Environmental Modelling & Software 70 (2015) 32–44

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

Open source software for visualization and quality control of continuous hydrologic and water quality sensor data

Jeffery S. Horsburgh [a, *], Stephanie L. Reeder [b], Amber Spackman Jones [b], Jacob Meline [b]

[a] *Department of Civil and Environmental Engineering, Utah Water Research Laboratory, Utah State University, 8200 Old Main Hill, Logan, UT 84322-8200, USA*
[b] *Utah Water Research Laboratory, Utah State University, Logan, UT, USA*

A R T I C L E I N F O

Article history:
Received 9 January 2015
Received in revised form
10 March 2015
Accepted 9 April 2015
Available online 29 April 2015

Keywords:
Data management
Open source software
Sensor data
Python
Hydrologic observations
Quality control

A B S T R A C T

It is common for *in situ* hydrologic and water quality data to be collected at high frequencies and for extended durations. These data streams, which may also be collected across many monitoring sites require infrastructure for data storage and management. The Observations Data Model (ODM), which is part of the Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI) Hydrologic Information System (HIS), was developed as a standard data model in which to organize, store, and describe point observations data. In this paper we describe ODM Tools Python, an open source software application that allows users to query and export, visualize, and perform quality control post processing on time series of environmental observations data stored in an ODM database using automated Python scripting that records the corrections and adjustments made to data series in the quality control process and ensures data editing steps are traceable and reproducible.

Software availability

Name of software: ODM Tools Python
Developers: Jeffery S. Horsburgh, Stephanie L. Reeder, Amber Spackman Jones, Jacob Meline, and James Patton
Contact: jeff.horsburgh@usu.edu
Year first available: 2014
Hardware required: A personal computer
Software required: Microsoft Windows, Mac OSX, or Linux operating system
Software availability: All source code, installers, example ODM databases, and documentation for the ODM Tools Python software application can be accessed at https://github.com/UCHIC/ODMToolsPython.
Cost: Free. Software and source code are released under the New Berkeley Software Distribution (BSD) License, which allows for liberal reuse of the software and code.

1. Introduction

Environmental monitoring with *in situ* environmental sensors presents many challenges for data management, particularly for large-scale networks consisting of multiple sites, sensors, and personnel. Over the past decade, there has been a drastic increase in the use of automated data collection in scientific research. The high frequency, extended duration, and spatial distribution of data collection efforts require cyberinfrastructure to support and facilitate research using sensor data streams. Researchers and practitioners need tools for data import and storage as well as data access and management. In addition to addressing the challenges presented by managing the sheer quantity of data, monitoring network managers need practices to ensure high data quality, including standard procedures and software tools for data post processing and quality control.

In this paper we describe a workflow for scripted quality control editing of continuous, *in situ* time series datasets and the architecture and functionality of an open source software tool called ODM Tools Python that implements this workflow. ODM Tools Python enables users to query and export, visualize, and edit time

* Corresponding author. Tel.: +1 435 797 2946.
E-mail address: jeff.horsburgh@usu.edu (J.S. Horsburgh).

http://dx.doi.org/10.1016/j.envsoft.2015.04.002

series observations stored in an Observations Data Model (ODM) database (Horsburgh et al., 2008). ODM was developed as a standard data model in which to organize, store, and describe point observations (e.g., observations made at fixed point monitoring sites such as streamflow, water quality, and weather monitoring stations) with sufficient metadata for observations to be unambiguously interpreted by multiple users. ODM is implemented in relational database software to permit flexibility in querying and data retrieval. ODM Tools Python is a modernized and advanced version of the original ODM Tools software, which, along with ODM, was developed as part of the HydroServer software stack (Horsburgh et al., 2010) within the Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI) Hydrologic Information System (HIS) (Horsburgh et al., 2009; Tarboton et al., 2009).

Previous versions of ODM Tools (Horsburgh et al., 2011) were developed in Microsoft Visual Studio .NET and only supported ODM databases implemented in Microsoft SQL Server, limiting deployment to Microsoft Windows-based computers. Functionality consisted of exporting data series and associated metadata, plotting and summary of single data series, generation of derivative data series, and editing data series using a set of simple tools implemented within the ODM Tools graphical user interface (GUI). However, there were no capabilities for recording or storing the sequence of edits made to a data series in the quality control process. Furthermore, there were not adequate capabilities to support the full quality control workflow, especially for long time series that were difficult to process in a single editing session or time series continually being updated as new data were collected. These were major limitations that affected the usefulness of ODM Tools where recording the provenance of data edits was desired or required. Irrespective of the tool being used to perform quality control, it remains common for some scientists to perform these steps at the same time as data visualization and analysis without preserving the provenance of the quality control process in a way that it is accessible to others or for future reference. We sought to address this deficiency with the software described in this paper.

The new, Python-based version of ODM Tools described here adds a modernized GUI with dockable components, multiple platform support (Windows, Linux, and Mac), support for multiple relational database management systems (RDBMS) (Microsoft SQL Server, PostgreSQL, and MySQL), enhanced plotting and visualization, and automated scripting of quality control edits performed on data series through an integrated Python script editor and console. Additional improvements include enhanced queries for data selection and export, interactive data selection directly on a time series plot, additional options for how edits to data series are versioned and saved, and the ability to plot multiple data series simultaneously with various plot types. The contribution of this paper is in the description the quality control editing workflow for continuous sensor data streams and in the architecture of the ODM Tools Python software that implements the workflow and enables the features listed above. We anticipate that the functionality that we have developed in ODM Tools Python will be widely applicable to managers of continuous sensor data and that the architecture, software development approach, and deployment approach that we have used for the ODM Tools Python software will be informative for researchers developing similar tools.

In Section 2 we provide further background on quality control for hydrologic and water quality sensor data streams and associated challenges. Section 3 describes our workflow for supporting the quality control editing process. Section 4 describes our approach to address these challenges and the architecture and software implementation for ODM Tools Python. Section 5 describes the deployment approach we selected for the ODM Tools Python software, and Section 6 provides an example of using ODM Tools Python for quality control editing and post processing of a continuous water quality time series. Finally, we conclude with a summary of the research results.

2. Background

2.1. Quality control of sensor data

Errors in continuous environmental datasets primarily occur from fouling of sensors and from sensor calibration shift, although anomalies and erroneous data values can occur for many other reasons, including failures of sensors, recorders, transmission systems, or unforeseen environmental conditions that adversely affect sensor readings (Wagner et al., 2006). Producing high quality, continuous data streams from raw sensor output requires application of data corrections to mitigate for instrument calibration drift, fouling, and other errors (Mourad and Bertrand-Krajewski, 2002; Campbell et al., 2013). Fig. 1 shows common types of errors in raw sensor data streams and illustrates the associated types of corrections needed in the quality control process. These are also listed in Table 1, which provides a more extensive list of common error types along with examples and description of the most common procedures for correction. The errors listed in Table 1 are illustrative of those commonly encountered within raw environmental sensor datasets, and the software described in this paper was developed to advance data from raw to quality controlled versions using the corrections described in Table 1.

Correction procedures listed in Table 1 include simple methods such as: deletion of erroneous values, insertion of individual values to fill brief no-data gaps, and interpolation of erroneous values using prior and subsequent values that are known to be good. More sophisticated corrections include variable data corrections applied to correct for sensor drift and fouling. It is beyond the scope of this paper to describe the circumstances under which each of these corrections should be applied. Rather, we describe a workflow and software that enable application of any of these corrections according to the judgment of the data analyst in a way that the provenance of the changes can be saved. Wagner et al. (2006) provide excellent guidance and discussion on which correction procedures to use for various data collection situations encountered in hydrologic and water quality monitoring. Similarly, Fiebrich et al. (2010) review checks and correction procedures for common meteorological variables.

Another important distinction is that of “automated” quality control versus “interactive” quality control editing performed by an analyst. Most automated quality control procedures are focused on automatically flagging raw data values that do not meet one of several plausibility tests (e.g., Sheldon, 2008; Lerner et al., 2011; Taylor and Loescher, 2013). These include programmatic checks aimed at identifying instances of the types of errors described in Table 1 in raw data streams. Algorithms for identifying these errors range from simple range checks to more complicated techniques using statistical models or machine learning methods (e.g., Moatar et al., 2001; Hill et al., 2009; Fiebrich et al., 2010; White et al., 2010; Dereszynski and Dietterich, 2012). While these techniques can be quite good at identifying and flagging potentially erroneous values, they do not excel at choosing and applying appropriate data corrections to resolve the error for which the data were flagged, steps that generally require the attention and expertise of field or data technicians (Fiebrich et al., 2010; White et al., 2010). This paper, the workflow we present, and the ODM Tools Python software focus on this secondary level of quality control (or post processing) and provide a set of tools for applying the common data corrections listed in Table 1.

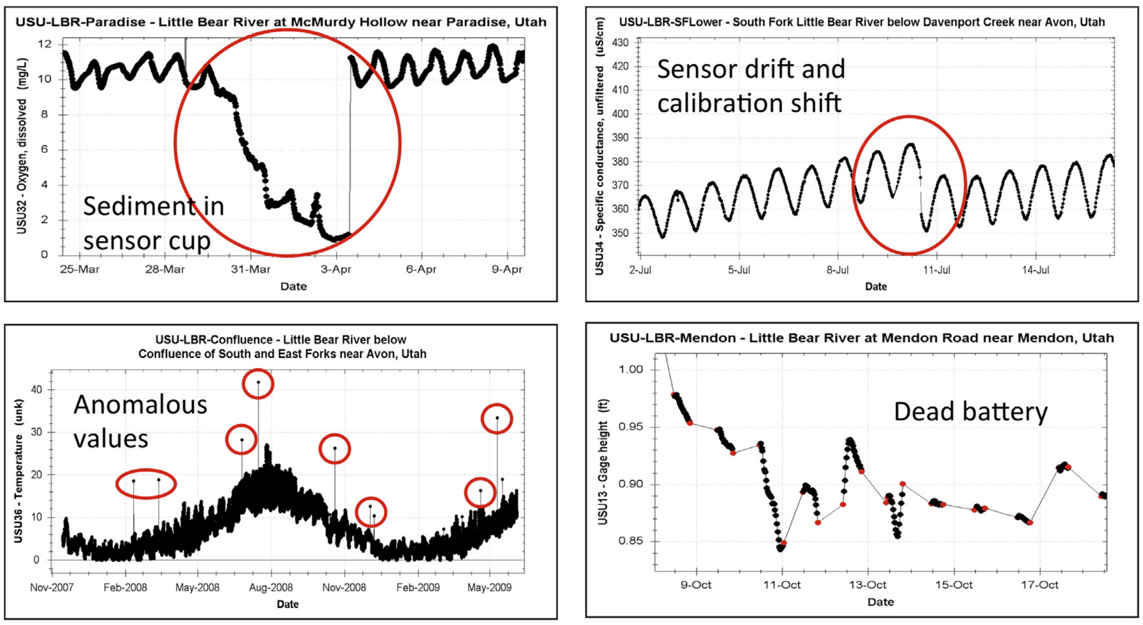


Fig. 1. Example anomalous sensor data.

2.2. *The role of visualization in data quality control*

Quality control of sensor data is often an iterative process that requires visualization of raw data to identify anomalies, to inspect modified data after application of quality control procedures to ensure corrections were correctly applied, and to examine additional datasets that provide context for an editing session. For example, in deciding whether anomalous values in a sensor dataset need to be edited, it is common to inspect data for the same or other variables at the same monitoring site (i.e., internal consistency) or to examine data for the same variable collected at a nearby monitoring site (i.e., external consistency) to determine whether corrections should be made (Moatar et al., 2001; Campbell et al., 2013). The iterative process of quality control editing and visualization can be tedious and challenging, especially when the tools used for each are not the same or where action is required to update a visualization each time an edit is made.

2.3. *Challenges in integrating tools for visualization and quality control of sensor data*

While most scientists use the general techniques described above to visualize and perform quality control of sensor data, there is large variety in the ways in which this is currently done. This heterogeneity arises because: 1) scientists work across computing platforms (e.g., Windows, Linux, Macintosh); 2) scientists have a spectrum of technical expertise with respect to automation of tasks like data quality control via computer programming; and 3) scientists use the software tools that they are most comfortable with and that meet their combination of technical expertise and computing platform. For example, some scientists open data and perform adjustments in Microsoft Excel using built in visualization capabilities to plot and verify the changes they have made. Some script their data edits using R, Python, or Matlab and use the native or imported plotting capabilities of these languages for visualization. Examples

Table 1
Common types of error in hydrologic and water quality sensor data streams.

Type of error	Description and examples	Typical correction method
Sensor drift	Error in sensor readings due to electronic drift in the sensor reading away from the instrument's calibration during the period between calibrations.	Variable data correction
Sensor fouling	Error in sensor readings due to biological growth or other residue buildup on the sensor between maintenance visits.	Variable data correction
Out of range values	Data values that are beyond the range of plausible values for the particular phenomenon being measured.	Deletion or interpolation and flagging
Data value persistence	Constant data values that are recorded when a sensor becomes stuck in a single position (e.g., in the case of a wind direction sensor) or when a sensor fails and the datalogger repeatedly records the last measured value.	Deletion
Failed sensor or logger	Erroneous data values recorded by a sensor or datalogger that has failed (e.g., a dissolved oxygen sensor fails and the datalogger records dissolved oxygen concentrations that drop to 0 mg/L).	Deletion
Power failure	A battery or power supply fails to supply required levels of power to a sensor or datalogger resulting in suspect measurements.	Deletion or flagging
Adverse site conditions	Conditions immediately surrounding the sensor are not representative of the site (e.g., sediment buildup in a sensor cup, ice buildup around a sensor).	Deletion
Incorrect offset or calibration	Data values that are in error by a constant value due to a datalogger program applying an incorrect offset or an error during sensor calibration.	Add or subtract a constant
Truncation	Data values recorded at the reporting limit for a sensor because its maximum or minimum recording level has been exceeded.	Deletion or flagging
Skipped or no-data values	Gaps in data caused by datalogger errors or skipped scans.	Insertion and/or interpolation and flagging

in this category include the sensorQC R package developed by the United States Geological Survey (https://github.com/USGS-R/sensorQC) and the Georgia Coastal Ecosystems (GCE) Data Toolbox for Matlab (https://gce-svn.marsci.uga.edu/trac/GCE_Toolbox/). Some scientists use available commercial software systems such as the Aquarius software package from Aquatic Informatics (http://aquaticinformatics.com/products/aquarius-time-series/) or the Kisters WISKI software (http://www.kisters.net/wiski.html), which have good functionality but can cost enough that they are out of reach for many small research groups. Some commercial products are also tied to specific instrument/data collection equipment manufacturers, limiting their general applicability. There are currently few software products designed specifically for quality control editing of continuous environmental time series data with integrated visualization capabilities. Meeting this need is challenging given the diversity of computing platforms and technical expertise among potential users.

Maintaining the provenance of data edits is another major technical challenge for data managers. It is common for data managers to maintain “raw” and “quality controlled” versions of their data, but it is less common for data managers to maintain a complete provenance record of how they moved from “raw” to “quality controlled” versions (Lerner et al., 2011; Campbell et al., 2013; Mason et al., 2014). Data edits made manually can produce quality controlled versions of data series, but since the edits are manual and not recorded, they can be impossible to reproduce. In contrast, scripting of quality control edits provides a written record of the various steps involved in the quality control process, effectively encoding data corrections in a way that they can be easily revised and re-executed at any time. Indeed, scripting of data management and analysis tasks has been recommended as a best practice (e.g., Borer et al., 2009), and some groups have begun implementing scripting to post process environmental data (Mason et al., 2014). Scripting is, however, difficult for many scientists that have no computer science background or expertise and subsequently lack the requisite computer programming skills.

2.4. Python as a platform for data quality control editing

Although we have pointed out that many engineers and scientists do not yet use scripting in their day-to-day work, the number of those that do is growing. For example, the use of Python for scientific computing on diverse computing platforms has increased significantly in recent years (Millman and Aivazis, 2011). Python provides a high-level, object-oriented, interpreted programming language that has been used extensively for exploratory, interactive, and computational scientific research. Major advantages of Python include its platform independence, the rich collection of functionality that can be added via extension packages, and the relative ease with which Python can be integrated with software written in other languages (Perez et al., 2011). For example, the Python data analysis library pandas (http://pandas.pydata.org) includes several easy to use and highly optimized data structures and analysis tools suitable for the types of analyses and functionality needed for performing quality control on continuous environmental data streams. Another advantage of Python is the ability to both work in scripting/command line mode and to develop complex software programs with GUIs. These characteristics made Python a good candidate for the development work described in this paper.

One difficulty in using Python for scientific computing and in software development is getting a complete set of Python tools needed for a particular project installed on a computer. This challenge remains, although new Python distributions such as Enthought Canopy (https://www.enthought.com/products/canopy/) and the Anaconda Python distribution (https://store.continuum.io/cshop/anaconda/) provide an entire Python ecosystem with many of the most popular Python packages for science, math, engineering, and data analysis within a Python environment that can be easily installed. In fact, many scientists maintain multiple Python environments on their computers for different purposes. Using Python for the software development effort described in this paper presented two deployment-related challenges that had to be overcome: 1) deploying the software we developed and all required Python components to computers running one of multiple operating system platforms; and 2) deploying the software in a way that it did not overwrite or conflict with a user's existing Python environments.

3. Data quality control editing workflow

Quality control of continuous sensor datasets is often an ongoing process (Mason et al., 2014). Some data collection efforts may continue for months or years, and it is not always feasible to wait until the end of data collection activities to perform quality control to produce finalized versions of collected data for use in scientific analyses. Furthermore, quality control activities typically lag behind the collection of data as scientists need time to review data within the context of environmental conditions (Campbell et al., 2013). Because of these disparities in timeframe between raw and quality controlled data, a quality control editing workflow is needed that supports creation of quality controlled versions of the data while data collection activities continue. Implementing data edits within a script enables regeneration of a processed, quality controlled version of the data from the raw data on demand, and the script effectively becomes both the definition of the quality controlled version of the data and the record of the provenance of data edits. Subsequent edits to the data can be added to the script as needed (e.g., as new data are collected and added to the raw time series).

Fig. 2 shows the data quality control workflow that we developed and implemented for generating and maintaining scripted quality control edits. To begin the workflow, a user first selects a raw time series for editing and plots the data to visually inspect it and determine whether edits are needed. The user then begins an editing session, which creates a memory copy of the dataset on which any edits are performed. If the user is editing a time series for the first time, a new script is created to record the sequence of data edits performed within the editing session. The user then performs edits, each of which is recorded as one or more lines of code in the script. The user can add descriptive comments to the script at any time. At the end of the editing session, the user creates code to delete any unreviewed data from the time series (e.g., in the case where the user does not review all the way to the end of the raw data record during the editing session) and saves the script to disk. The user can then create code to save the edited data series back to the database as a new, quality controlled version of the data. The workflow implements best practices in that all edits are made on a copy of the raw data (Campbell et al., 2013), and the raw data are maintained in the database in their original condition.

When new raw data have been added to a previously edited time series or when a user wants to return and complete quality control editing for a time series they have reviewed only partially, a subsequent editing session begins in much the same way. The user again selects the raw time series for editing and begins an editing session. However, instead of creating a new script, the user opens the script that they saved during their previous editing session. The user then removes from the end of the script any lines that deleted unreviewed data and executes the script on the raw data. The result is a memory copy of the time series that contains quality controlled

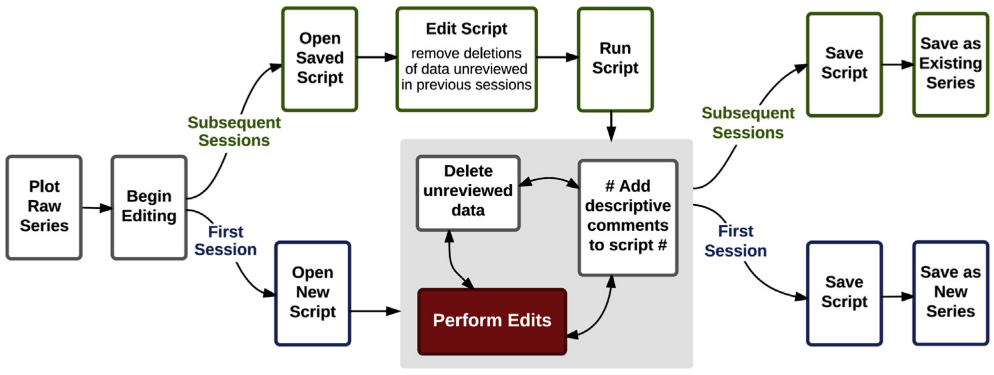


Fig. 2. Workflow for scripting quality control editing of continuous time series data.

data up to the point that the user left off during their previous editing session, with any new or unreviewed raw data appended to end. The user can then continue the same process described above – perform additional edits as needed, add descriptive comments, and then delete any unreviewed data. These steps are appended to the original script. At the end of the current editing session, the user saves the modified script to disk and has a choice of whether to save the resulting edited time series back to the database as a completely new version or to save it back to the database by overwriting the previous quality controlled time series.

This workflow is generic, and does not prescribe a specific coding language or implementation. However, the process can be drastically simplified by selecting a simple language for capturing the scripts, implementing convenient data editing functions that can be used within the scripts, and developing user interface tools for generating the code. In the following sections we describe a software implementation of the above workflow using Python scripting within a cross-platform, open-source, desktop software program.

4. Software implementation

One of the key goals of this research was to investigate how data visualization and quality control editing could be coupled within a single software environment while preserving the provenance of the data editing process. This goal and the challenges described in the sections above inspired the following design requirements for the software we developed:

1. Multi-platform support for Windows, Linux, and Macintosh
2. Multi-database support for Microsoft SQL Server, MySQL, and PostgreSQL
3. Preserving the provenance of all data edits through scripting
4. Coupled graphical and scripting-based editing interface for users
5. Linked visualization and editing capabilities for immediate user feedback
6. Simple software deployment on multiple platforms

For developing the GUI for ODM Tools Python, we chose to adopt an approach similar to that of Williams (2009), who developed a GUI-based software program called Rattle for performing data mining using the statistical computing environment R. Within Rattle, R code is created and executed based on a user's actions within the GUI – e.g., clicking on a button within the GUI generates the R code for the function tied to that button. There are several advantages to this approach when considering the list of requirements above. First, it provides convenient access to the most relevant code functions via buttons in the GUI, but translates each user action into executable code that can be captured in a script. The resulting script can serve as a record of all of the user actions that were performed. Second, it provides a way to support novice users (who will focus on functionality within the GUI) and advanced users (who may go directly to scripting and skip much of the GUI interaction). Finally, implementing scripting within the GUI enables immediate visualization of results via graph and table-based views of the data. We successfully used a similar approach to integrate scripting of data visualization and analysis using R within the HydroDesktop client application for the CUAHSI HIS (Horsburgh and Reeder, 2014).

4.1. Architecture

The ODM Tools Python software was developed in Python as a desktop program with a GUI. It is a client application for a local or remote ODM database implemented within an RDBMS. Time series data are retrieved from an ODM database into local memory for data visualization and editing within ODM Tools and then the results can be saved back to the ODM database. The architecture of the ODM Tools Python software consists of four layers (Fig. 3). In the following sections we describe each of these layers and highlight their main functionality.

4.1.1. Data storage layer

In the CUAHSI HIS, an ODM database implemented within an RDBMS is the underlying data storage mechanism that supports publication of observational data via standardized web services that query data from the ODM database in response to user requests and then return data in a standard XML schema called Water Markup Language (WaterML) (Zaslavsky et al., 2007; Taylor, 2012). Regardless of whether users intend to publish their data using the CUAHSI HIS, ODM provides a convenient and robust structure within which to store, manage, and manipulate sensor data. User access control for ODM databases is handled by the RDBMS within which they are deployed. Using ODM Tools Python requires users to have an appropriate user account to access an ODM database. ODM's existing capability for storing environmental time series data and the query and data manipulation capabilities of the RDBMS made them a suitable platform on which to build the software we envisioned. ODM is described by Horsburgh et al. (2008), and additional documentation including instructions on implementing ODM can be found on the CUAHSI HIS website (http://his.cuahsi.org/odmdatabases.html). At the time of this writing, ODM Tools Python is fully functional with ODM 1.1.1 databases, and we are working on compatibility with a soon-to-be released revision of ODM called ODM2. Sample ODM databases

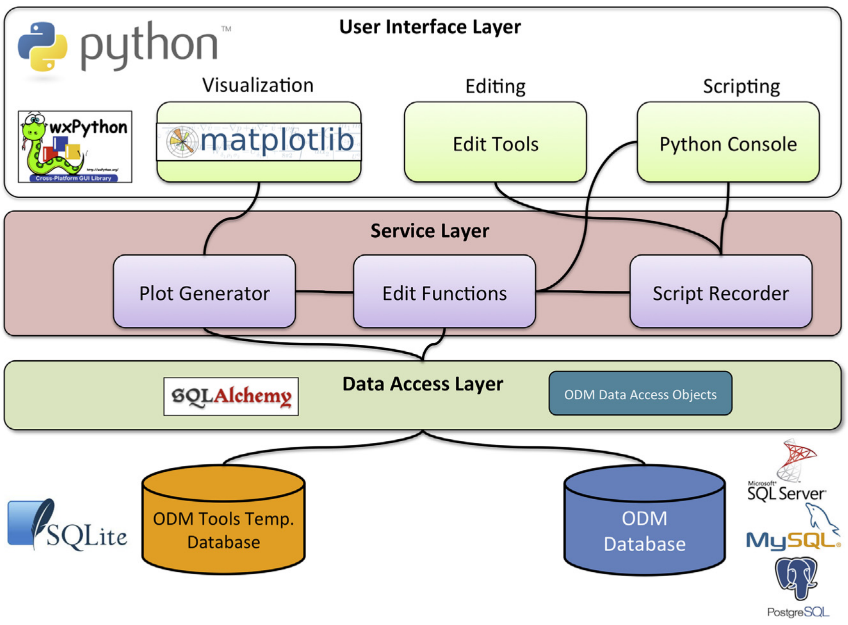


Fig. 3. ODM Tools Python software architecture.

for testing the software are available on the ODM Tools Python GitHub website.

4.1.2. Data access layer

ODM Tools Python uses a SQLAlchemy-based data access layer (http://www.sqlalchemy.org/). This layer serves to abstract data from the ODM database and provides a set of programmable objects that facilitate data management within the ODM Tools software application – rather than repeatedly programming Structured Query Language (SQL) queries directly against the ODM database. This is a significant improvement over earlier versions of ODM Tools, which used SQL queries throughout the code that were difficult to maintain. As a new feature, the data access layer also provides support for connections to ODM databases within multiple RDBMS (currently Microsoft SQL Server, MySQL, and PostgreSQL), whereas older versions worked only with Microsoft SQL Server. Multiple database support is facilitated by SQLAlchemy, which includes SQL dialect implementations for the various RDBMS. An additional benefit of the data access layer is that it insulates the code of the service layer and the user interface layer from the data storage layer. Changes to the data storage layer, for example updating ODM 1.1.1 to ODM2, only affect the data access layer, minimizing the amount of code that has to be refactored as the storage layer evolves.

Additionally, ODM Tools Python uses SQLite as an in-memory database to temporarily store and manipulate time series data objects during data editing sessions. SQLite is a software library that implements a file-based, serverless, transactional SQL database engine (http://www.sqlite.org). The performance of the in-memory SQLite database is dependent upon the amount of available memory on the user's computer. In our testing, loading a time series with approximately 250,000 data values representing 15-min sampling frequency for more than 7 years required approximately 300 MB of available memory.

4.1.3. Service layer

The service layer consists of a set of Python-based services containing the core functionality of the software application. A service is an object that creates and receives data objects defined in the data access layer. The GUI interacts with the data access layer through the service layer. For example, a Plot Generator service provides functions for creating the visualizations within the GUI, the Edit Functions service provides functions for performing data quality control editing on time series data, and the Script Recorder service manages the automated recording of Python function calls during data editing sessions. The Edit Functions service was designed as a Python module that is installed automatically on a user's computer with the ODM Tools Python software deployment. It serves as the underlying code for the data editing tools on the ODM Tools Python editing toolbar, and, because it is independent of the ODM Tools Python GUI, it can be added to and used within any Python script. This means that a data editing Python script created within the ODM Tools software and then saved to a file can be executed independent of the ODM Tools software as long as the Edit Functions Python module is present. Table 2 lists the data editing functions available in the ODM Tools Python Edit Functions Python module and provides examples of how they are used.

4.1.4. User interface layer

The user interface layer provides the GUI within which users can visualize and export data, generate summary statistics, and perform data quality control editing. The GUI was designed and implemented using wxPython (http://www.wxpython.org/), which is a toolkit for Python that provides programmers with components for building interactive GUIs. The matplotlib plotting library (http://matplotlib.org/) was used for creating the data visualizations and provides interactive zooming and panning of the plot. We added customized mouse hover functionality to display individual data values and a plot-based data selection tool to aid in data editing. The user interface layer also includes an integrated Python script editor and console that were implemented using the PyCrust component of wxPython. These libraries are cross-platform, which was necessary for meeting our design objectives and enabled us to reuse the same code on Windows, Linux, and Macintosh platforms.

The ODM Tools Python GUI consists of a ribbon control, a data visualization panel, a time series selection panel, a table viewer panel, a Python script editor, and an integrated Python console. Fig. 4 shows the standard configuration of these components upon

Table 2
ODM Tools Python data editing functions in the data editing service.

Function Name	Description
add_points()	Given a list of data values, inserts values from the list into the data series Example: points = [[(u'8.5', **None**, datetime.datetime(2015, 1, 6, 0, 0), '-7', datetime.datetime(2015, 1, 6, 7, 0), **None**, **None**, u'nc', **None**, **None**, 1, 35, 21, 1, 0)]][0] edit_service.add_points(points)
change_value()	Modifies selected data values by adding, subtracting, multiplying by, or setting equal to a constant value Example: edit_service.change_value(10, '+')
create_method()	Creates a new method in the underlying ODM database Example: edit_service.create_method('Quality controlled data created using ODM Tools Python'), **None**)
create_qualifier()	Creates a new data value qualifier in the underlying ODM database Example: edit_service.create_qualifier('I', 'Quality controlled data created using ODM Tools Python')
create_qcl()	Creates a new data processing/quality control level in the underlying ODM database Example: edit_service.create_qcl('Raw', 'Raw data', 'Raw data that have not undergone quality control.')
create_variable()	Creates a new variable in the underlying ODM database Example: edit_service.create_variable('DO', 'Dissolved Oxygen', 'Not Applicable', 'mg/L', 'Surface Water', 'Field Observation', **False**, 30, 'min', 'Average', 'Water Quality', -9999)
delete_points()	Deletes selected data values from the data series Example: edit_service.delete_points()
drift_correction()	Applies a linear drift correction on the selected data values Example: edit_service.drift_correction(-0.15)
filter_value()	Selects data values falling within a specific data value range Example: edit_service.filter_value(8.25, '>')
filter_date()	Selects data values falling within a specific date range Example: edit_service.filter_date(datetime.datetime(2007, 9, 30, 23, 59, 59), datetime.datetime(2007, 9, 1, 0, 0))
data_gaps()	Selects data values on either side of data gaps of a given time duration Example: edit_service.data_gaps(30.0, 'minute')
flag()	Adds a data qualifying comment to selected values Example: edit_service.flag(1)
interpolate()	Linearly interpolates the selected data values in the data series using the previous and next data values Example: edit_service.interpolate()
reset_filter()	Removes all applied filters for data selection Example: edit_service.reset_filter()
restore()	Discards any changes made and replaces the in-memory database with a new copy of the data series Example: edit_service.restore()
get_filtered_points()	Gets the data values that meet the criteria of the currently applied filter Example: edit_service.get_filtered_points()
get_series_points()	Gets the data values in the currently selected data series Example: edit_service.get_series_points()
filter_from_previous()	Alerts ODM Tools to select data values from the full set or the existing selected set of data values Example: edit_service.filter_from_previous(**True**)
value_change_threshold()	Selects all data values where the change from one value to the next is greater than or less than a threshold Example: edit_service.value_change_threshold(0.01,'>')
save()	Saves the modified data series to the database, overwriting the series from which it was created Example: edit_service.save()
save_as()	Saves the modified data series to the database as a new data series Example: new_variable = series_service.get_variable_by_id(35) new_qcl = series_service.get_qcl_by_id(1) new_method = series_service.get_method_by_id(62) edit_service.save_as(new_variable, new_method, new_qcl)

Table 2 (*continued*)

Function Name	Description

save_exsiting()	Saves the modified data series to the database by overwriting an existing data series in the database
	Example: new_variable = series_service.get_variable_by_id(35)
	new_qcl = series_service.get_qcl_by_id(1)
	new_method = series_service.get_method_by_id(62)
	edit_service.save_existing(new_variable, new_method, new_qcl)

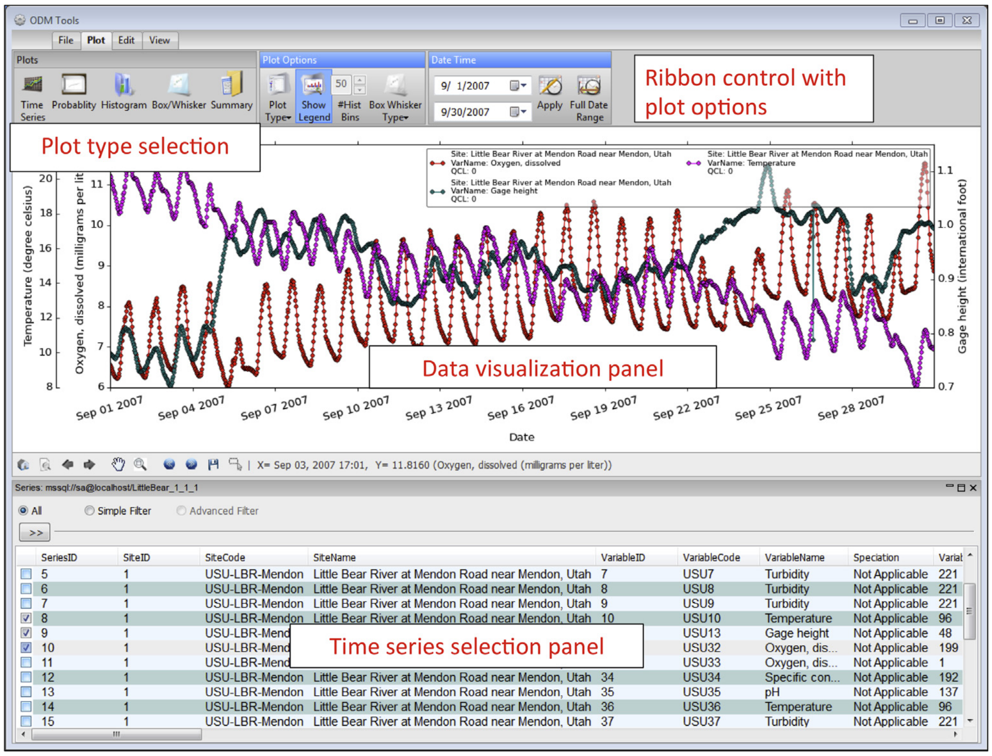
select_points()	Given a list of date/time values, selects the corresponding data value points associated with those dates Example: points = [[datetime.datetime(2007, 9, 14, 8, 30)]][0] edit_service.select_points([], points)

opening the software (not all components are shown). Components can be toggled for visibility from the View menu at the top of the window. Several components of the GUI (e.g., the time series selection panel, table view panel, Python script editor, and Python console) are dockable, which enables users to arrange and configure their window(s) according to their preference.

ODM Tools Python provides several data editing buttons on the main ribbon toolbar, including value filtering (i.e., selection of data values based on user input criteria), insertion or deletion of values, linear interpolation, linear drift correction (a form of variable data correction described by Wagner et al. (2006)), value adjustment (i.e., set equal to, multiply by, or offset by), and value flagging. Flags are an important component of sensor data quality control and serve as indicators of suspect data, failures of quality control checks, or methods used to process or estimate data values (Campbell et al., 2013).

The ODM Tools Python scripting interface automatically records the user's actions as they click these buttons to perform corrections and adjustments on data series in the quality control process. Each button on the data editing toolbar is mapped to a data editing function in the ODM Tools Python Edit Functions service. When a button is clicked, it executes the underlying data editing function on the local memory copy of the data, it fires an equivalent line of code to the Python script editor (Fig. 5), and updates the display of data in the GUI (plot and table) to reflect the change. The sequence of edits automatically recorded in a Python script can be saved as a file, ensuring that the editing steps are traceable and reproducible.

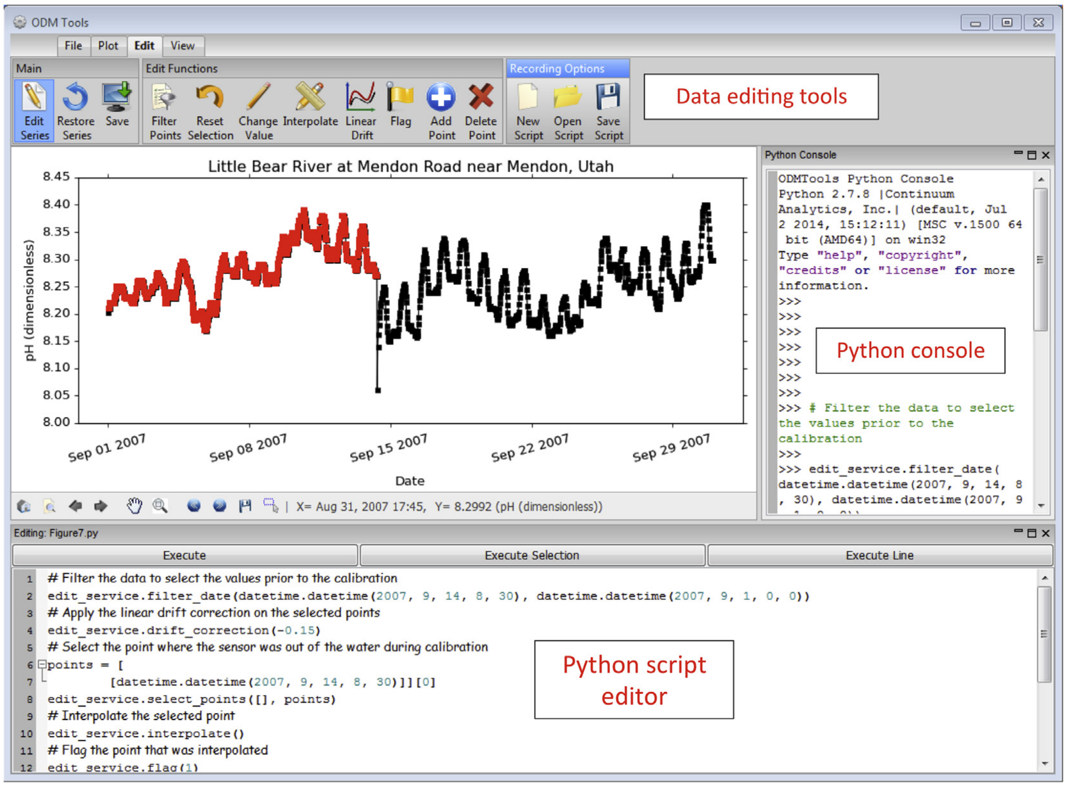
Within ODM Tools Python, all edits are made on a copy of the raw data in order to preserve the original raw data. When a data series is selected for editing, a copy of the data series is created within an in-memory SQLite database. All subsequent edits and modifications are made to the memory copy of the time series. When editing is finished, the user can choose whether to save the edited series. Saving writes the time series stored in memory to the underlying ODM database as a new version, and the memory copy is destroyed. Users can choose whether to create an entirely new version when saving or to overwrite an existing quality controlled version (e.g., in the case of a secondary editing session). These options preserve the metadata associated with the data series and permit the user to specify the version by modifying the method, variable, or quality control level. Using SQLite as the in-memory



X= Sep 03, 2007 17:01, Y= 11.8160 (Oxygen, dissolved (milligrams per liter))

	SeriesID	SiteID	SiteCode	SiteName	VariableID	VariableCode	VariableName	Speciation	Variat
☐	5	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	7	USU7	Turbidity	Not Applicable	221
☐	6	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	8	USU8	Turbidity	Not Applicable	221
☐	7	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	9	USU9	Turbidity	Not Applicable	221
✓	8	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	10	USU10	Temperature	Not Applicable	96
✓	9	1	USU-LBR-Mend			USU13	Gage height	Not Applicable	48
✓	10	1	USU-LBR-Mend			USU32	Oxygen, dis...	Not Applicable	199
☐	11	1	USU-LBR-Mend			USU33	Oxygen, dis...	Not Applicable	1
☐	12	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	34	USU34	Specific con...	Not Applicable	192
☐	13	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	35	USU35	pH	Not Applicable	137
☐	14	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	36	USU36	Temperature	Not Applicable	96
☐	15	1	USU-LBR-Mendon	Little Bear River at Mendon Road near Mendon, Utah	37	USU37	Turbidity	Not Applicable	221

Fig. 4. ODM Tools Python graphical user interface.



```
ODMTools Python Console
Python 2.7.8 |Continuum
Analytics, Inc.| (default, Jul
2 2014, 15:12:11) [MSC v.1500 64
 bit (AMD64)] on win32
Type "help", "copyright",
"credits" or "license" for more
information.
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> # Filter the data to select
the values prior to the
calibration
>>>
>>> edit_service.filter_date(
datetime.datetime(2007, 9, 14, 8
, 30), datetime.datetime(2007, 9
```

```
# Filter the data to select the values prior to the calibration
edit_service.filter_date(datetime.datetime(2007, 9, 14, 8, 30), datetime.datetime(2007, 9, 1, 0, 0))
# Apply the linear drift correction on the selected points
edit_service.drift_correction(-0.15)
# Select the point where the sensor was out of the water during calibration
points = [
        [datetime.datetime(2007, 9, 14, 8, 30)]][0]
edit_service.select_points([], points)
# Interpolate the selected point
edit_service.interpolate()
# Flag the point that was interpolated
edit_service.flag(1)
```

Fig. 5. ODM Tools Python graphical user interface in editing mode with integrated Python script editor and console.

storage for data series being edited provides a robust and convenient structure that supports querying and filtering in a memory object that does not have to be persisted on disk.

The time series selection panel at the bottom of the window enables users to select which data series are shown in the data visualizations or which series is selected for editing. Simple filters can be applied to the list of data series stored within the attached database to identify a subset of data series that meet particular criteria. This is especially useful where the underlying ODM database contains many data series. Users can right click on a data series listing in the time series selection panel and choose an option in the context menu to export the data or the metadata, providing a simple method for exporting data from the underlying database to a file on the user's computer.

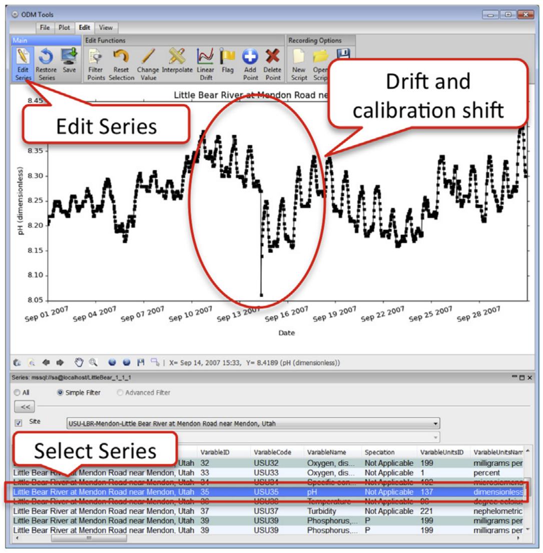
Enhanced visualizations within ODM Tools Python include the ability to plot multiple time series simultaneously (Fig. 4). This is a significant improvement over previous versions of ODM Tools and is particularly useful in data quality control editing. Anomalous values in the time series being edited can easily be compared to other time series at the same site or a time series for the same variable from another nearby site to determine whether the anomalous values should be removed, interpolated, left alone, etc. Additional plot types supported by ODM Tools Python include probability plots, histograms, and box-and-whisker plots. Users can switch between plot types by selecting the plot type on the ribbon control.

5. Software deployment

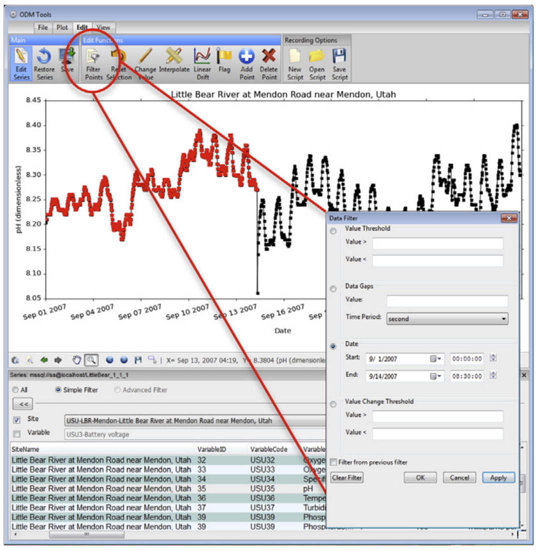
ODM Tools Python users currently have multiple choices for running the software. The first option is to download the source code for a particular release of ODM Tools Python from the GitHub repository and then run ODM Tools directly from the source code. This requires that a user first configure a Python environment on his/her computer that has all of the requisite Python modules and components already installed. Documentation on the ODM Tools Python source code repository website describes the Python modules that are required for running ODM Tools Python from source code. Prior to posting the source code for a release, it is tested on Windows, Mac, and Linux platforms to ensure that it will work for users on these platforms. This approach may be preferable for experienced Python users who want flexibility in how Python and the various components are installed. Potential developers who want to modify the source code of ODM Tools can either download the source code for a particular release or they can clone the GitHub repository to get the latest development code. Running from source code is currently the only option available for Linux users.

Knowing that there are many within the scientific community that might want to use the ODM Tools Python software but may lack the software expertise for setting up a development environment for running from source code (or that have an established Python environment on their computer that they do not want to change), we designed two additional ODM Tools Python software deployments: 1) a no-install zip file; and 2) executable software installers. Both of these use a Python virtual environment within which the ODM Tools Python software runs. The virtual environment is an isolated working copy of Python that is deployed with the software and contains the correct version of all of the requisite Python modules. It does not require installing anything in the user's existing Python environments and, therefore, will not conflict with any existing Python environments on the user's computer.

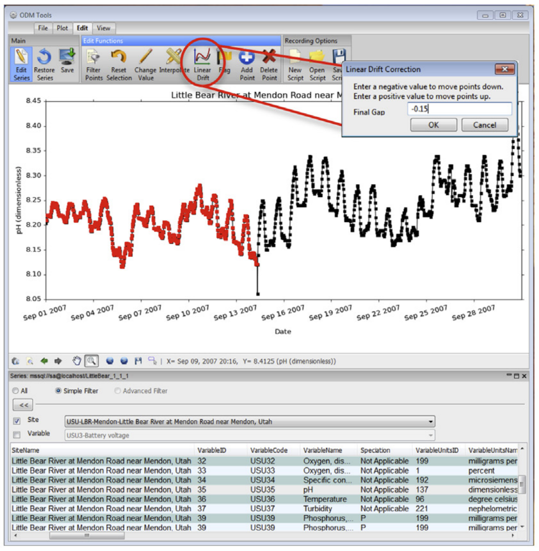
For the no-install option, a zip file containing the Python virtual environment and the latest source code for ODM Tools is downloaded and extracted on a user's computer. This enables users to run ODM Tools from source code without an install, but within the


Edit Series
Drift and calibration shift
Select Series
Little Bear River at Mendon Road near Mendon, Utah

(a)



(b)



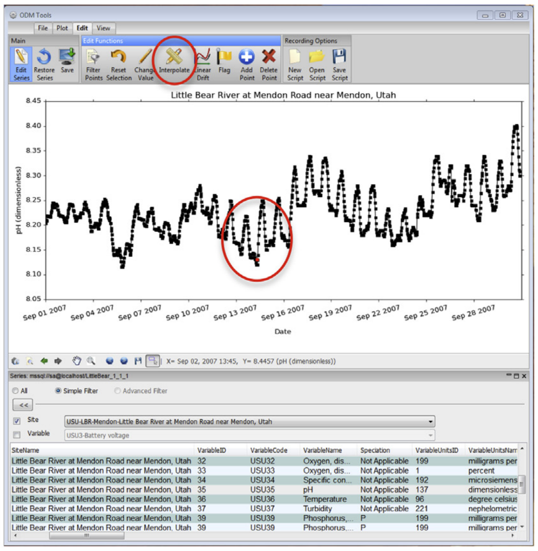
Linear Drift Correction

Enter a negative value to move points down.
Enter a positive value to move points up.

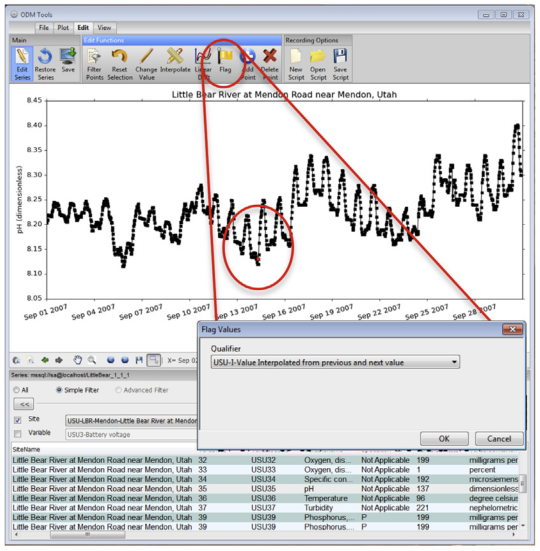
Final Gap: -0.15

OK Cancel

(c)



(d)

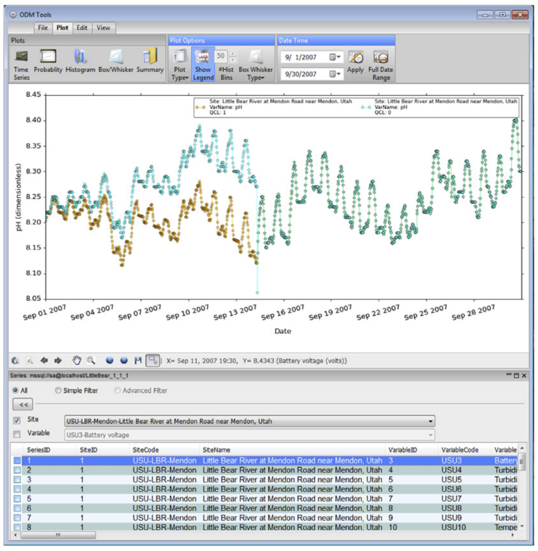


Flag Values

Qualifier: USU-I-Value Interpolated from previous and next value

OK Cancel

(e)



(f)

Fig. 6. Example of ODM Tools Python data editing workflow.

encapsulated Python environment delivered within the zip file. The no-install option requires launching the main ODM Tools Python form from the Python console. This approach is preferable for users who may not have administrative access to run software installers on their computers. This option works on both Windows and Mac platforms and does not have any pre-requisites, as the Python environment delivered within the zip file contains Python and all of the required software libraries. The last option is to use one of the executable installers provided via an ODM Tools Python software release. Users can currently download executable installers for both Windows and Mac OS X from the ODM Tools GitHub website, and, like the no-install option, there are no prerequisites.

6. Example application

An example is presented in this section to illustrate the use of ODM Tools Python for performing quality control edits and to highlight the benefits of the ODM Tools Python scripting functionality. Fig. 6 illustrates the ODM Tools Python data editing workflow for a time series of pH data measured in the Little Bear River of northern Utah, USA. These data were recorded by an *in situ* pH sensor every 30 min. The time period between sensor cleaning and calibration checks at this site is approximately 2 weeks, during which some sensor fouling and instrument drift occurs. In panel (a) of Fig. 6, the pH data series was plotted by selecting it in the series selection panel and then placed in editing mode using the “Edit Series” button on the ribbon toolbar. The calibration date and resulting shift in the data can be seen near the middle of the plot. There is also a single anomalous value that resulted from the sensor being out of the water during the calibration process.

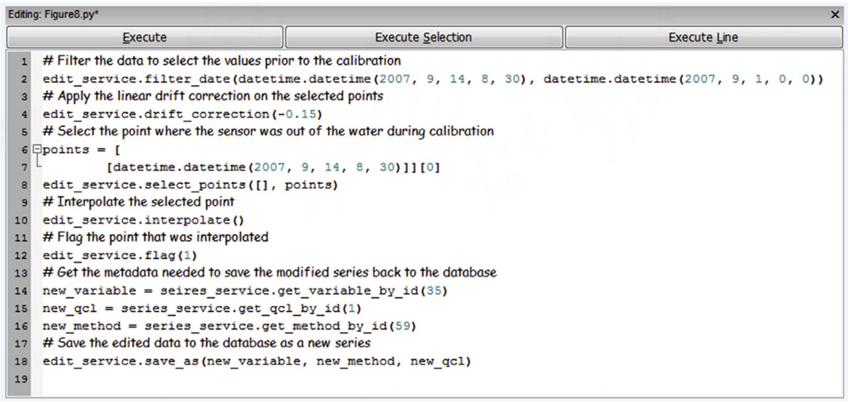
In panel (b), the pH data values prior to the sensor calibration were selected using a date filter in the “Filter Points” tool (ODM Tools highlights selected data values in red). Points can also be selected using other filters in the Filter Points tool (e.g., value threshold, value change threshold, data gaps, etc.), by drawing a selection polygon around them on the plot using a Lasso tool, or by clicking on them in the table view. In panel (c), a linear drift correction was applied to remove the sensor drift between the beginning of the data series and the calibration date. For this example, a correction value of −0.15 pH units was applied based on calibration check values measured in the field and recorded in the field notes. In panel (d), the single remaining anomalous data value that resulted when the sensor was out of the water for calibration was selected and interpolated using the “Interpolate” tool on the toolbar. Panel (e) shows the dialog for applying data qualifying comments accessed by clicking on the “Flag” button on the editing toolbar. ODM Tools applies the selected qualifying comment to any data values that are selected; in this case, the single data value that was interpolated was flagged with a qualifying comment indicating that it was interpolated. Finally, panel (f) shows both the original and edited pH data series after the edited data series was saved to the database as a new version.

The Python script generated by this editing workflow is shown in Fig. 7. Each button click and/or action in the graphical user interface using the editing tools on the toolbar was recorded in the Python script editor as one or more lines of code that call a function in the ODM Tools Python edit service. The script in Fig. 7 is shown within the ODM Tools Python script editor, which includes usability features such as code folding, syntax highlighting, and execution buttons to make it easier for users to work with the code. Within the script editor, users can add comments to the code to indicate why they added particular edits or performed particular actions. The final Python script can then be saved to the user's hard drive as a record of the edits that were performed and for later execution, modification, or review by other scientists or data analysts. This may help address inconsistencies and human bias that can occur with manual data quality control (Campbell et al., 2013).

7. Summary and future work

We have developed a new version of ODM Tools in Python that provides multiple platform support, multiple database support, and support for automated scripting of quality control edits performed on data series – each of which were major limitations in the original versions of ODM Tools. Overcoming the platform and database limitations provides flexibility for users, regardless of their chosen platform or RDBMS. The automated scripting capabilities provide an effective method for tracking the quality control edits performed during an editing session, preserving the provenance of how quality controlled datasets were created, and enabling subsequent editing sessions and regeneration of the quality controlled data at any time.

Several new features were included in modernizing the ODM Tools GUI. The integrated ribbon control provides flexibility in implementing toolbar buttons and menus and is more consistent with common desktop software with which users will be familiar. Dockable window components enable users to configure their window(s) to suit their needs. The ability to plot multiple data series simultaneously with various plot types is particularly useful in the quality control editing process as users can examine an edited series versus the raw data to make sure data edits have been applied correctly, or they can examine a series that is being edited alongside other time series collected at the same or nearby sites to investigate anomalous data and determine whether edits need to



```
# Filter the data to select the values prior to the calibration
edit_service.filter_date(datetime.datetime(2007, 9, 14, 8, 30), datetime.datetime(2007, 9, 1, 0, 0))
# Apply the linear drift correction on the selected points
edit_service.drift_correction(-0.15)
# Select the point where the sensor was out of the water during calibration
points = [
        [datetime.datetime(2007, 9, 14, 8, 30)]][0]
edit_service.select_points([], points)
# Interpolate the selected point
edit_service.interpolate()
# Flag the point that was interpolated
edit_service.flag(1)
# Get the metadata needed to save the modified series back to the database
new_variable = seires_service.get_variable_by_id(35)
new_qcl = series_service.get_qcl_by_id(1)
new_method = series_service.get_method_by_id(59)
# Save the edited data to the database as a new series
edit_service.save_as(new_variable, new_method, new_qcl)
```

Fig. 7. Listing of the Python code generated by the data editing steps in the example. The code is shown in the ODM Tools Python integrated Python script editor.

be made. The code of the GUI also serves as an example of how interactive user interfaces can be created using Python, which can be challenging in practice.

The workflow we developed for scripting time series data quality control editing is a simple but effective procedure for creating and maintaining scripts that encode data edits, and it is facilitated by the redesigned ODM Tools software. ODM Tools combines editing functions exposed via buttons on the main GUI toolbar with automatically generated Python code. The strength of this approach is that it exposes the ability to perform data editing actions through the GUI, but translates any actions within the GUI into executable Python code. The resulting script serves as a record of the edits and can be re-executed at any time to generate the same result. Users can create new quality control editing scripts based on their visual editing through the ODM Tools GUI or they can open and execute the scripts resulting from previous editing sessions. This approach supports both novice and more experienced users.

Last, we believe that the deployment approach we chose for ODM Tools Python may be instructive for how Python-based software can be more easily distributed without requiring users to set up their own Python environment with all of the prerequisite components. Deploying ODM Tools Python with its own Python virtual environment enables it to run on computers that do not have Python installed, providing a quick and easy way to get up and running. For users that already have one or more Python environments on their computer, it isolates the Python components used with ODM Tools Python and ensures that the required versions of Python and modules run regardless of whether other versions exist in other Python environments. This avoids version conflicts that could lead to errors in ODM Tools or other Python programs the user has created. Finally, more advanced users or potential developers can access the source code from the GitHub repository and can run from source code within a Python environment that they have set up. These multiple options provide flexibility in reaching a broad range of potential users. Indeed, we anticipate that ODM Tools Python will be useful for data managers needing procedures and software tools for data post processing and quality control.

Future work will be aimed at adding additional editing tools to the data editing service and the data editing toolbar. These may include two point variable data corrections, smoothing algorithms, and data aggregation/derivation algorithms. Visualization of data qualifying comments with plotted data is another area we are seeking to improve. We are also working to better support the ability for users to define and use custom, user-defined functions within the Python scripting environment for data quality control. We have recently completed a round of modifications to improve the performance of the software, and we are currently in the process of making ODM Tools compatible with ODM2, which is the next version of ODM. ODM2 adds capability for integrating both sensor and sample-based observational data within a single database and enhances the ability to describe and manage both types of environmental observations. This capability will make ODM2 and ODM Tools Python more versatile tools for data managers working with multiple types of environmental observations data.

Acknowledgments

This work was supported by the National Science Foundation under EPSCoR grant IIA 1208732 awarded to Utah State University, as part of the State of Utah EPSCoR Research Infrastructure Improvement Award. Additional support was provided by National Science Foundation grants EAR 1224638 and ACI 1339834. Support for previous versions of the ODM Tools software was provided by National Science Foundation grant EAR 0622374. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We gratefully acknowledge Chris Cox, Dave Eiriksson, and Joe Crawford who provided valuable testing and feedback on the ODM Tools Python software.

References

Borer, E.T., Seabloom, E.W., Jones, M.B., Schildhauer, M., 2009. Some simple guidelines for effective data management. ESA Bull. 90 (2), 205–214. http://dx.doi.org/10.1890/0012-9623-90.2.205.

Campbell, J.L., Rustad, L.E., Porter, J.H., Taylor, J.R., Ethan, W., Shanley, J.B., Gries, C., Henshaw, D.L., Martin, M.E., Wade, M., Boose, E.R., Dereszynski, E.W., 2013. Quantity is nothing without quality. Bioscience 63 (7), 574–585. http://dx.doi.org/10.1525/bio.2013.63.7.10.

Dereszynski, E.W., Dietterich, T.G., 2012. Probabilistic models for anomaly detection in remote sensor data streams. In: Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence. http://arXiv:1206.5250.

Fiebrich, C.A., Morgan, C.R., McCombs, A.G., Hall Jr., P.K., McPherson, R.A., 2010. Quality assurance procedures for mesoscale meteorological data. J. Atmos. Ocean. Technol. 27 (10), 1565–1582. http://dx.doi.org/10.1175/2010JTECHA1433.1.

Hill, D.J., Minsker, B.S., Amir, E., 2009. Real-time Bayesian anomaly detection in streaming environmental data. Water Resour. Res. 45 (4) http://dx.doi.org/10.1029/2008WR006956.

Horsburgh, J.S., Reeder, S., 2014. Data visualization and analysis within a hydrologic information system: integrating with the R statistical computing environment. Environ. Model. Softw. 52, 51–61. http://dx.doi.org/10.1016/j.envsoft.2013.10.016.

Horsburgh, J.S., Tarboton, D.G., Maidment, D.R., Zaslavsky, I., 2008. A relational model for environmental and water resources data. Water Resour. Res. 44, W05406. http://dx.doi.org/10.1029/2007WR006392.

Horsburgh, J.S., Tarboton, D.G., Maidment, D.R., Zaslavsky, I., 2011. Components of an environmental observatory information system. Comput. Geosci. 37, 207–218. http://dx.doi.org/10.1016/j.cageo.2010.07.003.

Horsburgh, J.S., Tarboton, D.G., Schreuders, K.A.T., Maidment, D.R., Zaslavsky, I., Valentine, D., 2010. Hydroserver: a platform for publishing space-time hydrologic datasets. In: Proceedings 2010 American Water Resources Association Spring Specialty Conference Geographic Information Systems (GIS) and Water Resources VI, Orlando, Florida. American Water Resources Association, Middleburg, Virginia, TPS, pp. 10–11.

Horsburgh, J.S., Tarboton, D.G., Piasecki, M., Maidment, D.R., Zaslavsky, I., Valentine, D., Whitenack, T., 2009. An integrated system for publishing environmental observations data. Environ. Model. Softw. 24 (8), 879e888. http://dx.doi.org/10.1016/j.envsoft.2009.01.002.

Lerner, B., Boose, E., Osterweil, L., Ellison, A., Clarke, L., 2011. Provenance and quality control in sensor networks. In: Jones, M., Gries, C. (Eds.), Proceedings of the Environmental Information Management Conference, September 2011, pp. 98–103. http://dx.doi.org/10.5060/D2NC5Z4X.

Mason, S.J.K., Cleveland, S.B., Llovet, P., Izurieta, C., Poole, G.C., 2014. A centralized tool for managing, archiving, and serving point-in-time data in ecological research laboratories. Environ. Model. Softw. 51, 59–69. http://dx.doi.org/10.1016/j.envsoft.2013.09.008.

Millman, K.J., Aivazis, M., 2011. Python for Scientists and Engineers. Comput. Sci. Eng. 13 (2), 9–12. http://dx.doi.org/10.1109/MCSE.2011.36.

Moatar, F., Miquel, J., Poirel, A., 2001. A quality-control method for physical and chemical monitoring data. Application to dissolved oxygen levels in the river Loire (France). J. Hydrol. 252 (1–4), 25–36. http://dx.doi.org/10.1016/S0022-1694(01)00439-5.

Mourad, M., Bertrand-Krajewski, J.L., 2002. A method for automatic validation of long time series of data in urban hydrology. Water Sci. Technol. 45 (4–5), 263–270.

Perez, F., Granger, B.E., Hunter, J.D., 2011. Python: an ecosystem for scientific computing. Comput. Sci. Eng. 13 (2), 13–21. http://dx.doi.org/10.1109/MCSE.2010.119.

Sheldon Jr., W.M., 2008. Dynamic, rule-based quality control framework for real-time sensor data. In: Gries, C., Jones, M.B. (Eds.), Proceedings of the Environmental Information Management Conference 2008: Sensor Networks, September 2008, pp. 145–150. http://gce-lter.marsci.uga.edu/public/files/pubs/wsheldon_dynamic_qc_eimc2008_final.pdf.

Tarboton, D.G., Horsburgh, J.S., Maidment, D.R., Whiteaker, T., Zaslavsky, I., Piasecki, M., Goodall, J., Valentine, D., Whitenack, T., 2009. Development of a community hydrologic information system. In: Anderssen, R.S., Braddock, R.D., Newham, L.T.H. (Eds.), 18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation, Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation, July 2009, ISBN 978-0-9758400-7-8, pp. 988–994.

Taylor, J.R., Loescher, H.L., 2013. Automated quality control methods for sensor data: a novel observatory approach. Biogeosciences 10, 4957–4971. http://dx.doi.org/10.5194/bg-10-4957-2013.

Taylor, P. (Ed.), 2012. OGC WaterML 2.0: Part 1 – Timeseries, OGC Candidate Standard OGC 10-126r2, Version 2.0.0. https://portal.opengeospatial.org/files/?artifact_id=48531 (accessed 27.12.14.).

Wagner, R.J., Boulger, R.W., Oblinger, C.J., Smith, B.A., 2006. Guidelines and Standard Procedures for Continuous Water-quality Monitors: Station Operation, Record Computation, and Data Reporting. U.S. Geological Survey Techniques and Methods 1-D3, 51 pp. + 8 attachments. http://pubs.water.usgs.gov/tm1d3.

White, D.L., Sharp, J.L., Eidson, G., Parab, S., Ali, F., Esswein, S., 2010. Real-time quality control (qc) processing, notification, and visualization services, supporting data management of the Intelligent River©. In: Proceedings of the 2010 South Carolina Water Resources Conference, October 2010.

Williams, G.J., 2009. Rattle: a data mining GUI for R. R J. 1 (2), 45–55. http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Zaslavsky, I., Valentine, D., Whiteaker, T. (Eds.), 2007. CUAHSI WaterML. OGC Discussion Paper OGC 07-041r1. Version 0.3.0. http://portal.opengeospatial.org/files/?artifact_id=21743 (accessed 27.12.14.).