored in CSV files, on Sensor Web Services. A graphical user interface guides domain experts in specifying correct reference systems and in adding missing metadata. Finally, XML files compliant to OGC specifications will be sent to a Sensor Observation Service. There, measurements from different sources can be interrelated and advanced analyses be performed.

1. Introduction

Sensors all over the world produce millions pieces of data every day. In many cases, data sets are stored in proprietary formats and varying structures on local computers. However, with the rise of the World Wide Web and the possibility to combine measurements from different sources, the demand of standardization grew. Therefore, the Open Geospatial Consortium[1] (OGC) has defined a set of standards, the so-called Sensor Web Enablement [2].

But often, domain experts are not familiar with these specifications. Thus, this project aims to develop a graphical interface for computers which assists the user in converting sensor measurements stored in CSV files into XML files according to OGC standards. This eases the integration into the Sensor Web and ensures interoperability with existing Spatial Information Infrastructures.

Potential users of the application will be operators of sensor stations and owners of sensor data sets, such as universities, governmental organizations or private persons. They can contribute to increasing the overall amount of sensor data in the Sensor Web. This subsequently allows using these data sets for further processing and analysis.

The project was carried out within the scope of the European research project EO2Heaven[2]. Here, data sets, e.g. about pollutants, exist which shall be provided via Sensor Web services. The interrelation of this data with other sources will help to gain a deeper insight into the relationship between environment and human health. Beyond that, temporal analyses on the data can be performed which might result in a better understanding of the impact of human activities on climate change.

2. Requirement Analysis

2.1. Users

The application is targeted at a quite specific audience. As potential user groups, universities, governmental organizations and private persons are considered who operate small to medium sensor networks. Those hold either real-time or archived sensor data which is often not standardized yet. Instead, heterogeneous formats - often CSV files - are used for performance or simplicity reasons. When users decide to make their data accessible for others, the application developed in this project will support this process.

In connection with the European research project EO2Heaven, two partners there will make use of the application. First, this is the Council for Scientific and Industrial Research[3] (CSIR) in South Africa, performing research in the fields of biology and geosciences. Second, the departments of Geodesy and Geoinformation technology from the Technical University in Dresden[4] are interested in this approach. But also other initiatives, like the Global Earth Ob-

[1] http://www.opengeospatial.org/ogc
[2] http://www.eo2heaven.org/node/2
[3] http://www.csir.co.za/about_us.html
[4] http://tu-dresden.de

servation System of Systems[5] (GEOSS), will benefit from this application. Here, the developed tool is a contribution to the Architecture Implementation Pilot 4.[6]

For an efficient and successful usage, the application poses some minimum requirements to the users. First of all, users should have some background knowledge of the data to be published. This is required in case not all metadata, which is necessary for an upload to a Sensor Observation Service, can be found in the CSV file. Users do not need to know the OGC specifications, but should have a general understanding of the Sensor Web. It would be helpful when users have a slight geographic and computer scientific background and are familiar with office applications.

2.2. Use Cases

Two different use cases can be distinguished for the application.

In the first scenario, users have a CSV file at hand, which contains sensor measurements and metadata. They would like to import it into a Sensor Observation Service for the first time. For this, the application reads in the CSV file and guides the user in adding missing metadata which is required by OGC standards (e.g. sensor names and reference systems). After that, the application compiles the beforehand collected data and generates XML files conforming to the Sensor Web Enablement. These XML files are then sent automatically to the Sensor Observation Service.

In the second use case, users want to upload another CSV file of similar structure to the Sensor Observation Service. For this purpose, the application has saved some user-defined settings entered during the first-time usage (e.g. features of interest and parsing options). Additionally, users can adapt small changes in respect of how new and old CSV file vary from each other.

2.3. Quality requirements

Quality	Rating	Meaning
Functionality	good	The program offers all features which are needed to complete the task. Some additional features enhance the usability.
Reliability	normal	Some small errors might still be in the application. However, the main progress should not be influenced by them.
Efficiency	normal	Response times of the application should lie in a normal range. Minor delays may occur while processing the data.
Usability	very good	The application navigates the user through the different steps, gives contextual help and is easy to learn.
Expandability	good	The application should be structured and programmed in a way that new features can be added with not much effort in future.
Portability	good	The application should be executable on common operating systems.

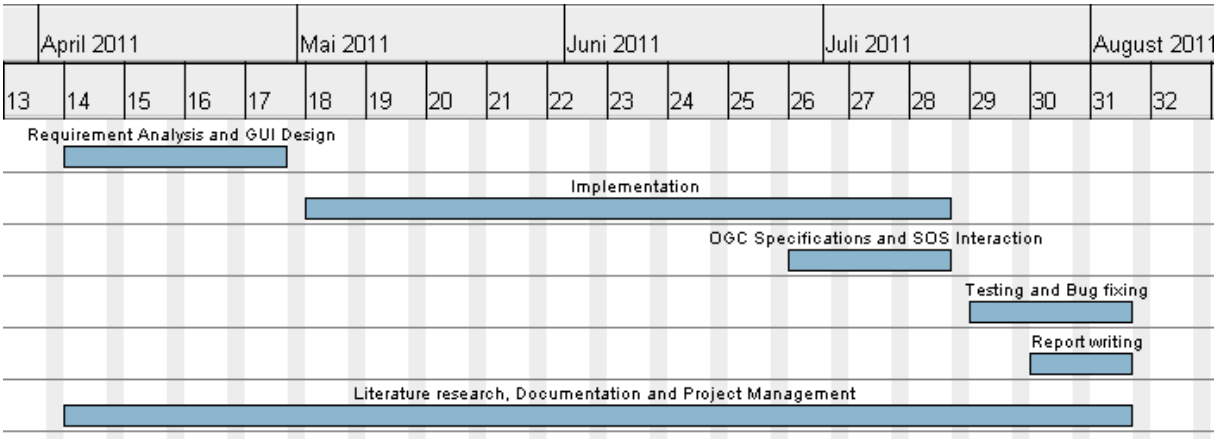
[5] http://www.earthobservations.org/geoss.shtml
[6] http://www.ogcnetwork.net/node/1677#EO2HEAVEN

3. Conception

3.1. Project phases

The project comprised a total amount of 150 hours of work. It was carried out during the time from April 4, 2011 to August 5, 2011.

In the Gantt chart below, it is shown which stages the project underwent:



3.2. Input Data

Comma-Separated Values (CSV) files are considered as the input for the application. CSV files are a common approach to represent tabular data in textual format. Outside the Sensor Web, they are often used to exchange sensor data. The structure of CSV files has never been standardized, though there are some guidelines [6]: Columns are separated with a consistent character and rows are separated with newlines. Traditionally, columns are delimited by a comma. In countries where decimal numbers are already separated with commas, other characters - like the semicolon or colon - are taken. Fields that contain a special character (e.g. a column separator or a newline) are enclosed by two quote symbols. Often, double quotes are used here. If a field includes a quote symbol, it is escaped by placing another quote symbol in front (e.g. the “”best“” for the “best”) [7]. Sometimes, comment characters (e.g. #) may appear which escape a whole line.

It is assumed that the content of the CSV file - measurements and sensor metadata - is stored column-wisely. For example, it could look like this:

01.01.2011; Sensor1; 1,3
01.01.2011; Sensor2; 2,5
02.01.2011; Sensor1; 1,7
02.01.2011; Sensor2; 2,2
…

At least one column of measurements should be included in the file. Otherwise, for plain sensor descriptions, the SID creator [3] could be used. The results of measurements can be numeric, positive integers (counts), Booleans (true or false) or textual. For numeric values (e.g. 2.3), thousands and decimal mark have to be specified in addition. This is necessary since regional differences exist for these separators. For example, in Germany a decimal comma is used, whereas the United Kingdom and the United States use a decimal point.

Optional to the measurement column, there can be one or more date and time column in arbitrary format. Date and time could be split over several columns (e.g. date in one, time in another one). Moreover, date and time may be not complete according to OGC standards. For instance, there are only times and no dates, or the time zone is missing. Similar applies to position columns. Their format can also be arbitrary, so patterns for converting them into a standardized scheme have to be specified. Position data could be available partitioned in several columns (e.g. latitude in one, longitude in the other one) and may not be complete. For example, only latitude and longitude are given and neither height, nor a spatial reference system. Also, the units of latitude, longitude and height might be missing.

It is assumed that within one column the same format is used (e.g. comma as a decimal separator or throughout the same date and time pattern). Other values than that (e.g. NULL-values or headings) will be ignored. Further columns in the CSV file might include units of measure, observed properties, sensors or features of interest. It is not very likely that URIs can be found here, so these columns will be interpreted as names. In the scope of this project, only stationary (not mobile) and in-situ (not remote) sensors are taken into account.

3.3. Workflow

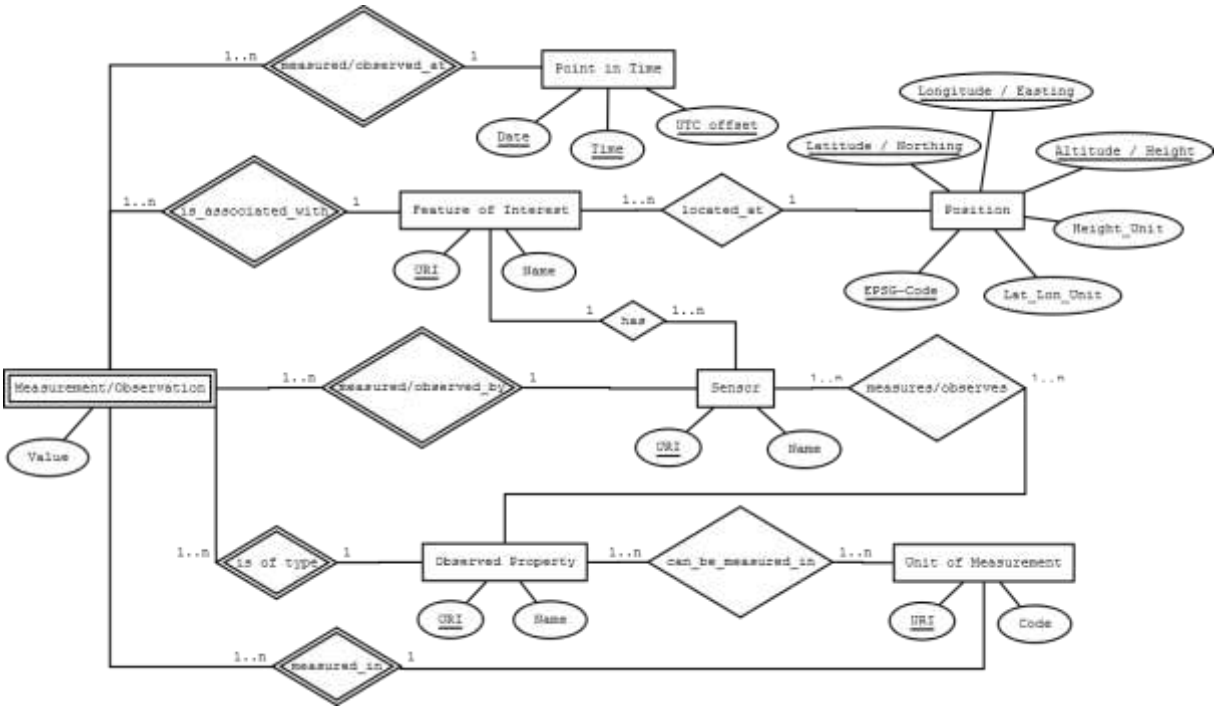
The application will make use of the wizard design pattern which guides the user through different steps. These and their purposes are briefly characterized in the table below. Screenshots to all steps can be found in the Appendix.

Step	Function
Step 1	Choose a CSV file from the file system to import
Step 2	Provide a preview of the CSV file and select settings for parsing (e.g. which character is used for separating columns)
Step 3	Display the CSV file in tabular format and assign metadata to each column (e.g. indicate that the second column consists of measured values). Offer customizable settings for parsing (e.g. for date/time patterns)
Step 4	In case of more than one date/time, feature of interest, observed property, unit of measurement, sensor identifier or position has been identified in step 3, point to measured value columns where they correspond to (e.g. state that date/time in column 1 belongs to the measured values in column 3 and date/time in column 2 belongs to the measured values in column 4). When there is exactly one appearance of a certain type, automatically assign this type to all measured values
Step 5	Check available metadata for completeness and ask the user to add information in case something is missing (e.g. EPSG-code for positions)
Step 6	When there is not any metadata of a particular type present in the CSV file (e.g. a sensor), let the user provide this information (e.g. name and URI of this sensor)
Step 7	Enter the URL of a Sensor Observation Service where measurements and sensor metadata in the CSV file shall be uploaded to
Step 8	Assemble all information from previous steps and convert the CSV file into XML files according to OGC's Observations & Measurements and SensorML specifications. Register sensors and insert observations at the given Sensor Observation Service. Show the progress and provide a report of errors and success

Note that steps 4 to 6 can sometimes be skipped for CSV files of a particular structure. Also, their functionality is split up in the application according to the type of metadata they represent (a – date/time, b – features of interest, observed properties, units of measurement, sensors, c – positions).

3.4. Output Data

The following Entity-Relationship Model shows the resulting data structure:



All those entities and their attributes are required for generating standardized XML files according to Sensor Web Enablement specifications. Not included in the model are settings for parsing (e.g. position patterns and numeric value separators) since those will not be appear in the final XML documents.

At the end, about each measurement/observation is known:

- which value was measured (e.g. 2.3)
- when it was measured (e.g. 01.01.2011 00:00:00) and in which temporal reference system (e.g. UTC +1)
- what the unit of measure was (e.g. °C)
- which property was observed (e.g. temperature)
- which sensor has measured it (e.g. thermometer XY)
- which feature of interest was examined (e.g. weather station at Robert-Koch-Str. 28)
- where it was measured (51°57′35.5″ N, 7°36′25.3″ E, 71m) and in which spatial reference system the location is provided (e.g. EPSG-Code 4326)

Optional or substitutional to the name of a feature of interest, observed property, unit of measurement or sensor, a URI could be given. This would help to semantically anchor the particular concept. A URI could be a URN, e.g. urn:ogc:def:phenomenon:OGC:1.0.30:Temperature for the observed property, or a URL, e.g. http://www.uni-muenster.de/Klima/wetter/stations_besch.html for the feature of interest.

3.5. Sensor Web Enablement

The Open Geospatial Consortium has developed amongst others a standards framework for sensor data: the Sensor Web Enablement (SWE). This was necessary since nowadays more and more sensors are connected via the World Wide Web, creating the so-called Sensor Web. SWE defines therefor a set of standardized interfaces and data models. With those, real-time or archived sensor data can be discovered and accessed through the World Wide Web. On this way, SWE ensures interoperability between heterogeneous sensor systems.

This study project makes use of three SWE standards: Observations&Measurements (O&M) [4], Sensor Model Language (SensorML) [1] and the Sensor Observation Service (SOS) [5]. O&M defines "standard models […] for encoding observations and measurements from a sensor" whereas SensorML contains "XML Schema for describing sensors systems and processes" [2]. After having gathered all metadata, O&M and SensorML XML files will be created. Then, the operations InsertObservation() and RegisterSensor() are used to upload these files to a Sensor Observation Service.

4. Implementation

4.1. Programming language

The complete application is written in the programming language Java. For the graphical user interface, Java's Swing API[7] is used.

4.2. Architecture

For a better structuring of classes and packages, the Model-View-Controller (MVC) pattern[8] has been applied. Here, the model comprises all transient and persistent data to be saved. This concerns all items in drop-down lists, like EPSG-codes. Also, the user's selections, for example metadata in a particular column, are stored in the model. To make the data of the model visible to the user, the view layer is used. The view has been implemented with Java's Swing toolkit. This made it easy to create UI components, like the table in steps 3 to 6 and the progress bars in the last step. The controller acts as the "glue" between model and view. Here, the communication between model and view takes place and complex operations are performed. For instance, generating the final XML documents and sending them to the Sensor Observation Service belong to the tasks of the controller. Moreover, the controller guides through the different steps and decides, whether they are needed or not. Nearly each step in the application has its own model, view and controller. By inheritance, common aspects of these steps have been generalized. Among them are for example the actions which are triggered when clicking on the next or the back button.

[7] http://java.sun.com/javase/technologies/desktop/
[8] http://www.oracle.com/technetwork/articles/javase/mvc-136693.html

4.3. Libraries

Besides the Log4j[9] library for logging information messages and errors, two other application programming interfaces have been used. The first one is opencsv[10] which parses CSV files taking account of the user-defined settings in step 2. The other API is Apache's HttpClient[11]. This library is used to send the generated XML files via the HTTP POST method to the Sensor Observation Service. All three libraries are under Open Source license. Other functions were realized with internal packages of Java. To be highlighted here is the parsing of dates and positions on the basis of a certain pattern as well as the recognition of numeric values with different decimal and thousands separators. This was implemented with the help of the classes SimpleDateFormat, MessageFormat and DecimalFormat of the package java.text.

4.4. Persistent Data

Three types of data can be distinguished which are persistently saved by the application:

4.4.1. Implementer-defined data

This data is solely provided by the implementer and cannot be modified by the user. The reason for this is that there are only a few possibilities or there are standardized formats. Latitude and longitude units (e.g. decimal degrees) and height units (e.g. meters) can be listed here. Also, EPSG codes and spatial reference system names can be assigned to this category. At the moment however, only a small choice of them is selectable. In future, they could possibly be enriched with data from the EPSG database.[12] Similar applies to UTC offsets. For testing purposes, these are currently available in a range from -12 to 12. In a next version of the application, time zone information for different cities/countries could be added from the tz database.[13] Lastly, decimal separators (e.g. ,) and thousands separators (e.g. .) for numeric values can be classified as implementer-defined data. Since there are not many of them, these should be more or less complete.

4.4.2. Implementer- and user-defined data

For some data elements, the implementer can offer a number of suggestions. These can be either accepted by the user or not. In the latter case, users can enter their own values. At that place, date and time formats (e.g. dd/MM/yyyy HH:mm) and position formats (e.g. LAT, LON, ALT) shall be stated. Also, observed property names and URIs as well as unit of measurement codes and URIs fit to this category. Similar to the previous subchapter, the list of choices could here be further extended. For instance, unified codes for units of measure[14] can be found. Since the format of CSV files has not been standardized, some degrees of freedom are left to the user. However, some common settings can be offered for the column separator (e.g. , ; : and Space), the text qualifier (e.g. #) and the comment indicator (e.g. "). Last in this group, there are the URLs of Sensor Observation Services. 52°North's Test SOS[15] is exemplary inserted here among others.

[9] http://logging.apache.org/log4j/
[10] http://opencsv.sourceforge.net/
[11] http://hc.apache.org/httpcomponents-client-ga/index.html
[12] http://www.epsg.org/
[13] http://en.wikipedia.org/wiki/Tz_database
[14] http://aurora.regenstrief.org/~ucum/ucum.html
[15] http://giv-sos.uni-muenster.de:8080/52nSOSv3/sos

4.4.3. User-defined data

To this category belong data items whereof the implementer has no information about. So, they have to be manually given by the user. This concerns for example sensor names and URIs as well as feature of interest names and URIs. Those differ from user to user. They are saved in case the user wants to import another CSV file of a similar structure. Then, he or she does not need to make all inputs again. In a next release of the application, positions of features of interest might be stored here as well.

4.5. Start

The application is packaged as JAR file and can be started with the enclosed batch file. In the batch file, you have to specify the path to the “bin” folder of the local Java Runtime Environment.

4.6. Test

The application has been tested with two real CSV files provided by two partners from EO2Heaven. In both cases, the software was able to successfully publish sensor measurements and sensor metadata to instances of 52°North’s Sensor Observation Service[16].

The first sample file consists of a date and time column and two measured value columns for different observed properties:

Date & Time	SO2 (ppb)	TRS (ppb)
01/06/2010 00:00	1,3	3,9
01/06/2010 00:05	1,5	4,8
01/06/2010 00:10	1,6	5,3
01/06/2010 00:15	1,5	5,6

Here, the date and time pattern “dd/MM/yyyy HH:mm” and comma as a decimal separator have to be chosen. Date and time are then automatically assigned to both measured value columns. Only seconds and time zone have to be additionally specified. For each measured value column, the user enters name or URI for the feature of interest, observed property, unit of measurement and sensor. Lastly, the position of the feature of interest is requested since there are not given any coordinate values in the CSV file.

The second exemplary file contains a column for date, feature of interest, observed property and measured value:

Datum	Station	Komp	Wert/myg/m3
01.11.03	Klinthal	PM10	19.94
02.11.03	Klinthal	PM10	22.61
03.11.03	Klinthal	PM10	13.10
01.11.03	Zwickau	PM10	20.06
02.11.03	Zwickau	PM10	11.15
03.11.03	Zwickau	PM10	14.72

For a conversion into an XML file compliant to SWE standards, the date is assigned the pattern “dd.MM.yy” and supplemented with a time of the day and the corresponding time zone.

[16] http://52north.org/communities/sensorweb/sos/

Next, the unit of measurement has to be chosen for the measured value column. Because sensor identifiers are also missing in the CSV file, they have to be determined for each feature of interest and observed property tuple. Finally, the user supplies locations for each feature of interest listed in the column.

Note: Columns in the first CSV file were delimited with a semicolon and in the second CSV file by space character. For a better readability, they have been represented as a table instead of a raw text file.

5. Conclusion and Future Work

5.1. Summary

In this study project, an application has been developed which helps to import sensor measurements and metadata, archived in CSV files, into a Sensor Observation Service. Guided by a step-by-step wizard, users identify existing and add missing metadata elements which are required by Sensor Web standards. Finally, all gathered information is compiled, converted into XML files and uploaded to a Sensor Observation Service.

Two CSV files provided by partners of EO2Heaven have been successfully integrated into a Sensor Observation Service. CSV files in other formats can be handled by the application as well, but there are still some limitations. For example, only column-wisely oriented CSV files can be processed and not every piece of information in the file can be extracted yet. How these and other things can be included in the application is discussed in the next subchapter.

5.2. Outlook

In future, a series of new features can be added to increase the overall usability of the application. These are described in the following, first for individual steps and then for the application in general.

To support further file formats besides CSV, for example non-standardized XML files, Step 1 has to be modified. Depending on the particular file extension, other types of wizards have to be implemented and called. Step 2 could be supplemented with a check-box indicating whether data in the CSV file is stored column-wisely or row-wisely. When data is oriented in a row-wise manner, it would make sense to identify metadata in step 3 first for the rows. It is also thinkable that a piece of information appears just in one single cell (e.g. the date). Step 3 then needs to be equipped with the functionality to extract metadata also from cells. Beyond, it should be possible to exclude single rows from export (e.g. the heading). For both step 2 and 3, reasoning about the values can be applied so that separators and patterns are recognized automatically. For instance, the column separator could be determined by counting how many times possible characters (e.g. the semicolon) occur in the CSV file. Then, the character with the most appearances can be chosen. Furthermore, step 3 could offer the function to parse metadata combinations. This would be useful when a field consists of two or more types of metadata, like observed property and unit of measurement. With “SO2 (ppb)” and “TRS (ppb)”, this was the case in the first tested CSV file.

In step 4, a controller has to be added which solves the ambiguity between more than one position column and more than one feature of interest column. Up to now, this has not been implemented since this case is very unlikely. When Linked Data becomes more important, step 5 needs to be enhanced with a model and a controller for finding the corresponding URI for a name (e.g. of a sensor). So far, either the name or the URI was sufficient. Step 6 could be equipped with an input verifier for positions which checks for a given spatial reference system valid ranges and units for latitude and longitude. Also, it would be nice to choose and to display the position on a map. It could be advantageous to move step 7 some steps forward. Then metadata for sensors, which are already registered at the SOS, would not have to be entered by the user again. Lastly, step 8 needs to provide further XML templates for a SOS import (e.g. for category observations and mobile sensors). At present stage, only stationary sensors and measurements with numeric values are supported.

The most important feature to improve the application on the whole would be saving all inputs and choices made by the user in a single settings file. With this file at hand, users should be able to upload CSV files of same structure to the SOS without specifying anything else. With the information available in the settings file, CSV files should then be converted automatically into standardized XML files by the application.

Additionally, some smaller functions might facilitate the usage of the application. For example to support different languages, all text appearing in the source code has to be externalized (e.g. for labels, buttons and messages). So far, this has been realized exemplarily for tooltips. As mentioned in chapter 4.4., more predefined EPSG-codes and time zones as well as observed properties and units of measurement should be available for selection. Possibly a database has to be installed for this purpose. A navigation bar could highlight the current progress of the user within the application (e.g. showing the percentage of completeness). The “Next” button should only be enabled when all necessary information for a step has been collected. Moreover, it has to be checked whether a step is still valid when going back and forward again. Finally, it could be beneficial - with regard to the second use case in chapter 2.2 - to save the last choice of the user in dropdown menus persistently.

Acknowledgments

I would like to thank my two supervisors, Simon and Eike, who gave me valuable advice throughout the project.

References

[1] Botts, M. and Robin, A. OpenGIS® Sensor Model Language (SensorML) Implementation Specification, 2007. Retrieved June 30, 2011, from Open Geospatial Consortium, Inc.: http://portal.opengeospatial.org/files/?artifact_id=21273

[2] Botts, M., Percivall, G., Reed, C. and Davidson, J. OGC® Sensor Web Enablement: Overview And High Level Architecture, 2007. Retrieved June 30, 2011, from Open Geospatial Consortium, Inc.: http://portal.opengeospatial.org/files/?artifact_id=25562

[3] Bröring, A., Bache, F., Bartoschek, T. and van Elzakker, C. The SID Creator: A Visual Approach for Integrating Sensors with the Sensor Web, 2011. in Geertman, S., Reinhardt, W., Toppen, F. ed. Advancing Geoinformation Science for a Changing World, Springer-Verlag, Berlin Heidelberg, 2011, 143-162

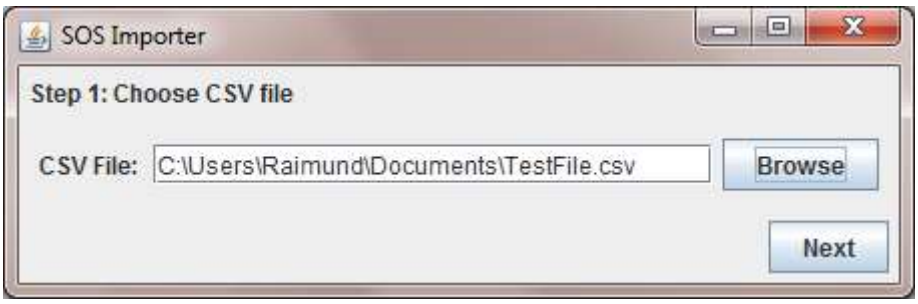
[4] Cox, S. Observations and Measurements - XML Implementation, 2011. Retrieved June 30, 2011, from Open Geospatial Consortium, Inc.: http://portal.opengeospatial.org/files/?artifact_id=41510

[5] Na, A. and Priest, M. Sensor Observation Service, 2007. Retrieved June 30, 2011, from Open Geospatial Consortium, Inc.: http://portal.opengeospatial.org/files/?artifact_id=26667

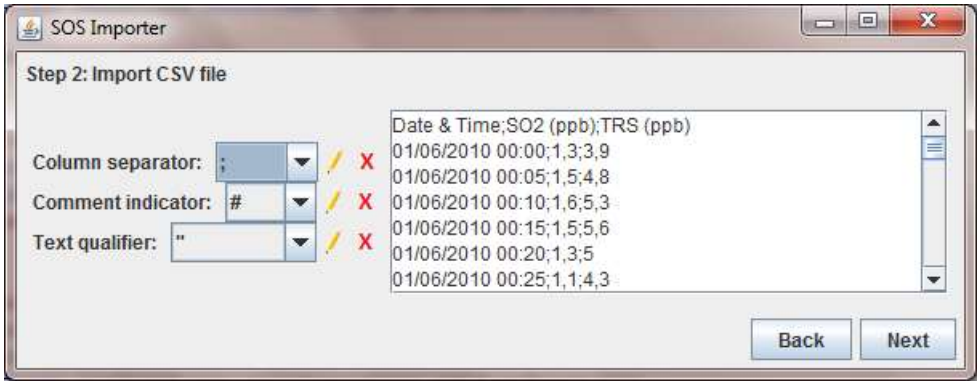
[6] Shafranovich, Y.: Common Format and MIME Type for Comma-Separated Values (CSV) Files, 2005. Retrieved April 27, 2011, from The Internet Society: http://tools.ietf.org/html/rfc4180

[7] Wikipedia: Comma-separated values, 2011. Retrieved April 27, 2011 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Comma-separated_values

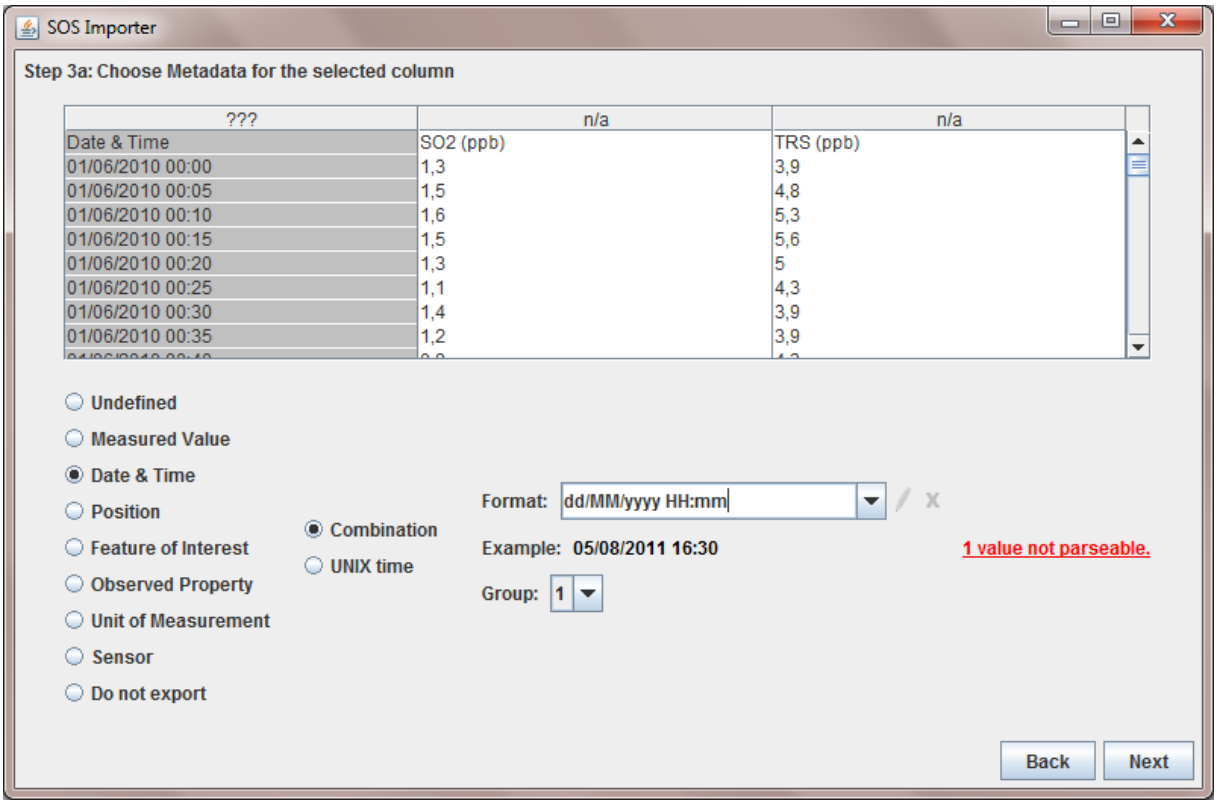
Appendix



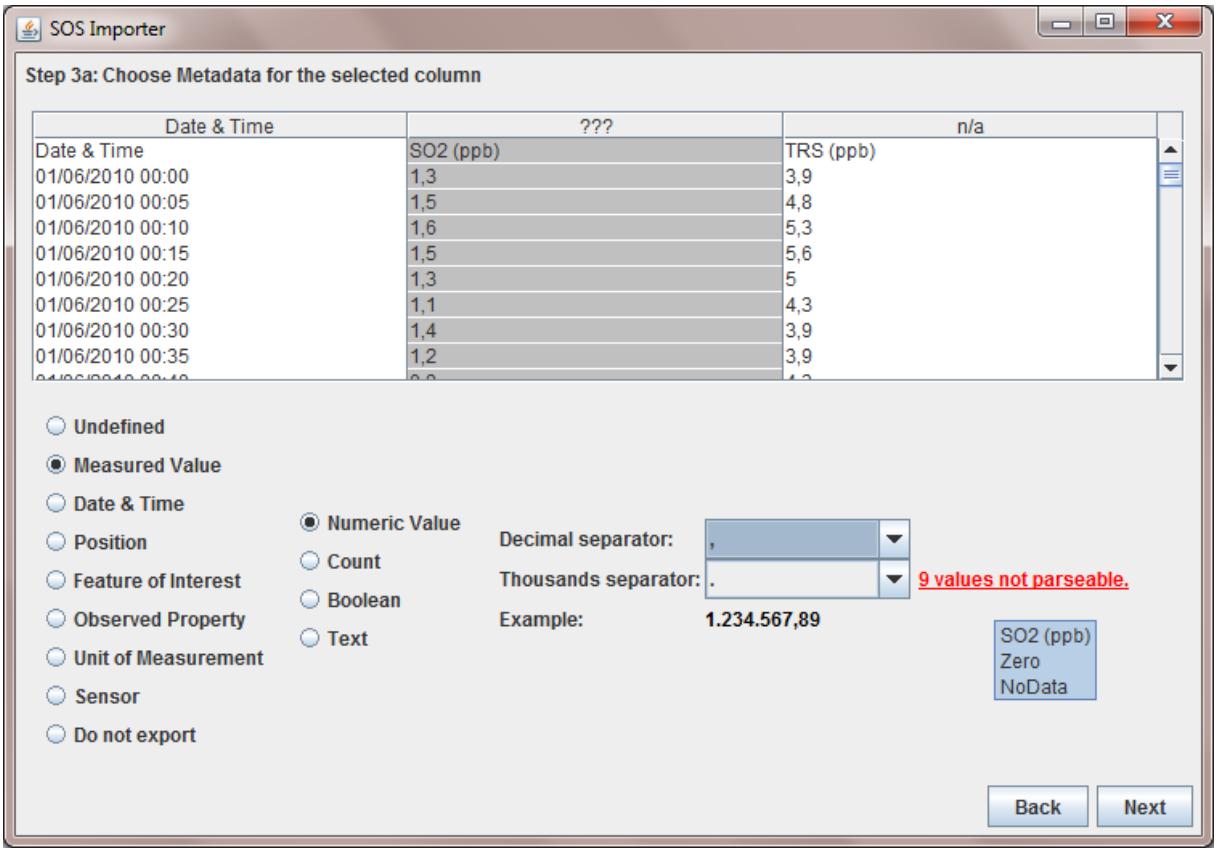
Step 1: Choosing a CSV file



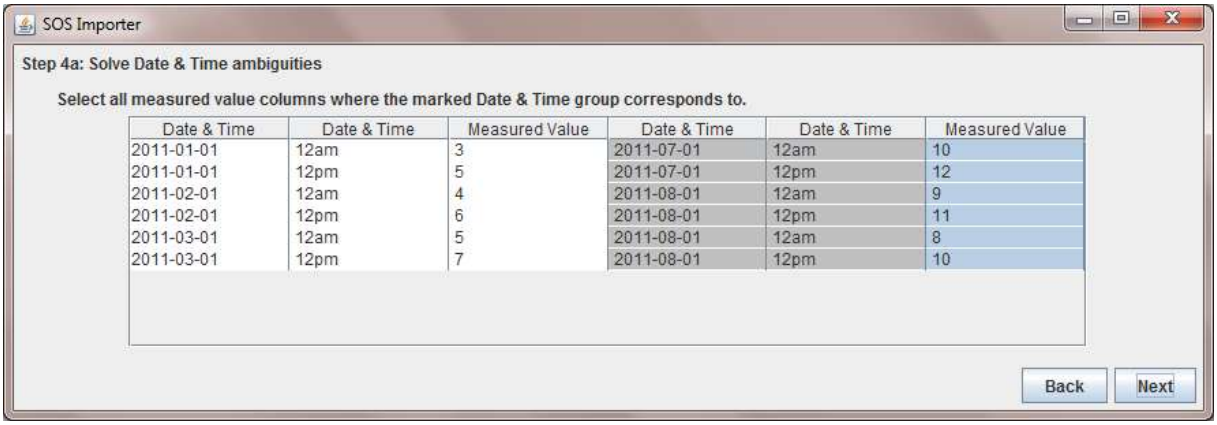
Step 2: Selecting settings for parsing the CSV file



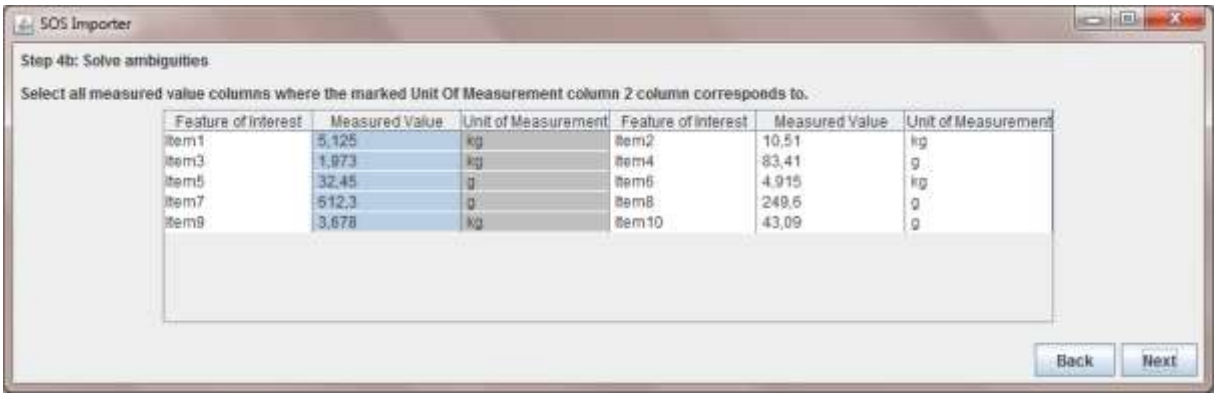
Step 3: Entering a date&time pattern



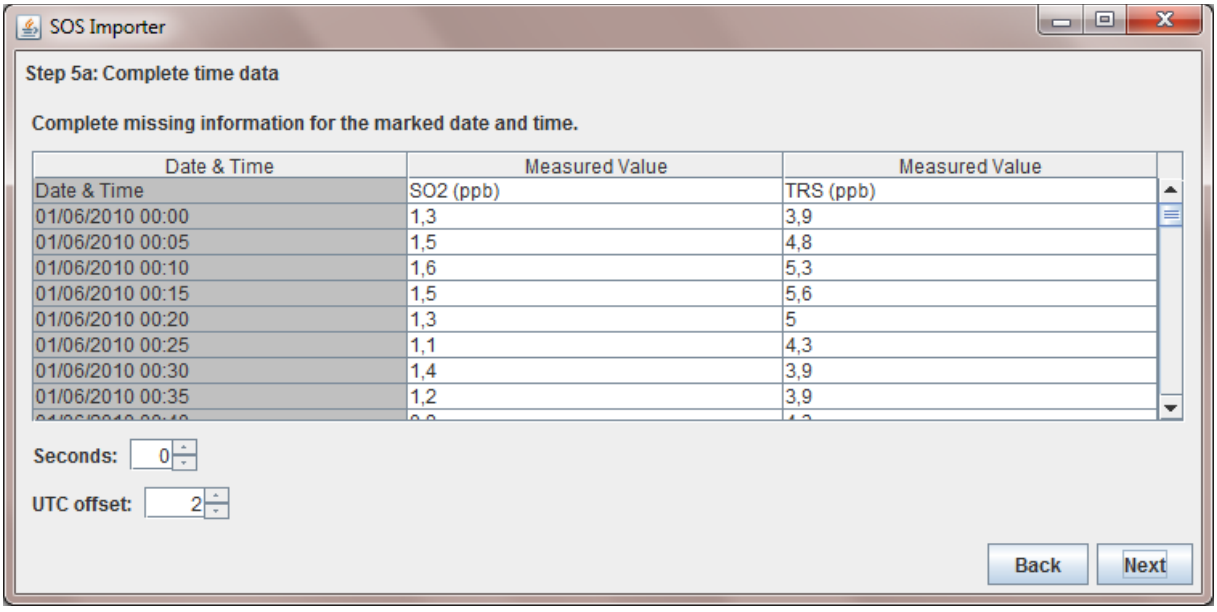
Step 3: Showing non-parseable values



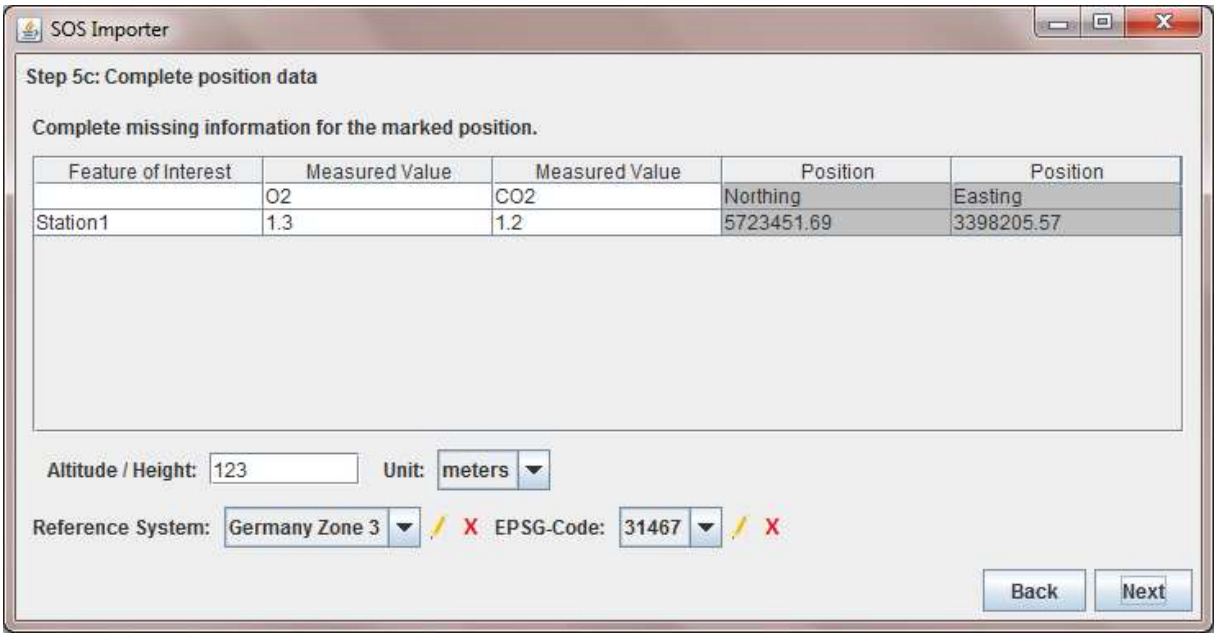
Step 4: Selecting a measured value column for a date&time group



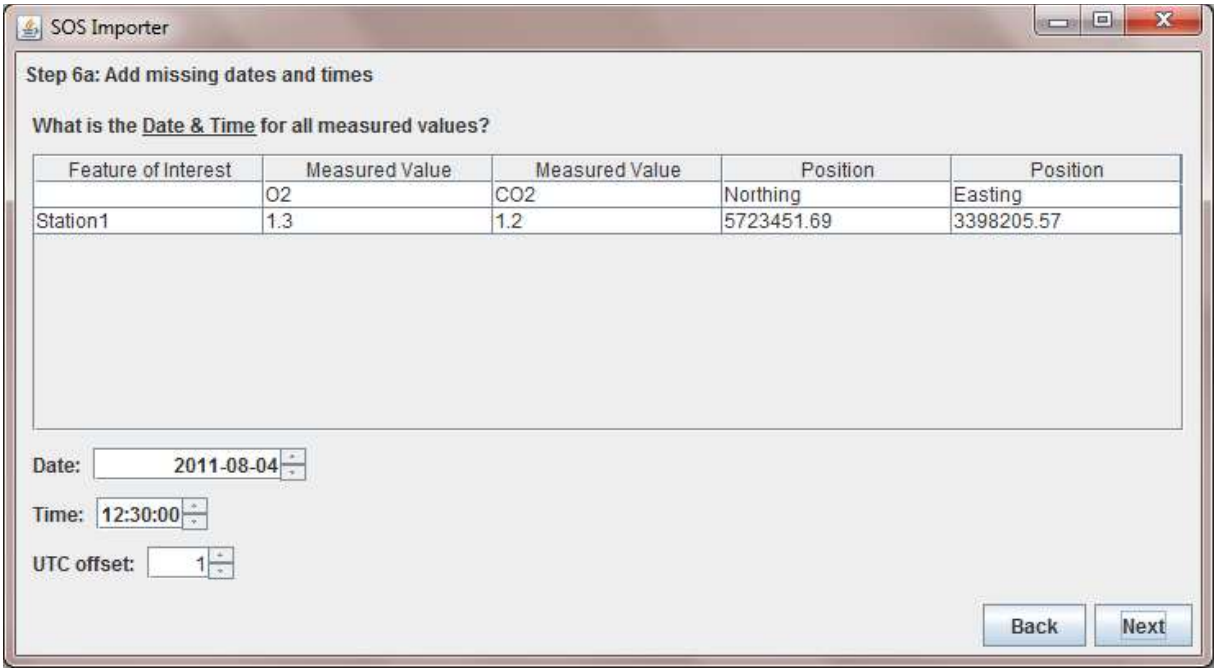
Step 4: Selecting a measured value column for a unit of measurement column



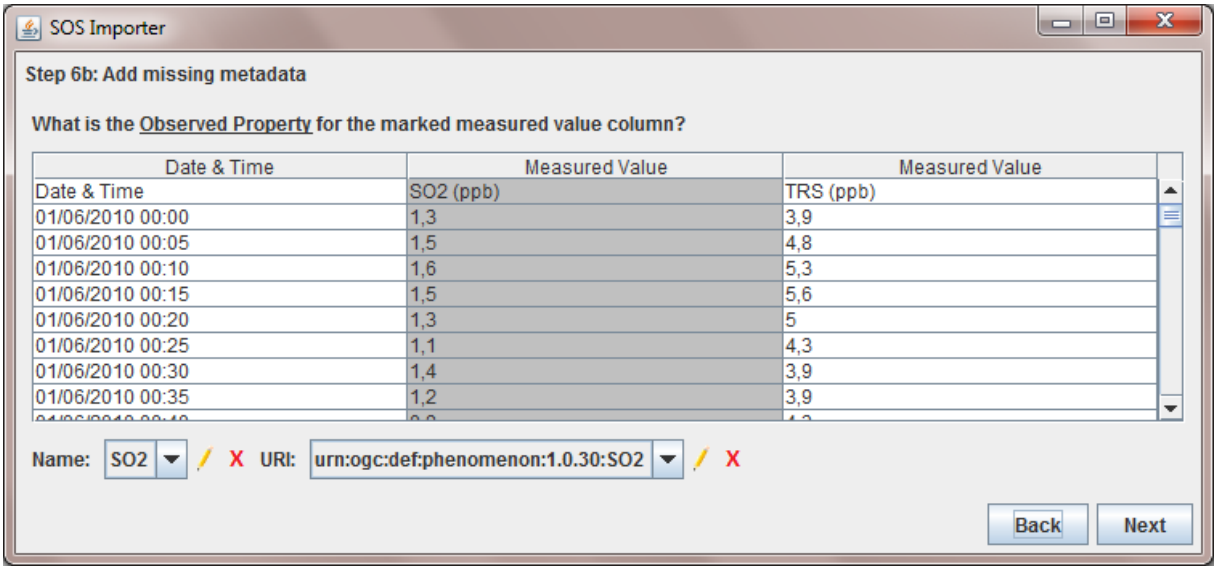
Step 5: Completing missing time metadata



Step 5: Completing missing position metadata

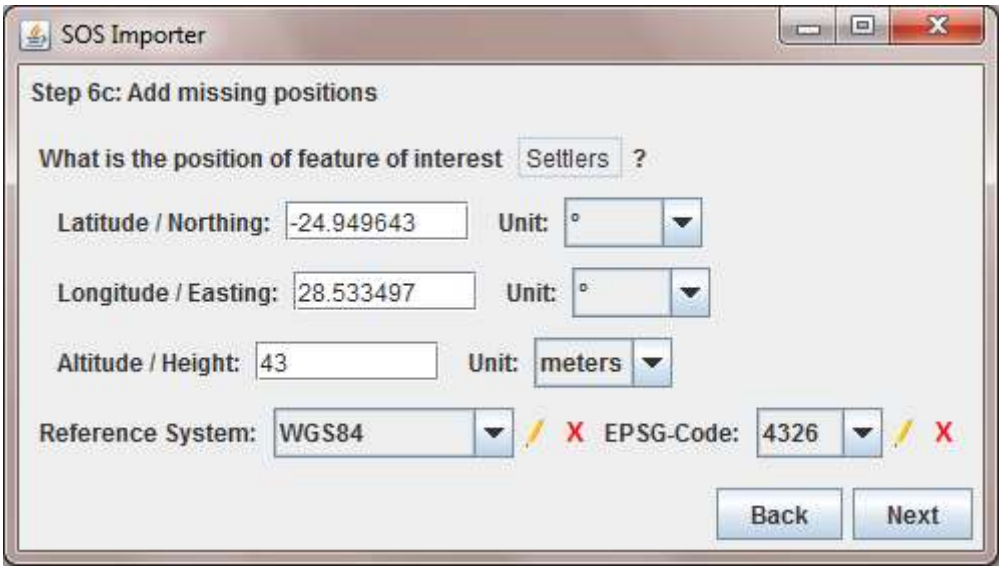


Step 6: Choosing a time for all measured values



Step 6: Selecting name and URI of the observed property
for the highlighted measured value column

Step 6: Selecting name and URI of the sensor
for the given feature of interest and observed property



Step 6: Requesting positional information
for the given feature of interest

Step 7: Choosing the URL of a Sensor Observation Service

Step 8: Registering sensors and inserting observations at the specified Sensor Observation Service

Step 8: Finished import