EPA United States Environmental Protection Agency

EPA 600/R-08/040B | September 2012 | www.epa.gov/ord

User's Manual for CANARY

Version 4.3.2

D. B. Hart and S. A. McKenna
Geoscience Research and Applications
National Security Applications Department
Sandia National Laboratories
Albuquerque, NM 87185-0751

CANARY Software
D. B. Hart, K. A. Klise, E. D. Vugrin, S. A. McKenna, and M. P. Wilson
Sandia National Laboratories
Albuquerque, NM 87185-0751

Project Officers
R. Murray and T. Haxton
National Homeland Security Research Center
Office of Research and Development
U.S Environmental Protection Agency
Cincinnati, OH 45268

Preamble

The U.S. Environmental Protection Agency (EPA) through its Office of Research and Development funded and collaborated in the research described here under an Inter-Agency Agreement with the Department of Energy's Sandia National Laboratories (IA # DW8992192801). This document has been subjected to the Agency's review, and has been approved for publication as an EPA document. EPA does not endorse the purchase or sale of any commercial products or services.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Questions concerning this document or its application should be addressed to:

Regan Murray
USEPA/NHSRC (NG 16)
26 W Martin Luther King Drive
Cincinnati OH 45268
(513) 569-7031
Murray.Regan@epa.gov

Forward

In July of 1970, the White House and Congress worked together to establish the United States Environmental Protection Agency (EPA) in response to the growing public demand for cleaner water, air, and land. The Agency was assigned the daunting task of repairing the damage already done to the natural environment and establishing new criteria to guide Americans in making a cleaner environment a reality. Since 1970, EPA has worked with federal, state, tribal, and local partners to advance its mission to protect human health and the environment.

EPA leads the nation's environmental science, research, education and assessment efforts. With more than 17,000 employees across the country, EPA works to research, develop and enforce regulations that implement environmental laws enacted by Congress. In recent years, between 40 and 50 percent of EPA's enacted budgets have provided direct support through grants to State environmental programs. At laboratories located throughout the nation, the Agency works to assess environmental conditions and to identify, understand, and solve current and future environmental problems. The Agency works through its headquarters and regional offices with over 10,000 industries, businesses, nonprofit organizations, and state and local governments, on over 40 voluntary pollution prevention programs and energy conservation efforts.

Under existing laws and recent Homeland Security Presidential Directives, EPA has been called upon to play a vital role in helping to secure the nation against foreign and domestic enemies. The National Homeland Security Research Center (NHSRC) was formed in 2002 to conduct research in support of EPA's role in homeland security. NHSRC research efforts focus on five areas: water infrastructure protection, threat and consequence assessment, decontamination and consequence management, response capability enhancement, and homeland security technology testing and evaluation. EPA is the lead federal agency for drinking water and wastewater systems and the NHSRC is working to reduce system vulnerabilities, prevent and prepare for terrorist attacks, minimize public health impacts and infrastructure damage, and enhance recovery efforts.

This User's Manual for the CANARY software package is published and made available by EPA's Office of Research and Development to assist the water community by improving the security of our Nation's drinking water.

Jonathan Herrmann, Director

National Homeland Security Research Center
Office of Research and Development
U. S. Environmental Protection Agency

Acknowledgements

The National Homeland Security Research Center would like to acknowledge the following organizations and individuals for their support in the development of the CANARY User's Manual and/or in the development and testing of the CANARY Software.

Office of Research and Development – National Homeland Security Research Center
- Jennifer Hagar
- John Hall
- Terra Haxton
- Robert Janke
- Regan Murray

Office of Water – Water Security Division
- Steve Allgeier
- Katie Umberg

Sandia National Laboratories
- Robert Aumer
- Samantha Cafferky
- Laura Cutler
- David Hart
- William Hart
- Sean Hollister
- Katherine Klise
- Shawn Martin
- Sean McKenna
- Marguerite Sorensen
- Eric Vugrin
- Mark Wilson
- Mark Wunsch

American Water Works Association Utility Users Group
- Kevin Morley (AWWA)
- David Hartman (Greater Cincinnati Water Works)
- Yeongho Lee (Greater Cincinnati Water Works)
- Dan Quintanar (Tucson Water)
- Zia Bukhari (New Jersey American Water)

Shaw Group
- Srinivas Panguluri (Working at EPA T&E Facility)

Contents

PREAMBLE 2
FORWARD 3
ACKNOWLEDGEMENTS 4
CONTENTS 5
TABLE OF FIGURES 6
LIST OF TABLES 7

1 INTRODUCTION 9

2 DESIGN 11

2.1 OFFLINE MODE 12
2.2 ONLINE MODE 14
2.3 TERMINOLOGY 15
2.4 WATER QUALITY ESTIMATION AND RESIDUAL CLASSIFICATION 18
2.5 WATER QUALITY EVENT DETERMINATION 22

3 CANARY INPUTS 26

3.1 CSV TYPE INPUT DATA SOURCES 26
3.2 DATABASE INPUT SOURCES (JDBC, DB) 27
3.3 CUSTOM INPUTS (EDDIES AND XML) 29

4 CANARY OUTPUTS 30

4.1 OUTPUT ALWAYS PROVIDED 30
4.2 USER-REQUESTED OUTPUTS 32

5 CONFIGURATION FILE DETAILS 34

5.1 CONTROL SECTION 34
5.2 TIMING OPTIONS SECTION 36
5.3 INPUTS AND OUTPUTS – DATA SOURCES SECTION 38
5.4 SIGNALS DEFINITION SECTION 49
5.5 ALGORITHMS DEFINITION SECTION 59
5.6 MONITORING STATIONS SECTION 63
5.7 COMPLETE CONFIGURATION FILES 66

6 TRAINING CANARY AND CHOOSING PARAMETER SETTINGS 67

7 REFERENCES 68

APPENDIX A - YAML TIPS 70

Table of Figures

Figure 1. CANARY's interaction with a network of online sensor stations and a SCADA system..... 12
Figure 2. Flowchart for CANARY's offline operation..... 14
Figure 3. Flowchart for CANARY's online operation..... 15
Figure 4. Graphical representation of an outlier, an event, and a baseline change..... 18
Figure 5. Schematic diagram of set-point algorithms SPPB and SPPE. 21
Figure 6. Example screen from event selection portion of pattern library creation. 25
Figure 7. Example screen from the pattern library editor..... 25

List of Tables

Table 1. Acceptable values and related descriptions for the *input type* configuration parameter. 26
Table 2. A sample showing the header and five data rows from a CSV data file. 27
Table 3. Example table from a database using the *input format:* row-based. 28
Table 4. Acceptable values and related descriptions for the *output type* configuration parameter. 30
Table 5. Basic control setup for a historical run with no extra driver files needed. 35
Table 6. A control configuration for a REALTIME run that requires two driver files. 36
Table 7. A *timing options* section that is configured for an offline (BATCH) run. 38
Table 8. A *timing options* section that is configured for an online (REALTIME) run. 38
Table 9. Configuration for *data sources* of *type:* CSV. 43
Table 10. Configuration for *data sources* of *type:* JDBC with *input format:* default and *output format:* default. 43
Table 11. Configuration for *data sources* of *type:* JDBC with *input format:* row based and *output format:* extended. 45
Table 12. Configuration for *data sources* of *type:* JDBC with *input format:* row based with customized field names and *output format:* default with customized field names. 45
Table 13. Complete configuration for a three-database system, reading from two different input databases and outputting to a third. 47
Table 14. Configuration for a single WQ type signal; includes the *signals* key. 53
Table 15. Configuration for a single WQ type signal with set points and a valid range. 54
Table 16. Configuration for a single ALM type signal associated the WQ signal in example S1. 54
Table 17. Configuration for several OP type signals, including a composite signal made from signals in both examples S4 and S1. 55
Table 18. Configuration for two CAL signals, a direct hardware signal and a composite signal. 58
Table 19. Configuration code for the LPCF algorithm with the BED enabled. 60
Table 20. Configuration code for a set-point algorithm. 61
Table 21. Configuration code for a consensus algorithm. 61
Table 22. Configuration code of an algorithm with a pattern matching cluster library. .. 62
Table 23. Configuration code for an example external Java based algorithm. 62
Table 24. Configuration code for a simple monitoring station entry. 65

Table 25. Configuration code for a multi-input and output monitoring station with complex algorithms. 66
Table 26. Examples of the three YAML data types. 70
Table 27. A full YAML document with annotations. 71
Table 28. YAML that will not work! 72

1 Introduction

Contamination warning systems (CWSs) have been proposed as a promising approach for reducing the risks associated with contamination of drinking water. In order to maximize detection likelihoods, CWSs can incorporate multiple detection technologies, such as online continuous water quality monitoring, public health surveillance, physical security monitoring, and customer complaint surveillance. The goal of a CWS is to detect contamination incidents in drinking water systems rapidly enough to allow for the effective mitigation of adverse public health and economic impacts. Since 2006, the U.S. Environmental Protection Agency (EPA) has been deploying and evaluating CWSs at a series of drinking water utilities.

With current technology, the online monitoring component of a CWS is based on available water quality sensors that measure, for example, free chlorine, total organic carbon, electrical conductivity, oxidation-reduction potential, and pH. Recent research has shown that many contaminants of concern can cause detectable changes in these water quality parameters (Hall et al., 2007; US EPA, 2005). However, these parameters are known to vary considerably over time in water distribution systems due to normal changes in the operations of tanks, pumps, and valves, and daily and seasonal changes in the source and finished water quality, as well as fluctuations in demands.

Data analysis tools are needed to distinguish between normal variations in water quality and changes in water quality triggered by the presence of contaminants. Often referred to as event detection systems (EDSs), such data analysis tools can read in Supervisory Control and Data Acquisition (SCADA) data (water quality signals, operations data), perform an analysis in near real-time, and then return the probability of a water quality event occurring at the current time step. A water quality event is defined as the period in time within which water of unexpected characteristics occurs. The CANARY event detection software described here provides a continuous measure of the probability of an event.

The goal of CANARY is to take standard water quality data and use statistical and mathematical algorithms to identify the onset of periods of anomalous water quality, while at the same time, limiting the number of false alarms that occur. The user sets the working definition of “anomalous” by selecting the configuration parameters. These parameters could be different from one utility to the next and might even need to vary across monitoring stations within a single utility. CANARY can be configured to receive data from a SCADA database, and return alarms to the SCADA system. In addition, it can analyze historical data to assist in the selection of the configuration parameters in order to provide the desired balance between event detection sensitivity and false alarm rates.

CANARY is designed to be extensible, allowing outside researchers to develop new algorithms that can be incorporated into CANARY. Several change detection algorithms are included within CANARY: a linear filter, a multivariate nearest-neighbor algorithm, and a set-point proximity algorithm. These algorithms identify a background water

quality signature for each water quality sensor and compare each new water quality measurement to that background to determine if the new measurement is an outlier (anomalous) or not.

The definition of the water quality background is updated continuously as new data become available. A binomial event discriminator (BED) examines multiple outliers within a prescribed time window to determine the onset of either an anomalous event or a change in the water quality baseline. Information on the development of the three event detection algorithms can be found in: McKenna, Klise, and Wilson (2006) (time series increments and linear filter), Klise and McKenna (2006a; 2006b) (multivariate nearest neighbor), and Hart et al. (2007) and McKenna et al. (2007) (binomial event discriminator).

CANARY also has a water quality pattern matching capability. Adjustments in utility operations often create changes in the water quality within the distribution network that can produce false alarms in EDS tools. In many cases, these water quality changes occur on a daily basis, although they do not occur at exactly the same time or produce exactly the same pattern from one day to the next. Pattern matching in CANARY analyzes historical data from a single monitoring station and identifies recurring patterns. CANARY allows the user to select the water quality changes that should be included in a pattern library. The stored patterns incorporate all water quality monitoring signals. A clustering algorithm is used to represent the various changes in water quality with a reduced parameter set. The changes in the various water quality signals are conceptualized as a series of points linked in time that define a trajectory. A trajectory clustering approach (Gaffney, 2004) employing fuzzy c-means clustering (Dunn, 1973; Bezdek, 1981) is used within CANARY. Pattern matching can significantly reduce the number of false alarms.

To be useful to water utilities, event detection systems must have low false positive rates, high likelihood of detecting true events, and be sufficiently reliable. The CANARY event detection software is being released to the public in order to promote widespread development and testing of event detection algorithms among researchers, consultants, vendors, and utilities. In this way, water utilities will be provided with high performing tools that can be trusted as part of their daily operations.

The rest of this manual is organized as follows:

- Design – the design section discusses modes of operation and the different algorithms that can be used in event detection.
- Inputs and Outputs – these sections describe how data should be formatted for use by CANARY, and the type of information provided in return.
- Configuration – this section describes how to configure CANARY for a specific need.

Webinar tutorials are available online at the CANARY software download site, which can be accessed from: http://www.epa.gov/nhsrc/.

2 Design

CANARY analyzes one time step of water quality data at a time. The data is read from a file or through an online connection to a SCADA system (Figure 1). CANARY determines if the measured data at the current time step is consistent with the predicted data. This determination is made by classification of the residual – the difference between the measured and predicted data values – as either normal (background) or anomalous (outlier). The predicted values are derived from a choice of one or more algorithms that use linear combinations of previously measured data values to estimate the next value. By examining the results of the residual classification over multiple consecutive time steps, the probability of a water quality event occurring at the current time step is calculated. If the probability of an event exceeds a user-defined probability threshold, CANARY declares that an event is occurring. Changes in water quality that occur with some regularity, such as those caused by daily adjustments in utility operations, can be recorded and stored in a pattern library for future reference.

CANARY has been designed for two different modes of operation: online and offline. The offline mode uses historical data to analyze algorithm performance for different parameter settings. Online mode uses real-time data, typically through connection to a SCADA database, to perform online analysis of water quality. While the algorithms operate identically in both cases, CANARY's control structure is different. Diagrams showing how CANARY operates are shown in Figure 2 (offline mode) and Figure 3 (online mode).

Parameters, controls, and input and output options for CANARY are specified in a configuration or "config" file. The config file is written in YAML format, a type of text file where elements are nested by indentation. A detailed description of the configuration file is presented in Section 5.

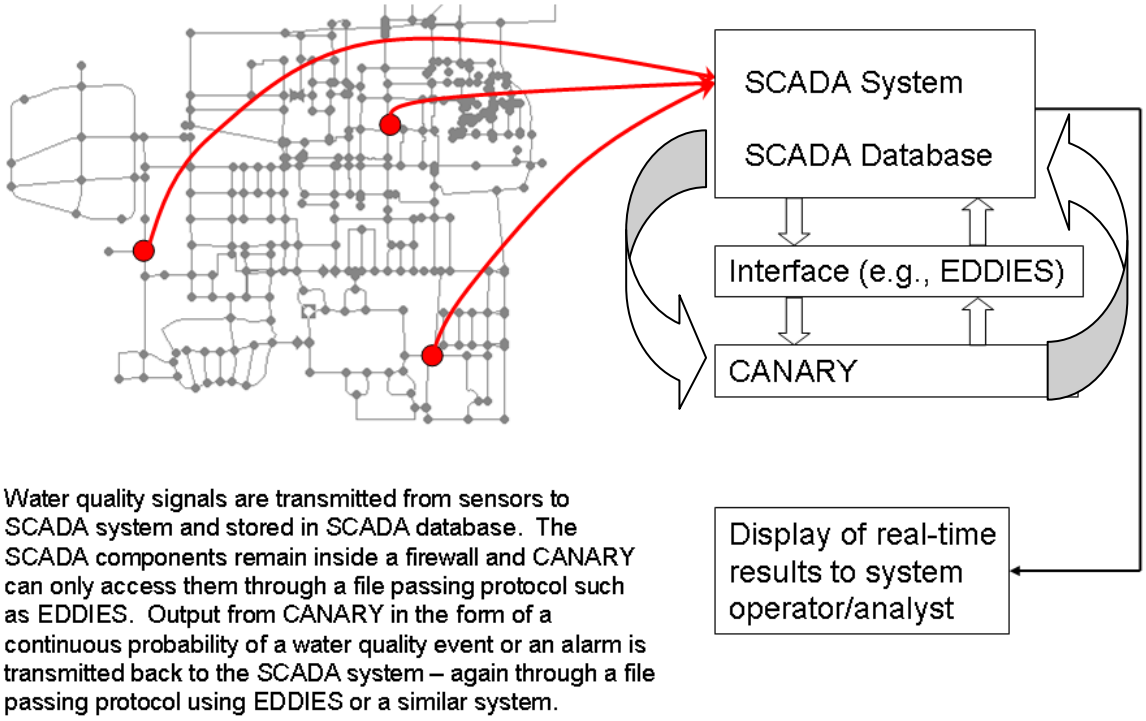

Figure 1. CANARY's interaction with a network of online sensor stations and a SCADA system.

2.1 Offline Mode

CANARY was originally designed to test the feasibility of using various algorithms to detect anomalous events in large sets of historical water quality data. These data sets were generally contained in a spreadsheet or text file, but occasionally in simple databases. CANARY's offline mode reads in the entire data set from a file or database and then processes it one time step at a time, as if the data were coming into the system sequentially. It does this without any delay between time steps but internally tracks the real-world time interval that would have occurred. As results are calculated, text messages are printed to a command prompt window, and are also saved in output files. The boxes highlighted in blue in Figure 2 Indicate the quantities written to output files and saved in offline mode.

Offline mode can calculate performance metrics of different algorithms if "real" event information is provided with the historical data. Real event information is needed to calculate detection rates (both true positive and false negative rates). Background data without any real events is suitable for calculating false positive rates. Real event data can be real world tracer tests performed in the field, data from laboratory studies, or simulated events added onto background water quality. If records are available, water main breaks, treatment plant changes, or other occurrences that could affect water quality can be incorporated and used as events.

The three main motivations to use CANARY in offline mode are to:

1. Configure algorithm parameters for a monitoring station using historical data without any known water quality events. This mode is used to identify the algorithm parameters that both create the best estimations of the measured water quality values (minimization of the residuals between predicted and measured water quality values) and reduce the false positive event alarms. Currently, this is the most common offline use of CANARY as it is necessary to define the correct parameters for each monitoring station and most historical data sets do not contain known events.
2. Adjust algorithm parameters for a monitoring station using historical data that contains known events. In order to determine the ability of the algorithm parameters to minimize the number of missed detection (false negative alarms), it is necessary to have some known events in the historical data set. Few historical data sets contain known events, and most do not have enough events to provide the statistical number necessary for setting algorithm parameters. In most cases, the known events must be simulated and added to the measured water quality data.
3. Identify recurring water quality patterns. CANARY contains a pattern recognition approach to construct a pattern library based on historical data. The patterns represent changes in water quality that could trigger an alarm under normal operation of CANARY. Once stored in the library in offline mode, these patterns can then be compared to any measured water quality that may trigger an alarm during online operation and can help to reduce false positives.

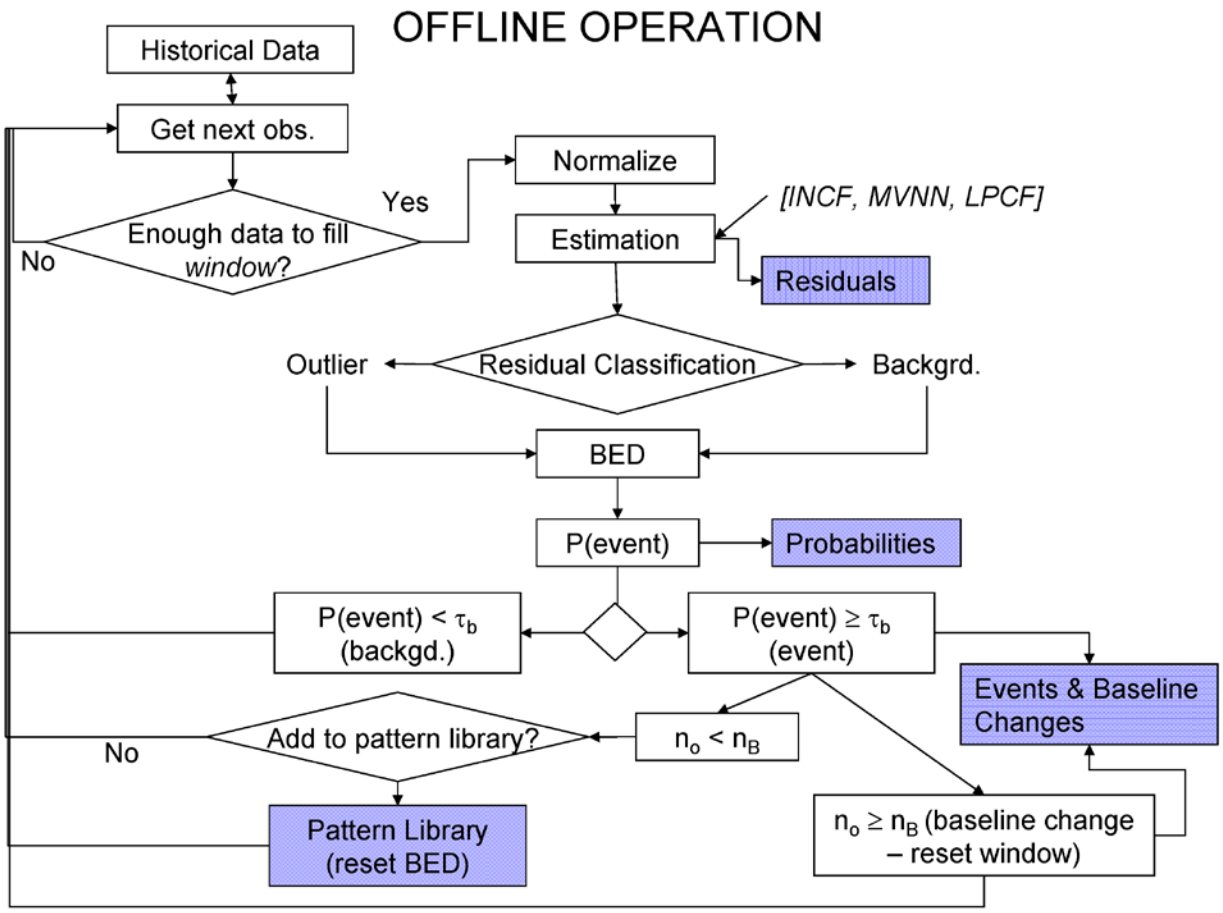

Figure 2. Flowchart for CANARY's offline operation.

2.2 Online Mode

While event detection on historical data is useful, water utilities need to be able to detect water quality changes in real-time. CANARY's online mode provides this functionality. Figure 3 illustrates the process flow for running CANARY in online mode. Multiple stations can be monitored simultaneously, and results can be output both to the command prompt window and back to a SCADA system. One option for connecting CANARY to SCADA is the EDDIES software, which is available from the U.S. EPA Water Security (WS) initiative pilot program. CANARY has direct interface capability with the EDDIES database for use in WS initiative pilots, and the details are discussed in the EDDIES.ReadMe file included in the distribution of the CANARY event detection software.

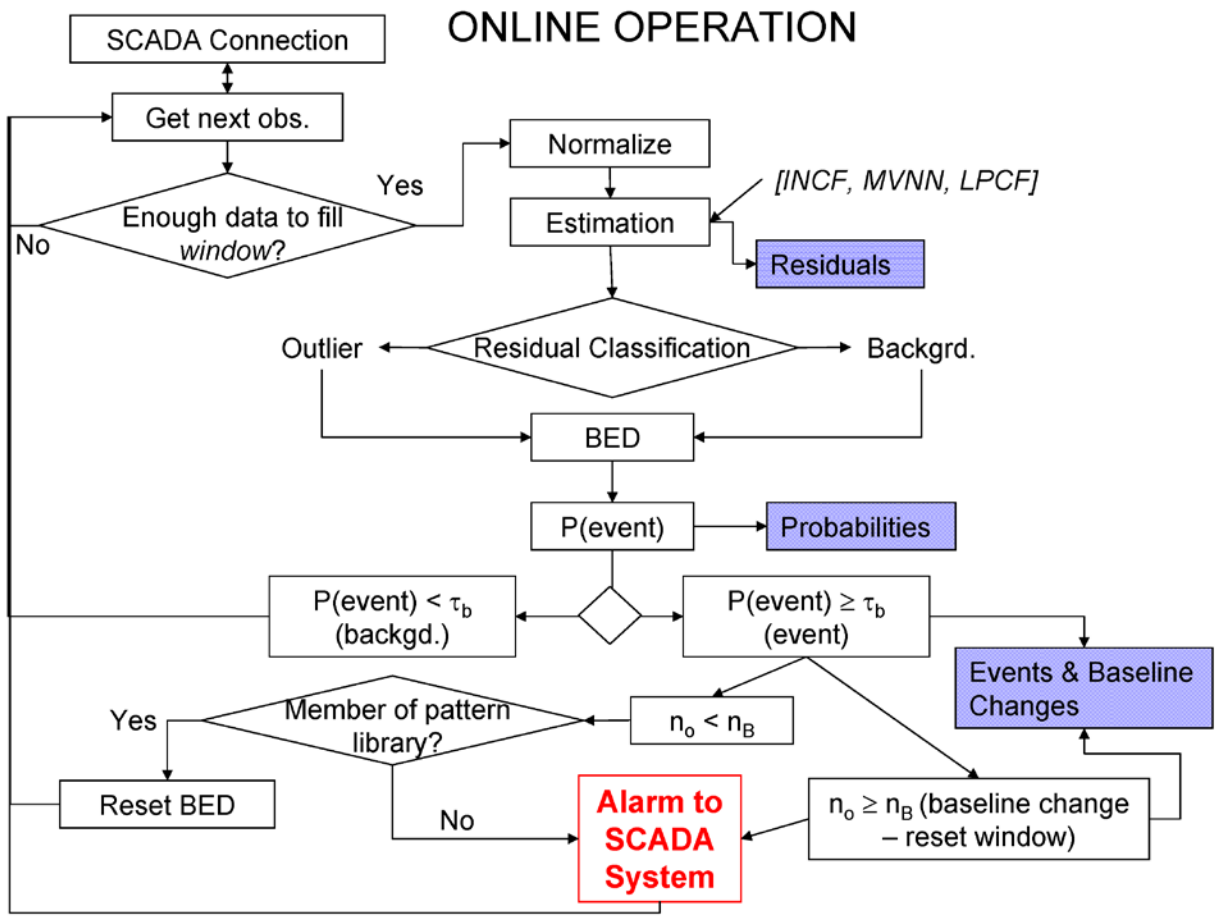

Figure 3. Flowchart for CANARY's online operation.

2.3 Terminology

The terminology used in this document is clarified below. A graphic showing some of the terms is provided in Figure 4.

- **signal** – Every data stream is considered a signal. Usually, the data comes from a SCADA system, and is composed of water quality (WQ) data (e.g., free chlorine concentration) and operations (OP) data (e.g., tank levels, flow rates) over time. Some sensor hardware provides error data to the SCADA controller and this information can be used by CANARY as an alarm (ALM) signal.
- **data-interval** – CANARY requires time-series data. These data are discretized according to the data-interval (sampling interval) of the sensor. A common data interval is two minutes. If data are provided more often than the data-interval, then only the most recent values are used by CANARY. If data are only provided every few data intervals (for example, total organic carbon sensors tend to take several minutes to process a single measurement and reset), the sensor

hardware generally provides the same value until new data is available, in which case the same value is sent to CANARY for multiple data-intervals.

- **precision** – The sensor hardware typically has a precision limit. The SCADA system, however, might send CANARY a default number of decimal places, which is typically much larger than warranted by the precision of the sensor. By specifying the precision of a signal, CANARY is able to recognize that what might look like a significant change (e.g., conductivity going from 310 to 320) is in fact only a small change given the precision of the sensor (e.g., 10). Setting this precision value properly can help reduce false alarms on very stable signals.
- **monitoring station** – A monitoring station is a set of sensors and their time series signals that are used together for event detection. These signals are generally from sensor hardware that is physically all at the same geographical location. Generally, a monitoring station always includes some water quality signals, and sometimes has operations or alarm signals.
- **sub-station** – A sub-station is defined as a monitoring station that is physically located at the same place as other monitoring stations. For example, a single tank might have multiple outlets that are monitored separately in CANARY, but which are located in the same building. In this case, each outlet becomes a sub-station in CANARY in order to differentiate them.
- **residual** – The residual is calculated by CANARY and is the difference between the predicted and measured water quality values at a single time step. The size of the residual is compared to the threshold, τ_a (see next item in list for the definition), in order to determine if the measured water quality is indicative of background water quality conditions or of anomalous conditions. Residuals can be positive or negative, indicating an under- or over-prediction of the measured water quality; however, only the absolute values of the residuals are used to compare against the threshold.
- **threshold** – The threshold value is the key parameter for residual classification. The threshold τ_a is in units of standard deviation, σ, not in the native units of the data. This allows data of differing ranges and units to use the same threshold. The standard deviation is calculated from the background data included in the normalization window (see next item in list). Setting the threshold as a relative measure in terms of standard deviations allows the absolute value of the threshold to increase and decrease depending on the variability of the measured data. Periods of higher background variation are compared to a higher absolute value of the threshold, and vice versa for periods of lower background variability.
- **normalization window** – This is a parameter used within CANARY and set by the user. It defines a moving window of time steps that contains the most recent data measurement for a given signal. The data values within this moving window are normalized to have a mean of zero and standard deviation of 1.0. The user, with consideration of site characteristics, sets the length of this window. For example, a site below a tank that fills and drains on a daily schedule may need a window size of slightly longer than one day, while an in-network location may only need three to six hours of history.

- **outlier** – An outlier is defined as a data value at a single time step that is considered anomalous relative to the background or predicted behavior for that time step. An outlier occurs when when one or more signals deviate from the predicted value at a time step by more than the threshold.
- **event** – An event is a period of sustained anomalous activity, when many outliers occur in a short period of time. The binomial event discriminator (BED) is a statistical algorithm that is used to decide how many outliers are needed in a given time period to create an event. The BED parameters are determined by the user.
- **probability threshold** – The probability threshold, τ_B, defines the probability of an event that must be exceeded before an event is signaled by CANARY.
- **baseline-change** – When an event has continued for a user-defined period of time, a baseline-change is said to have occurred. This means that the event has sounded a warning for long enough that two things have happened:
    - the CANARY alarm is no longer needed: the operators have identified the cause or initiated action to find the cause, and,
    - the user would like CANARY to begin looking for new events: enough time steps have passed that the user would like CANARY to start operating based on this "new normal background water-quality.

  Baseline-changes are significant because they tell CANARY to stop signaling that an anomalous water quality event is occurring. A baseline-change can only occur after an event, but not all events are long enough duration to become baseline-changes.
- **water quality pattern** – A recurring trend in one or more WQ signals that would normally be of significant magnitude to cause an alarm, but which is considered by the user to be a "normal event." Such patterns could be caused by routine operations such as daily demand changes, pumps or plants turning on and off, or water treatment activities. If the pattern is regular enough, it can be identified, stored in a library, and used as a recognized pattern to help eliminate false alarms associated with normal activities.
- **clustering** – The process of identifying water quality patterns is called clustering. Cluster files are used to keep a library of patterns to identify "normal events" and decrease false alarms. Typically, one or more OP signals provide useful information to the clustering algorithm.

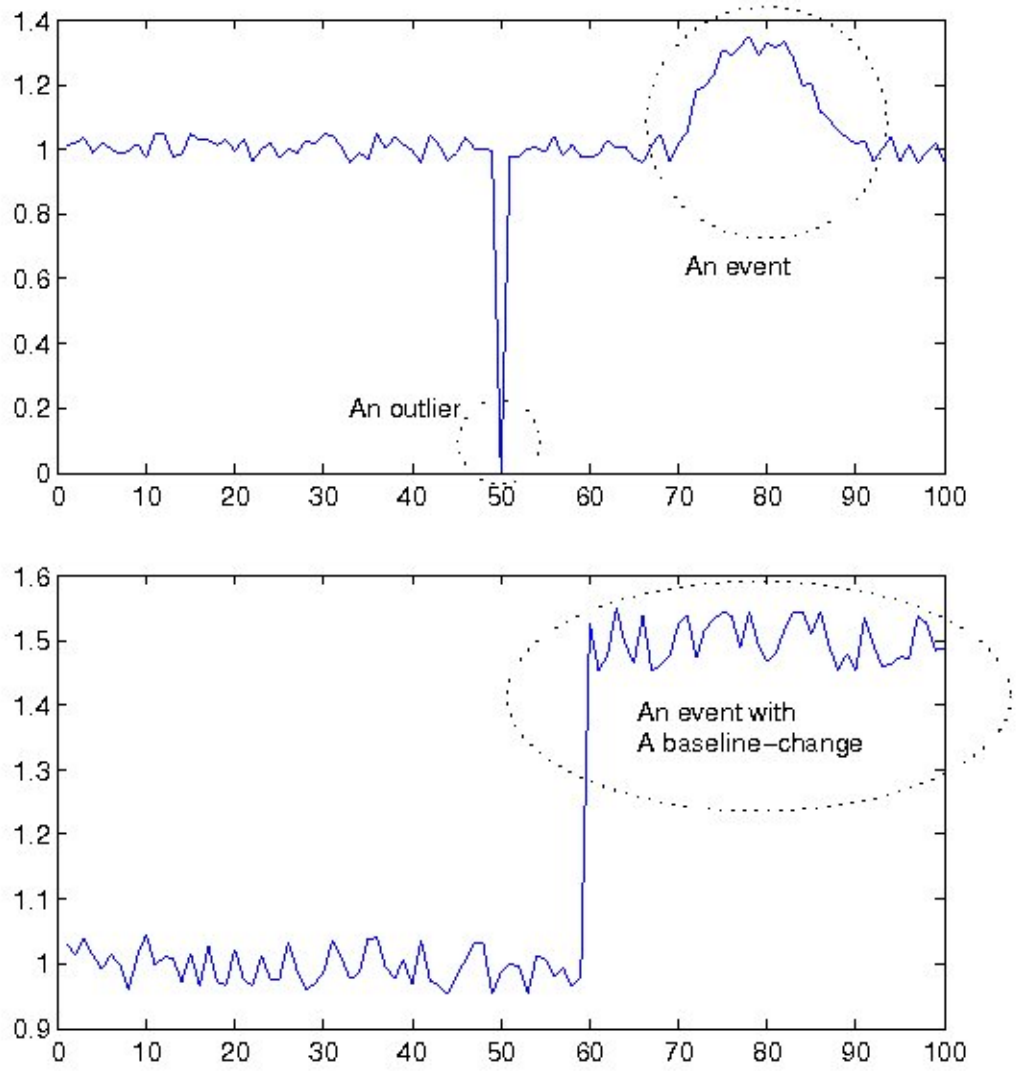

Figure 4. Graphical representation of an outlier, an event, and a baseline change.

2.4 Water Quality Estimation and Residual Classification

This section describes how CANARY uses water quality signals from previous time steps to estimate values for the next time step, and calculate and classify residuals.

2.4.1 Normalization Window

Because the signals that CANARY analyzes are of different magnitudes and measured in different units (e.g., mg/L, NTU, pH), the data is normalized. A window of time steps, the normalization window specified by the parameter n_h, determines the number of data points included in the normalization of the data. The historical data in the window, X_h, are then normalized by their mean and standard deviation to obtain the normalized data, X_S:

$$X_S = \frac{X_h - \bar{x}}{\sigma_x}$$

The normalized data is a set of length n_h, with a mean near 0.0 and a standard deviation, σ, near 1.0; however, the data does not typically constitute a normal distribution. As CANARY advances to a new time step, the oldest data point is removed from the window, the most recent data point is added to the window, and the data in the new window is normalized again using a new mean and standard deviation.

2.4.2 Event Detection Algorithms

The main algorithms used in CANARY and their associated parameters are described below. Peer-reviewed publications are referenced that can provide additional background on the algorithms. Additionally, Murray et al. (2010) provides more detail on the motivation, development, and application of CANARY.

Because event detection systems have an inherent trade-off between high sensitivity and the number of false alarms, each user must decide the best parameters for each specific use of CANARY. For example, a researcher might have more tolerance for false alarms while testing a new algorithm or piece of sensor hardware, while a utility operator could decide to use settings that provide less sensitivity in order to decrease the number of alarms which require further investigation and response.

2.4.2.1 Linear Prediction Coefficient Filter (LPCF)

The LPCF algorithm uses digital filtering along with linear coefficient estimation to predict the next observation based on the trends of the data in the historical window. The new water quality value is predicted using a weighted average of the previous values contained in X_S. The linear coefficients provide the weight for each previous value and are updated at every time step to adapt to changing background water quality. The coefficient calculations are formulated to provide estimates that are unbiased and have minimum variance. The algorithm uses the **filter** and **lpc** functions from MATLAB, a description of which can be found in MathWorks (2002), or in McKenna, Klise, and Wilson (2006). The threshold, τ_a, refers to the maximum error allowed, in units of standard deviation, between the predicted data values for the new time step and the measured data values. Error values or residuals greater than τ_a are considered outliers. Typical values for τ_a might range from as low as 0.5σ at stations with very stable water quality, up to 1.5σ or higher at stations with unstable water quality (e.g., mixing from multiple sources). This algorithm processes each signal separately, and then compares the largest absolute error to the threshold.

2.4.2.2 Multivariate Nearest Neighbor (MVNN)

The MVNN algorithm compares the newest measured water quality data against all the data in the history window. The normalized water quality data are mapped in an m-

dimensional multivariate space where m is the number of signals. In a two-signal example, which is the easiest to visualize, the data create a series of points in 2-D space (e.g., residual chlorine vs. pH). The newest measurement is plotted against the others. The residual is the distance from the new point to the closest historical data point. The threshold, τ_a, is the maximum allowable distance the new data point can be from its nearest neighbor, in units of standard deviations. The MVNN algorithm works for any number of signals, and, in contrast to the LPCF algorithm, only one residual value is calculated to be compared against the threshold, no matter how many signals are used. Additional information on the MVNN algorithm is available in Klise and McKenna (2006a; 2006b). Typical values for τ_a range from 0.5σ to 3σ.

2.4.2.3 Set-point Proximity Algorithms (SPPB and SPPE)

The SPPB and SPPE algorithms were new to CANARY in version 4.1. The goal of these algorithms is to provide a ramped warning as any water quality signal approaches a minimum or maximum set point limit established by a water utility. The threshold, τ_a, takes on a slightly different meaning for these algorithms, helping define a distance, ΔR, away from either set point value over which the probability of an event increases away from zero. The distance, ΔR, is calculated as: $\Delta R = \tau_a$ x (sensor precision) and is the number of precision “steps” away from the set point limit at which the probability of an event value starts to rise from zero. The two different options, SPPB and SPPE, indicate which probability distribution to use to define the increase in probability of an event as a function of the normalized distance away from a set point limit. The options available are a Beta distribution (SPPB) or an exponential distribution (SPPE). The beta distribution increases probabilities more quickly than the exponential distribution. Testing to date has shown that the SPPE algorithm is probably best for daily operations, while the SPPB algorithm can provide more sensitivity, earlier warning at monitoring stations.

A schematic diagram of this algorithm is shown in Figure 5. The Y-axis on the left side of the figure shows the range of water quality values between the two set points, minimum and maximum. The right side of the figure shows this same range divided into two lengths of ΔR ending at each set point value and the range of zero probability of event in the middle of the water quality range. As any water quality value becomes close to a set point (minimum or maximum), the probability of an event increases along the selected dashed line (SPPE or SPPB). Similar to the other algorithms, an event is signaled when the probability of an event exceeds the event threshold. These are perhaps the simplest event detection algorithms within CANARY as it is not necessary to use previous water quality data values or the binomial event discriminator to calculate the probability of an event. The user-adjustable parameters for this algorithm are the selection of SPPB or SPPE, the number of precision steps, entered as τ_a, and the probability threshold.

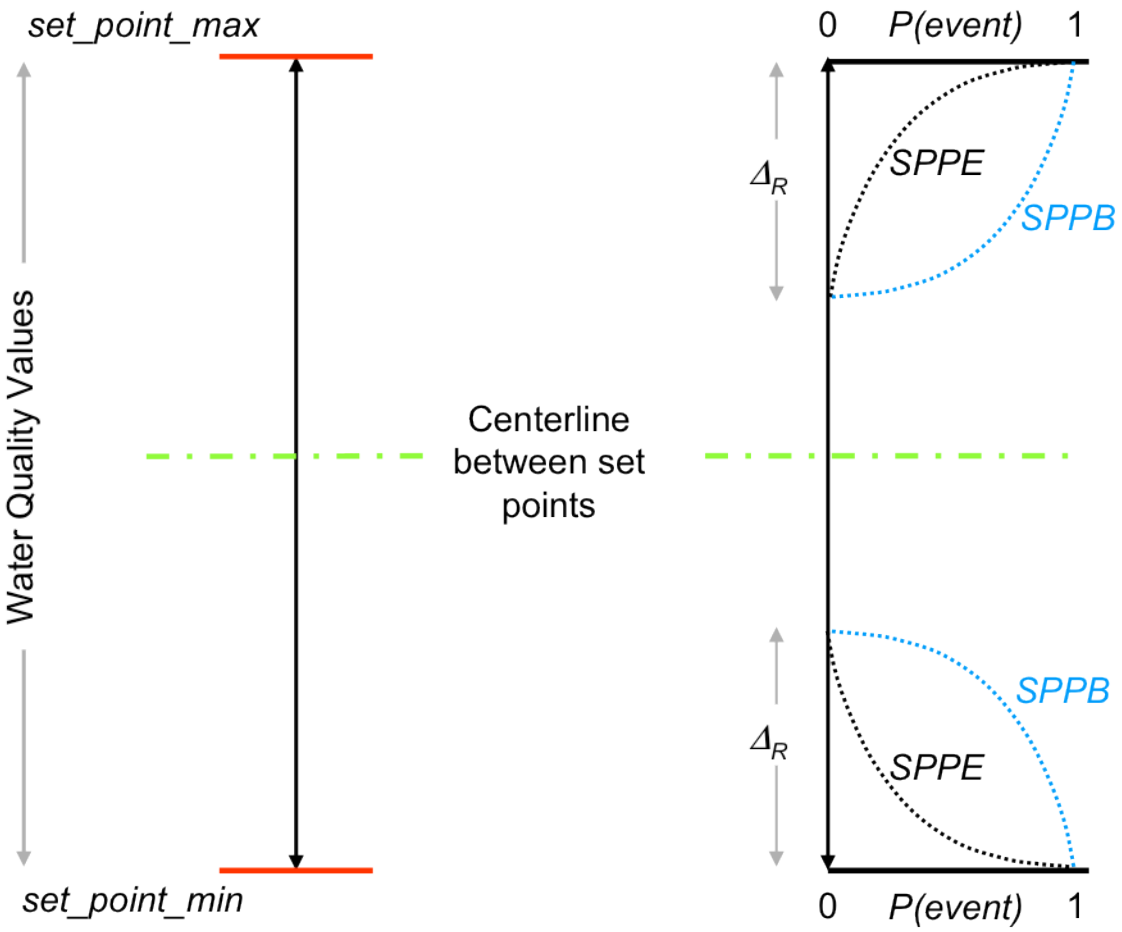

Figure 5. Schematic diagram of set-point algorithms SPPB and SPPE.

2.4.2.4 External / Custom Algorithms (JAVA)

Because CANARY was developed with algorithm testing in mind, users can enable external, compiled Java functions to calculate the residual. The application-programming interface (API) is described in the DummyAlgortihm.java file included in the CANARY software distribution.

2.4.2.5 Consensus Algorithms (CAVE and CMAX)

In real-time SCADA operations, CANARY might be limited to reporting output from a single algorithm. If the user wants to combine two or more algorithms in a single report, the consensus algorithms "CAVE" and "CMAX" are two methods of accomplishing this task. These consensus algorithms combine the outputs of more than one event detection algorithm. The "CAVE" algorithm calculates and reports the average probability of an event based on the probabilities from the other algorithms. The "CMAX" algorithm reports the maximum probability of an event from the other

algorithms' probabilities. The analyst might want to have duplicate outputs (such as a text file) to aid in identifying which algorithm caused an alarm.

2.5 Water Quality Event Determination

The estimation and residual classification algorithms described above provide a binary determination at each time step: outlier or background. These results are used as input to two additional pieces of the water quality event identification: (1) calculation of the probability of a water quality event at each time step and (2) determination if a measured outlier time step is part of a previously recognized water quality pattern. Both of these additional processing steps are designed to reduce the number of false positive alarms produced by CANARY. Neither of these additional steps is necessary if only a binary outlier/background indication of water quality is required. However, the determination of the event probability at each time step provides a much richer picture of water quality changes within a distribution network and the integration of results across time steps reduces false alarms. For monitoring stations with water quality changes induced by operations changes (e.g., monitoring stations near a tank), the pattern matching capability can provide a significant reduction in the number of false positives.

2.5.1 Binomial Event Discriminator (BED) and Event Time-out (ETO)

The binomial event discriminator is not an event detection algorithm itself, but an algorithm designed to integrate results over multiple time steps to calculate the event probability and to help limit false positives. As described in McKenna et al. (2007), the BED uses a separate, smaller history window to track the number of recent outliers. This window is n_B time steps long. The probability of n_O outliers occurring in n_B trials, when the probability of an outlier occurring at any given time step is P_B, is defined as P_E, where P_E is the output of the **binocdf**(n_O, n_B, P_B) probability calculation. If P_E exceeds the probability threshold, τ_B, then CANARY signals an event alarm. At each successive time step, the newest outlier calculation is added to the window, and the oldest is removed, and P_E is recalculated. If the event continues for the number of time steps that determines a baseline-change, n_{ETO}, then a baseline-change is noted, and all the algorithms are reset.

Unlike the estimation algorithms, which provide a Boolean "above threshold" (outlier) or "below threshold" (background) indicator, the BED provides the continuous probability that an event has occurred at each time step.

The values of the BED window, n_B, and baseline-change ETO window, n_{ETO}, are typically chosen for a specific class of events. For example, if anomalies lasting less than four time steps are not of interest, n_B should be set larger than four to avoid short anomalies sounding an event alarm. If operators believe that one hour is enough time to evaluate or start action on an alarm, then n_{ETO} should be set to the number of time steps that occur in one hour. Because of this, the BED or ETO parameters do not have "typical" values. Also note that the ETO window can be set regardless of whether the BED is used.

2.5.2 Water Quality Pattern Matching

A recurring issue in the application of water quality EDS tools is false positive alarms resulting from changes in water quality that are linked to routine adjustments in hydraulic operations. Uncertainty in water flows and demands makes it difficult to directly correlate operational adjustments at one location with the timing and characteristics of a water quality change at monitoring station elsewhere in the network. CANARY has the capability to create a multivariate pattern library of previously observed water quality changes, and compare any new potential event against that library to see if it has happened previously. Additional background and several example applications are available in Vugrin et al. (2009).

The water quality pattern matching tool uses the idea of trajectory clustering to represent a time-series of water quality data by a relatively low-order polynomial. In the regression, each water quality signal is the dependent variable and time is the independent variable. Regression models and clustering of the regression coefficients, not the actual data, are used to construct a multivariate library of common water quality patterns from historical data. The length of the time series in the pattern is generally limited to less than 50 time steps prior to and including the indication of a water quality pattern by CANARY.

Starting at the current time step and looking back n_C time steps, a low-order (e.g., 3rd-order) regression model is fit to each signal. The fits are done independently on each signal and, in the example case, the coefficients are quite different (decreasing vs. increasing vs. flat), but the degree of the polynomial model (e.g., 3^{rd}-order) applied to all signals must be constant. This process is repeated for all data that the user deems representative of common water quality patterns. The library of patterns is stored within a matrix that has one row for each event that was identified by CANARY. The total number of columns is the order of the polynomial plus one times the number of water quality signals examined (an n^{th}-order polynomial has n+1 coefficients). For example, use of a third-order polynomial for analysis of three water quality signals results in 12 columns in the matrix.

Online water quality pattern matching uses the information in the pattern library to discard potential events that are similar to previously identified water quality patterns. In this case, if the data closely matches a known event (within the 20^{th} percentile, for example), then a message is submitted that a known event has occurred, and no other alarm is sounded. If no known cluster is similar enough to the current data, processing continues, and an alarm might be produced. Operational data streams, such as flow or pressure, can be used in conjunction with water quality data streams to define the multivariate patterns, but only water quality data causes a water quality event.

The following steps are used to construct a pattern library from historical data

1. Gather four to six months of historical water quality data for a given location into a file format that can be read by CANARY (e.g., comma separated value (CSV) file).

2. Using the event detection parameters intended to be used for online event detection, analyze the historical data with CANARY. The output EDSD file contains all of the events detected by CANARY.
3. Right-click on an EDSD file to create a cluster pattern library. A graphical user interface opens to display a dialog box with the option to either have CANARY automatically place all identified events into the pattern library or to allow for manual selection of events into the library. It is strongly recommended that the manual selection be used to avoid unwanted water quality changes becoming part of the pattern library.
4. Figure 6 shows a time series of the water quality signals (first three graphs) as well as the probability of an event as predicted by different algorithms (last three graphs). Blue dots indicate the time steps at which an event has been identified. The screen allows the user to select “Yes” or “No” for inclusion of an event in the pattern library. The data are aligned such that there are n_C time steps between the left edge of the screen and the start of the event. The data for all signals within those n_C time steps becomes a single event in the pattern library. In the Figure 6 example, the water quality event begins at time step 393 and is caused by the decrease in pH that began at time step 386 or 387. Twenty time steps are included from the start of the event back to the left edge of the figure ($n_C = 20$). For this example, a combined event detection algorithm was used with both SPPE and MVNN algorithms active and the individual and combined probabilities of event shown in the bottom three graphs (Figure 6). The MVNN algorithm identified the water quality event in this example.
5. When all events within the data file have been examined, CANARY creates the pattern library in an EDSC file.
6. Right-click on an EDSC file to open the cluster pattern library editor. This editor allows the user to visualize the multivariate patterns assigned to each cluster within the pattern library. An example pattern library editor is shown in Figure 7. Four patterns were created from the data and pattern 3 is open. Thirty time steps are in each pattern, $n_C = 30$, and the changes in water quality for pattern 3 show a sinusoidal increase in total residual chlorine (TRC), a sinusoidal decrease in pH, and linearly decreasing values for conductivity. The pattern definitions for this example also include the rate of flow through the monitoring station and the flow values are flat. The black and grey lines are the individual events in the pattern and the red lines are the weighted mean signal for each pattern and signal. Currently, the pattern library editor only allows editing of the pattern ID and the description, the bottom two fields on the left-side panel of the editor (Figure 7). Future versions of CANARY will allow events to be added or deleted from individual patterns.

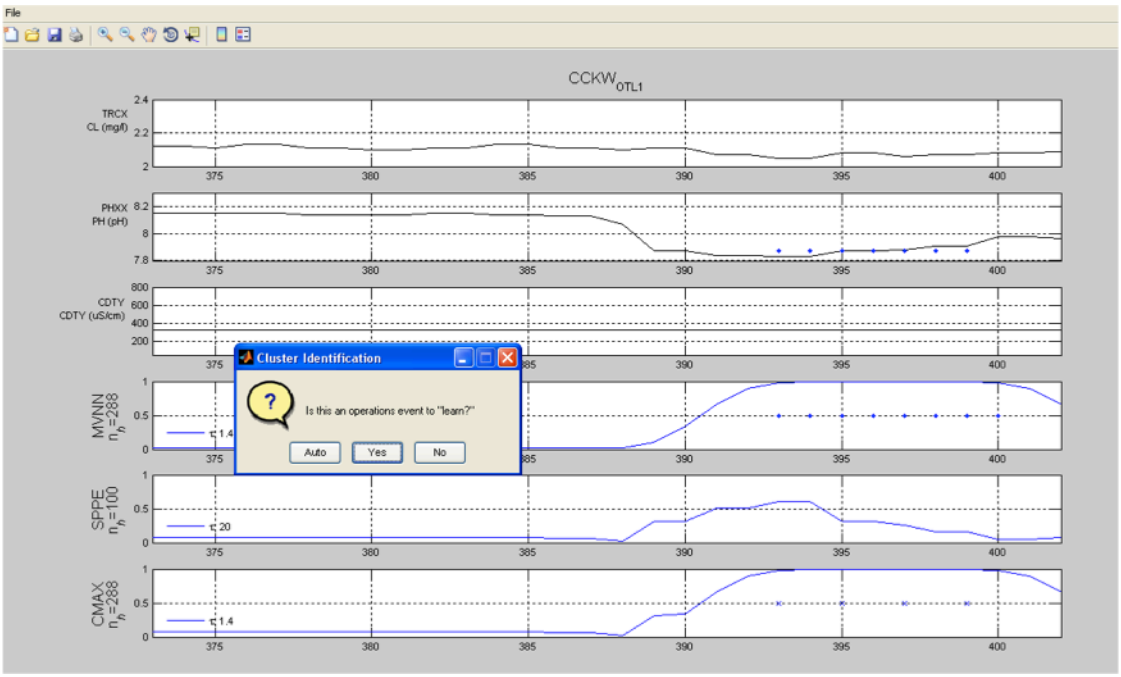

Figure 6. Example screen from event selection portion of pattern library creation.

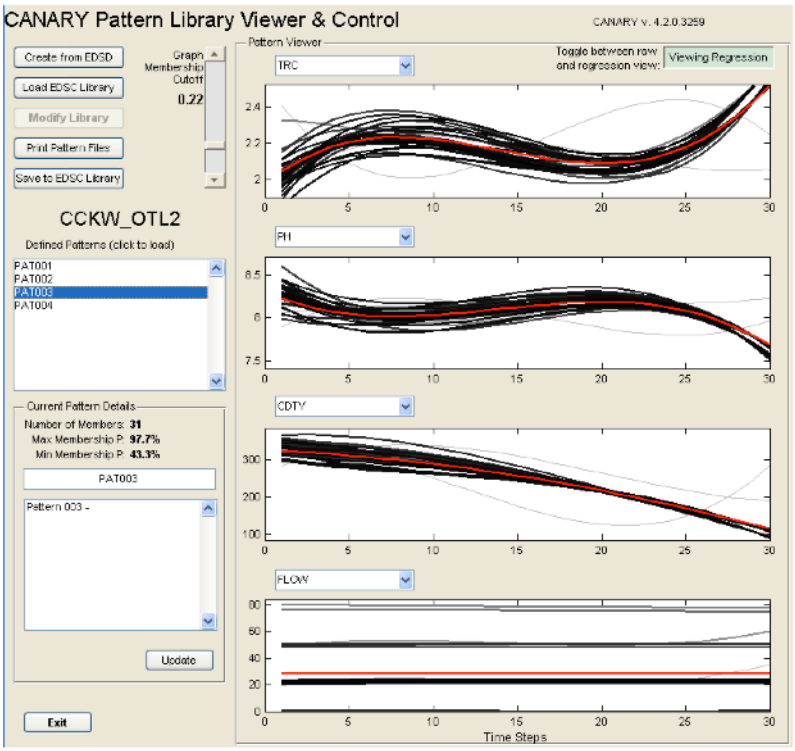

Figure 7. Example screen from the pattern library editor.

3 CANARY Inputs

CANARY inputs can come from one or more data sources. In this section, the format for an input-type data source is discussed.

Each data entry in the data source must include four different pieces of information: the date and time of the measurement, the name of the sensor taking the measurement, the name of the monitoring location in which the sensor is installed, and the data value itself. The date and time are converted into a discrete time index, k; the sensor identifier identifies which data column the value belongs to, with an index j. The monitoring location specifies which data set to access.

Additional information can be used when running CANARY in real-time mode, such as alarm flags, quality flags, comments, but the four data pieces described above are the only required elements of the input data. Table 1 shows the different entry values for the input-type parameter in the configuration file. The data is organized in two primary ways: column-based (field-based) and row-based (record-based).

Table 1. Acceptable values and related descriptions for the *input type* configuration parameter.

Data Source *input type*	Brief Description
CSV	A text or spreadsheet file, in comma-separated values (CSV) format. Described in Section 3.1.
DB, JDBC	A database table, in one of several possible formats. Described in Section 3.2.
EDDIES	A specific database format for use with the EDDIES software. See Section 3.3.
XML	A hardware specific XML-formatted message-passing format. See Section 3.3.

3.1 CSV Type Input Data Sources

One of the most common formats used to transfer data from one application to another is comma-separated values (CSV) format. Each column is separated by a comma (the "," delimiter) and a new-line indicates the end of a record; generally, the first row defines the column labels. When CANARY uses CSV files as input, it follows this convention.

The first column of a CSV CANARY file should contain the date and time of measurement, and be titled "TIMESTEP" or something similar. The exact name should be indicated in the *field* parameter of the *timestep options* in the configuration file settings for the data source. Each additional column should contain the SCADA tag

name for a signal defined in the configuration file. These tags should be unique and often are chosen to denote the monitoring station and sub-station as well as the type of signal "**WQ**" or "**OP**." An example CSV file (as it would be displayed in a spreadsheet application) is shown in Table 2. CSV files are most useful when examining historical data offline, though if a SCADA system outputs to a CSV file every time step, CSV files could theoretically be used in online mode too.

Table 2. A sample showing the header and five data rows from a CSV data file.

TIME_STEP,	TEST1_CL2,	TEST1_COND,	TEST1_PH,	TEST1_TEMP,	TEST1_TOC,	TEST1_TURB,
1/1/2008 0:00,	0.9,	308,	8.78,	10.1,	1.01,	0.04,
1/1/2008 0:02,	0.9,	308,	8.78,	10,	1.01,	0.04,
1/1/2008 0:04,	0.9,	309,	8.78,	10,	1.01,	0.04,
1/1/2008 0:06,	0.9,	308,	8.78,	10,	1,	0.04,
1/1/2008 0:08,	0.9,	309,	8.78,	10,	1,	0.04,

3.2 Database Input Sources (JDBC, DB)

Two different database input styles can be read by CANARY, and they vary depending on the record type that is used. The first style, which is a single record per time step, is a translation of the CSV style into a database. In this style, each field in the database table is named with the SCADA tag of the data that goes into that field. The time step information is also a field. All the data for a single time step value is contained in the fields of that record. This format is also the "default" format for a database typed input. The data in Table 2 adequately represents a default database.

The second format is called "**row based**" in the configuration file, and is perhaps a more common format in database tables. In this style, the name of the SCADA tag is a value within one of the fields of the table. An example is shown in Table 3 using the same data as the CSV data in Table 2. Many records with the same time step value are present, but each of them has a different SCADA tag value in another field. At a minimum, a table using this style must have fields for the time step, the SCADA tag name, and the actual value. If additional fields exist, CANARY ignores these fields. The names of the fields can be customized, but the data types and the default field names of an input table are listed below:

- **time-step** – the input table must contain timestamp information for each row.
    - field name – "TIME_STEP"
    - data type – DateTime
- **parameter tag** – the SCADA tag associated with a particular value must be on each row in the table.
    - field name – "TAG_NAME"
    - data type – String

- **parameter value** – the value must be included on each row in the input table.
    - field name – “VALUE”
    - data type – Real (or other double-like number)
- **parameter quality** – a quality string can be included in the input table, but is not read by default.
    - omitted by default
    - data type – String

In both cases, the name of the field that contains the time step data should be provided in the “time-step field” parameter in the configuration file, and should be of "DateTime" type in the database. Please see the configuration instructions in section 5.3.

Table 3. Example table from a database using the *input format:* row-based.

TIME_STEP	TAG_NAME	VALUE
2008-01-01 00:00:00	TEST1_CL2	0.9
2008-01-01 00:00:00	TEST1_COND	308.0
2008-01-01 00:00:00	TEST1_PH	8.78
2008-01-01 00:00:00	TEST1_TEMP	10.1
2008-01-01 00:00:00	TEST1_TOC	1.01
2008-01-01 00:00:00	TEST1_TURB	0.04
2008-01-01 00:02:00	TEST1_CL2	0.9
2008-01-01 00:02:00	TEST1_COND	308.0
2008-01-01 00:02:00	TEST1_PH	8.78
2008-01-01 00:02:00	TEST1_TEMP	10.0
2008-01-01 00:02:00	TEST1_TOC	1.01
2008-01-01 00:02:00	TEST1_TURB	0.04
2008-01-01 00:04:00	TEST1_CL2	0.9
2008-01-01 00:04:00	TEST1_COND	309.0
2008-01-01 00:04:00	TEST1_PH	8.78
2008-01-01 00:04:00	TEST1_TEMP	10.0
2008-01-01 00:04:00	TEST1_TOC	1.01
2008-01-01 00:04:00	TEST1_TURB	0.04
2008-01-01 00:06:00	TEST1_CL2	0.9
2008-01-01 00:06:00	TEST1_COND	308.0
2008-01-01 00:06:00	TEST1_PH	8.78
2008-01-01 00:06:00	TEST1_TEMP	10.0
2008-01-01 00:06:00	TEST1_TOC	1.00
2008-01-01 00:06:00	TEST1_TURB	0.04

3.3 Custom Inputs (EDDIES and XML)

Some CANARY users might prefer to use middle-ware applications that separate the SCADA system from the event detection system, and reformat and screen information prior to passing information back to the SCADA system. These formats tend to combine control elements and input/output elements into a single package, defining how CANARY should communicate with the system. These types of inputs are not typically used, and are not discussed in detail here. For more information about the EDDIES software, contact Katie Umberg at EPA: umberg.katie@epa.gov.

4 CANARY Outputs

CANARY produces several types of output. The primary output is to the command prompt window on which CANARY is running. This same information is also copied to a log file. In addition, CANARY provides notification to the SCADA database, or another output database, when it detects an event.

For offline operation, real-time alerts are not needed. CANARY provides CSV file output that lists the date and time of events, the name of the algorithm that detected the event, and CANARY's estimated probability of an event at every time-step. Regardless of the mode of operation, CANARY always creates an *.edsd binary file (EDSD file) and a summary text file that contains a record of all the results calculated during a particular CANARY run.

The output options are described in Table 4.

Table 4. Acceptable values and related descriptions for the *output type* configuration parameter.

Data Source *output type*	Brief Description
<always provided>	Certain output is always provided. This includes a log file, a binary EDSD data file, a summary text file, and output to the screen. Described in section 4.1.
DB, JDBC	Output results to a database table. Described in section 4.2.1.
EDDIES	Output to the EDDIES database. Counterpart to the *input type:* **EDDIES**.
XML	Output using the XML messaging. Counterpart to the *input type:* **XML**.

4.1 Output Always Provided

Certain output is always provided by CANARY: output to the screen, to a log file, and to a binary data file. CANARY also produces a summary of the results for batch runs (or real-time runs that are exited cleanly). The binary EDSD files can be converted to spreadsheets by right-clicking on the file, or by running "canary --convert" from the command line.

4.1.1 Console Output and Log File

CANARY does not provide command window output for every time step processed because of the significant increase in the required processing time. However, to indicate

to the user that CANARY is functioning properly, a message is listed to the screen after every day (24 hours) of data processed. A typical message is similar to the one below:

```
Info:  07/17/2010 11:56  Time to process day: 5.9 sec; est. remain: 2.1 min
```

When an event or other warning occurs, CANARY prints this information to the screen. The message indicates that an event has been detected along with the date, time, the monitoring station name associated with the event, and the name of the algorithm that detected the event. A typical event message is similar to the one below:

```
WARN:  07/17/2010 21:34 RMTP - LPCF{0.9}:(98.1%)
```

4.1.2 EDSD Data Files

Regardless of the mode of operation, CANARY always produces a data file with the extension ".edsd" called the EDSD file. The EDSD file is a binary file which contains the time-series data and the results of the CANARY analysis. CANARY creates a new EDSD file at the end of every day (24 hours) of data processed, after the first two days of execution. After exiting, CANARY combines all the daily EDSD files into a single EDSD file. In Windows®, this file (or any EDSD file) can be right-clicked to bring up the content menu. Four options are available:

- **Graph Data** - the default option if the file is double-clicked. This brings up the graphing tool which creates PNG formatted files. Input and output data from multiple locations can be graphed at once, but each picture has data from only one station. Time scales can be selected by the user and include daily, weekly, and monthly graphs.
- **Convert to CSV** - this option creates CSV files from the EDSD output data file. A summary CSV file is created for each location, listing the probability and event code for each algorithm. In addition, a details file is created for each location/algorithm pair. The details file contains probabilities, pattern match details, residuals, contributing factors, and comments, as well as event codes.
- **Create Cluster** - use this option to launch the pattern matching cluster creation utility from a completed CANARY run. See Section 2.5.2 for more details.
- **Combine Files** - use this option to combine multiple EDSD files into a single EDSD file for easier processing.

The EDSD file can also be loaded directly into MATLAB as a MAT file, or it can be reused as an input to CANARY. The user does not need to specify an output in the configuration file to produce EDSD files.

4.1.3 Summary File

At the end of a run, before CANARY exits, a summary text file is created for each station. The first section of the summary file contains the configuration that was used for the run. This is followed by a list of the total number of events and their average duration. After this, each algorithm is described, followed by a list of each signal and the

number of events associated with each signal. Finally, each event is listed on a separate line, with the algorithm, start date, duration, pattern information, and contributing signals.

4.2 User-requested Outputs

If the JDBC or DB data sources output type is selected, CANARY writes results to a database.

4.2.1 Database Output (JDBC, DB)

CANARY writes to a database when the **DB** output type is used. This does not have to be the same database used for input. However, if the same database is used, the user-ID given to CANARY must have write privileges for the specified output-table defined in the configuration file.

CANARY creates, if necessary, and updates the table specified by assigning fields named after the different monitoring stations, and all data for a given time step is provided on a single row (record). One of three formatting options is selected by the user in the configuration file to define the output table. The first two, "**default**" and "**extended**," provide a specific subset of possible output values and have default field names. The third, "**custom**," allows the user to fully specify the names of each field and which fields should be used. This is more fully described in Section 5.3.

For the "default" format, default values for the field names in the output table are used. They are listed here, along with their type:

- **time-step** – the date and time of the result.
    - field name – "TIME_STEP"
    - data type – DateTime
- **instance id** – the name of the EDS instance that provided the result. The value is always "CANARY."
    - field name – "INSTANCE_ID"
    - data type – Char
- **station id** – the *station tag* is assigned to this field, allowing the user to identify which combination of algorithms and signals produced the result.
    - field name – "LOCATION_ID"
    - data type – Char
- **event code** – the event code for the result.
    - field name – "DETECTION_INDICATOR"
    - data type – Integer

- **event probability** – the probability that the current water quality is significantly anomalous.
    - field name – “DETECTION_PROBABILITY”
    - data type – Real
- **contributing parameters** – the *parameter type* of all the current outlier signals.
    - field name – “CONTRIBUTING_PARAMETERS”
    - data type – Char(100+)
- **comments** – any additional text comments produced by CANARY.
    - field name – “ANALYSIS_COMMENTS”
    - data type – Char(100+)

The “**extended**” format also has default values. They include all the fields from the “**default**” plus:

- **algorithm id** – the *id* of the algorithm that was used to produce the result.
    - field name – “DETECTION_ALGORITHM”
    - data type – Char
- **pattern match id** – the ID number of the pattern that was matched (if any).
    - field name – “MATCH_PATTERN_ID”
    - data type – Integer
- **pattern match probability** – the probability of membership in the pattern (if any).
    - field name – “MATCH_PROBABILITY”
    - data type – Real

4.2.2 Debugging Outputs

If debugging is active, two additional files, "debug.xml" and "debug.sql," might be created. The XML file contains debugging information regarding the internal workings of CANARY. The SQL file contains all the SQL queries and/or updates that CANARY sends to the database. Both these files might become very large, and debugging should be turned off during standard deployment of CANARY.

5 Configuration File Details

The format of CANARY's configuration file is YAML (extension of ".yml"). Prior to version 4.3.1, the configuration file was in XML format. A YAML configuration file is automatically produced when CANARY is run with an XML file. By convention, the YAML configuration file ends in "edsy" while a XML file ends in "edsx."See Appendix A - YAML Tips for more information about YAML formatting.

In what follows, portions of the configuration file are presented and described along with the associated options. A series of examples is provided and each line of the sample configuration files is explained. Comments about particular lines are noted. The file sections are formatted to make it as easy as possible for users to cut and paste these sections into their own YAML files. YAML files are intended to be simple to read and use. The only things most users need to remember are: list entries start with a "-" character, spacing is important for nesting, and key-value pairs have a ":" separating them. Finally, spaces should always be used, never "tabs."

5.1 Control Section

A CANARY YAML configuration file typically begins with a "canary" section heading. Control values listed underneath and are used to determine how CANARY runs – whether in offline or online mode – and to load any extra drivers needed to work with databases or home-grown algorithms. Two examples are provided in 5.1.1 and 5.1.2.

- *canary:* – This starts the CANARY control section.
    - *run mode:* – This parameter specifies the CANARY operation mode. This tag must occur exactly once in each configuration file. The options are **BATCH**, **REALTIME**, or **EDDIES**. **BATCH** mode is used for offline data analysis, for example, using a CSV file of historical data to evaluate parameter settings. **REALTIME** mode is used when connecting CANARY to a database for online data analysis. It analyzes the data in real-time and looks for new data at specified time intervals. **EDDIES** mode is a specific control/run option when using the EDDIES software for analysis.
    - *control type:* – This parameter defines how CANARY interacts with the SCADA system. The options are **INTERNAL**, **EXTERNAL**, or **EDDIES**. **INTERNAL** causes CANARY to use the internal computer clock and should be used for most **BATCH** or **REALTIME** runs. **EDDIES** indicates that the EDDIES software controls CANARY. **EXTERNAL** is a custom control included for backwards compatibility.
    - *control messenger:* – This parameter defines a particular data sources for the **EDDIES** or **EXTERNAL** messenger service. For the **INTERNAL** control type, this line should be omitted or left as **null**.

- *use continue:* – This parameter specifies whether a restart file, "continue.edsd," should be used to pre-populate the data and configuration for analysis. The options are **yes** or **no.** If this line is omitted, then CANARY will ignore the restart file. This option is typically only used for **REALTIME** runs. This is a new feature for this version.
- *data provided:* – This parameter modifies how all data sources operate. The options are **NEW VALUES** or **ALL VALUES.** The default, **ALL VALUES,** means a real value for every tag at every time step is provided and any missing values are consider as a bad sensor; this is CANARY's traditional operating mode. The optional **NEW VALUES** mode assumes unreported values during a time step are unchanged from the previous time step. This option is for delta-only SCADA historian systems, and is new in this version.
- *driver files:* – This parameter defines a list of extra driver files needed for a run. Typically, these are database JDBC2 driver JAR files, but this is also the place to list the JAR files containing custom algorithms. Each entry must be preceded by a minus character (**-**) and be indented equally.

  Each list entry is composed of:

  - The filename of the JAR file to include.

5.1.1 Example C1: Batch mode controls

This example shows the control section of a YAML configuration file for a **BATCH** CANARY run on historical data.

Table 5. Basic control setup for a historical run with no extra driver files needed.

YAML Configuration of Control	Comments
canary:	# ex. C1
run mode: **BATCH**	
control type: **INTERNAL**	
control messenger: **null**	# C1-1

1. The *control messenger* value is set to the reserved word **null** to tell the YAML parser that there is no value for this parameter.

5.1.2 Example C2: Real-time mode controls

This example shows the control section of a YAML configuration file for a **REALTIME** run with driver files.

Table 6. A control configuration for a REALTIME run that requires two driver files.

YAML Configuration of Control	Comments
canary:	# ex. C2
run mode: **REALTIME**	
control type: **INTERNAL**	
control messenger: **null**	
use continue: **no**	
data provided: **ALL VALUES**	
driver files:	# C2-1
- C:\jdbcDrivers\SQLServer\sqljdbc4.jar	# C2-2
- C:\Program Files\CANARY\lib\MyAlgorithms.jar	# C2-3

1. The *driver files* parameter signals the start of a list, so nothing should follow after the colon (:) except comments.
2. This is a JAR file to drive a Microsoft® SQL Server database.
3. This is a JAR file containing a custom algorithm.

5.2 Timing Options Section

This section of the configuration file determines the timing options. It specifies the time/date stamp of the input water quality and operational data and the spacing and interval of the data to be analyzed.

- *timing options:* – This starts the timing options section.
    - *dynamic start-stop:* – This parameter tells CANARY to use the current date and time or a specific date and time for the start and stop date. The options are **on** or **off**. **BATCH** and **EDDIES** run modes usually use a value of **off**, while **REALTIME** mode typically uses a value of **on**.
    - *date-time format:* – This parameter defines the format for dates and times used by CANARY. The database formats must be defined to match the format listed here.
        - The value for this parameter is a quotes encased string that contains a combination of the following character codes:
            - **mm** – the code for a two-digit month, such as “02” for February.
            - **mmm** – the code for a three-letter month, such as “Feb” for February.
            - **dd** – the code for a two-digit day, such as “03” for the third day of the month.
            - **yy** – the code for a two digit year, such as “08” for 2008.

- **yyyy** – the code for a four digit year, such as “2008.”
- **HH** – the code for the hour of the day; 12- or 24-hour format is decided based on the presence of an AM code later on.
- **AM** – if this code is present, AM or PM is printed, and the HH code outputs 12-hour times.
- **MM** – the code for the minutes of the hour.
- **SS** – the code for the seconds of the minute.
- Most other characters, such as **-**, **:**, **/** or **T** are used verbatim as delimiters.

▪ Some common date/time format strings:

- ODBC database canonical form – **yyyy-mm-dd HH:MM:SS**
- ISO standard – **yyyymmddTHHMMSS**
- US standard – **mm/dd/yyyy HH:MM AM**
- European standard – **dd/mm/yyyy HH:MM:SS**

o *date-time start:* – This parameter defines the first data point to include in the analysis. This value should be in the format specified in the date-time format.

o *date-time stop:* – This parameter defines the last data point to include in the analysis. This value should be in the format specified in the date-time format.

o *data interval:* – This parameter defines the amount of time between two data points read by CANARY. This determines the bin size used in the event detection algorithms. This value must be in the format HH:MM:SS, for example, **00:05:00** for a five minute interval between data points.

o *message interval:* – This parameter defines the amount of time between clock checks in real-time. This is essentially a sleep interval, during which CANARY does not use the processors and waits for the next data time step to occur. The value must be in the HH:MM:SS format. It should generally be smaller than the *data interval*.

5.2.1 Example T1: Timing configuration with start- and stop-dates specified

This example shows the timing options section of the configuration file using an offline run of a month and a half of two-minute data.

Table 7. A *timing options* section that is configured for an offline (BATCH) run.

YAML Configuration of Timing Options	Comments
timing options:	# Ex. T1
dynamic start-stop: **off**	# T1-1
date-time format: **mm/dd/yyyy HH:MM AM**	# T1-2
date-time start: **03/01/2000 12:00 AM**	
date-time stop: **04/15/2000 11:58 PM**	
data interval: **00:02:00**	
message interval: **00:00:01**	# T1-3

1. For a historical data run in **BATCH** mode, the dates have to be specified in the *date-time start* and the *date-time stop* parameters.
2. This example uses a standard date-time format for spreadsheets. Remember, however, that spreadsheet programs may change formatting without changing the representation in the file!
3. The message interval can be set to 1-second, since in **BATCH** mode, there is no need to wait for new data.

5.2.2 Example T2: Timing configuration with dynamic dates for a real-time run

This example shows the timing options section of a configuration file for an online run with current, new data.

Table 8. A *timing options* section that is configured for an online (REALTIME) run.

YAML Configuration of Timing Options	Comments
timing options:	# Ex. T2
dynamic start-stop: **on**	# T2-1
date-time format: **yyyy-mm-dd HH:MM:SS**	# T2-2
data interval: **00:02:00**	
message interval: **00:30:00**	# T2-3

1. For a real-time run, set *dynamic start-stop* to **on** instead of setting the start- and stop-dates every time.
2. The ODBC database canonical format is specified here. This is important, since every database has an equivalent format; using a custom format might not be possible in every database.
3. Because this is a **REALTIME** run, the message interval is set to 30-seconds to avoid slowing down the computer while CANARY waits for new data.

5.3 Inputs and Outputs – Data Sources Section

The inputs and outputs used by CANARY are defined in the *data sources* list. The list contains the configuration instructions for inputs, outputs, or inputs/outputs. *Data*

sources can only be specified one time per configuration file – some of the examples below have that line while others do not.

- *`data sources:`* – This starts the *data sources* section. Below this entry, the data sources used for input and/or outputs to CANARY are defined. This section is a list, and each entry should start with a single minus sign (**`-`**) and all the entries should then have the same indentation.

  Each list entry is composed of:

  - *`id:`* – This parameter is a text string and defines an internal identifier within CANARY. This value can be named anything, and is case sensitive, however, no spaces can be included in the name.
  - *`type:`* – This parameter defines the type of data source. The options are **`CSV`**, **`DB`** (or **`JDBC`**), **`EDDIES`,** and **`XML`**. See Section 3 for more details.
  - *`location:`* – This parameter is the directory path or a URL to the data source file.
  - *`enabled:`* – This parameter activates or deactivates the entire data source. The options are **`yes`** or **`no`**. This option is useful when multiple data sources are defined within the same configuration file.
  - *`configFile:`* – This parameter is only used to keep consistency with the older format (XML) configuration files. The database options section below has replaced this file.
  - *`timestep options:`* – This parameter starts the time step options subsection.
    - *`field:`* – This parameter specifies the column name header that the date/time information can be found in the data source file. It can be a column header in a CSV file or the field name in a database table.
    - *`format:`* – This parameter specifies the data/time format. It can be left blank and the format found in the global timing options section would be used. For databases, this must be specified using the database's format for the associated date-to-string conversion function, and it must match the equivalent format specified under the global timing options.
    - *`conversion function:`* – This parameter is the function needed to convert strings to dates for databases.
  - *`database options:`* – This parameter starts the database options subsection. This entire section can be omitted when using a **CSV** type data source.
    - *`time drift:`* – This parameter specifies the time delay between the computer (server) system clock and the database system clock. This number can be positive or negative. For example, Server Time

+ Drift = Database Time or Drift = DB – Server. The value of time drift needs to be entered as a real number (float) and the value is in fractional days (e.g., 0.75 = 18 hours or 3/4th of a day). If it is left blank or omitted, and **REALTIME** mode was selected, then CANARY tries to calculate the delay by checking against the last entry in the input table. This does not always work, and if dropped data is seen every few time steps, an error here is likely the cause.

- *JDBC2 class name:* – This parameter names the Java class inside the driver. It should always be one of the "Data sources" typed classes, such as**com.mysql.jdbc.jdbc2.optional.MysqlDataSource** or **oracle.jdbc.pool.OracleDataSource**. These are JDBC2-style classes. Please talk with your database support person, as they know what driver to use. Many systems now include these with the database, it is just a matter of finding the right file in the right directory.
- *input table:* – This parameter specifies the table or view inside the database from which the input data originates. The format of this table depends on the value provided in the input format field. The exact names of the entries can be specified using the following input fields entry.
- *input format:* – This parameter specifies the format of the input data. If omitted, the default format (spreadsheet-like) will be used. Valid options for this field are **default**, **row-based**, or **custom**. These values are case sensitive.
- *input fields:* – This parameter starts the input field subsection, in which customized database table field names can be defined.
    - *time-step:* – This parameter defines the field in which the time stamp of the new data is listed. *NOTE* if this is unspecified, the default value from the time-step options entry of the data sources will be used instead.
    - *parameter tag:* – This parameter defines the field in which the SCADA tag is listed.
    - *parameter value:* – This parameter defines the field in which the value of the SCADA tag is listed.
    - *parameter quality:* – This parameter defines the field in which a quality flag may be listed. This must be a string that is either "**Normal**" or "**Bad**" (case insensitive).
- *output table:* – This parameter specifies the table inside the database into which the CANARY results will be placed. The output

table must be formatted as described in Chapter 4 or as described using the output format entry.

- *output format:* – This parameter specifies the format of the output data. If omitted, the default format of the database table (see Section 4) will be used. The accepted values are **default**, **extended,** or **custom**. These values are case sensitive.
- *output fields:* – This parameter starts the output fields subsection, in which customized field names can be defined. To omit a field, use **no** or **false** as the value, or omit it altogether. Otherwise, the value should contain the database table field name where the associated value will be saved. If an output format of **custom** was used, there are no default field names provided for any fields! If **default** or **extended** are used, then the values provided below will override the default field names; however, extra fields cannot be defined. The user must use the **custom** format to change which fields are populated.
    - *write conditions:* – This parameter is an optional string that limits new rows. The default and currently only available option is **all**, which writes output at every time step.
    - *time-step:* – This parameter defines the field for entry date/time values.
      NOTE: this field is ALWAYS used during output. If omitted, the value will be the same as defined in the timestep options of the data source.
    - *instance id:* – This parameter defines the field for the CANARY instance ID that created the values.
    - *station id:* – This parameter defines field for the station definition that provided the result.
    - *algorithm id:* – This parameter defines the field for the algorithm that was used to generate the result.
    - *parameter tag:* – This parameter defines the field for the SCADA tag of the parameter that caused the event (single value). If a field is specified for this entry, a new row will be inserted for each contributing parameter. If a single string of all contributing parameters is desired, use the contributing parameters entry instead.
    - *parameter residual:* – This parameter defines the field for the residual of the parameter in parameter tag. It is always omitted unless parameter tag has been defined.

- *parameter type:* – This parameter defines the field for the parameter type short text. It is only valid if parameter tag has been defined.
- *event code:* – This parameter defines the field for the generated integer event code.
- *event probability:* – This parameter defines the field for the probability of an event generated.
- *contributing parameters:* – This parameter defines the field for a whitespace delimited list of contributing parameters.
- *comments:* – This parameter defines the field for the analysis comments.
- *pattern match id:* – This parameter defines the field for the pattern with the highest match probability.
- *pattern match probability:* – This parameter defines the field for the probability of highest match pattern.
- *secondary match id:* – This parameter defines the field for the pattern with the 2nd highest match probability.
- *secondary match probability:* – This parameter defines the field for the probability of 2nd highest match pattern.
- *tertiary match id:* – This parameter defines the field for the pattern with the 3rd highest match probability.
- *tertiary match probability:* – This parameter defines the field for the probability of 3rd highest match pattern.

▪ *login info:* – This parameter starts the login options subsection.

- *prompt for login:* – This parameter specifies that the login process is interactive. The options are **yes** or **no**. If **yes**, CANARY will request a user's name and password. If **no**, the following two lines will be used.
- *username:* – This parameter specifies the username for database access.
- *password:* – This parameter specifies the password for database access.

5.3.1 Example IO1: A *data sources* section using a CSV file as the input.

This example shows the data sources section of a configuration file using **BATCH** mode. In this case, only a single input CSV file is used and there are no special output data sources.

Table 9. Configuration for *data sources* of *type:* CSV.

YAML Configuration of Data sources	Comments
data sources:	# IO1-1
- *id*: **EXMPL_CSV_IN**	# Ex. IO1
type: **CSV**	
location: **Station_B_Train.csv**	# IO1-2
enabled: **yes**	
timestep options:	
field: **TIME_STEP**	

1. Only include *data sources* once per file!
2. The *location* is one of the following for a **CSV** type input:
    a. A complete, fully qualified path and filename.
    b. A path and filename relative to the location of the configuration file.
    c. A filename of a file in the same directory as the configuration file.

5.3.2 Example IO2: A *data sources* list entry for a database with default formats.

This example shows the data source section of a configuration file using a default database input and output. In this case, the default input format means that the table is in a spreadsheet-like format. The lack of a *data sources* parameter means that one would have to be added or already exists in the file.

The IO2 example uses a SQL Server database on the local machine. The appropriate driver file would need to be added to the *drivers* list in the *canary* control section. The JDBC2 class should always have the text “DataSource” as part of the class name, otherwise it is probably the wrong class and CANARY will fail.

Table 10. Configuration for *data sources* of *type:* JDBC with *input format:* default and *output format:* default.

YAML Configuration of Data sources	Comments
- *id*: **EXMPL_DB_DEFAULT_IN_OUT**	# Ex. IO2
type: **JDBC**	
location: **jdbc:sqlserver://localhost;database=scada4**	# IO2-1
enabled: **yes**	
timestep options:	
field: **TIME_STEP**	
conversion function: **"CONVERT(datetime,"**	# IO2-2

YAML Configuration of Data sources	Comments
conversion format: **"120"**	# IO2-3
database options:	
time drift: **0.00**	# IO2-4
JDBC2 class name: **com.microsoft.sqlserver.jdbc.SQLServerDataSource**	
input table: **"[CANARY_TESTNG].[TEST_INPUT1]"**	
input format: **default**	
output table: **"[CANARY_TESTING].[TEST_OUTPUT1]"**	
output format: **default**	
login info:	
prompt for login: **no**	# IO2-5
username: **CANARY_TEST**	
password: **CANARY_TEST_PASS**	

1. The *location* parameter specifies a URL for a database. The exact format of the URL will be different for each database vendor, but it will always start with “jdbc:” and have a “//” in it somewhere.
2. Occasionally, the beginning of the conversion function may require a ‘-mark in the text. In this case, the entire string must be enclosed in double-quotes. Make sure these are the simple ‘-characters and “-characters, and not the alternating up-down quotes that word processors automatically change “-characters into.
3. Two items worth mentioning:
    a. It is important for this to be a string and not a number. If the format codes are numeric, such as the SQL server format codes, they must be enclosed inside quotes.
    b. All the formats for databases used in the examples use the ODBC canonical format. This means that the date and time always look like: “2011-11-21 01:13:14”
4. Remember that the time drift is specified in fractional days.
5. NOTE: always make sure to protect passwords! They are clear text in this file, so if a username and password are added, make sure to protect the file from others!

5.3.3 Example IO3: A *data sources* list entry for a database with row-based and extended formats.

This example shows a data source section of a configuration file using a **row based** input table and an **extended** output table. These are the only differences between examples IO3 and IO2.

Table 11. Configuration for *data sources* of *type:* JDBC with *input format:* row based and *output format:* extended.

YAML Configuration of Data sources	Comments
- *id*: **EXMPLE_DB_EXTND_IN_OUT**	# Ex. IO3
type: **JDBC**	
location: **jdbc:sqlserver://localhost;database=scada4**	
enabled: **yes**	
timestep options:	
field: **TIME_STEP**	
conversion function: **"CONVERT(datetime,"**	
conversion format: **"120"**	
database options:	
time drift: **0.00**	
JDBC2 class name: **com.microsoft.sqlserver.jdbc.SQLServerDataSource**	
input table: **[CANARY_TESTNG].[TEST_INPUT2_RB]**	
input format: **row based**	# IO3-1
output table: **[CANARY_TESTING].[TEST_OUTPUT2_EXTND]**	
output format: **extended**	# IO3-2
login info:	
prompt for login: **no**	
username: **CANARY_TEST**	
password: **CANARY_TEST_PASS**	

1. The *input format* of **row based** with no *input fields* means that the default field names is expected in the *input table*.
2. The *output format* of **extended** with no *input fields* means that the default field names is expected in the *output table*.

5.3.4 Example IO4: A *data sources* list entry for a database with some customized field names.

This example shows a data source section of a configuration file using customized, user-specified, field names for the **row based** and **extended** formats. This is a customized version of example IO3. The only other change is that the login information is not provided in the configuration file; CANARY should ask for that information when the program starts. This option might be desired if a multi-user machine is being used, or if it is required by the local security policies.

Table 12. Configuration for *data sources* of *type:* JDBC with *input format:* row based with customized field names and *output format:* default with customized field names.

YAML Configuration of Data sources	Comments
- *id*: **EXMPLE_DB_EXTND_IN_OUT_CUSTOM**	# Exn IO4
type: **JDBC**	
location: **jdbc:sqlserver://localhost;database=scada4**	
enabled: **yes**	

YAML Configuration of Data sources	Comments
timestep options:	
field: **TIME_STEP**	
conversion function: **"CONVERT(datetime,"**	
conversion format: **"120"**	
database options:	
time drift: **0.00**	
JDBC2 class name: **com.microsoft.sqlserver.jdbc.SQLServerDataSource**	
input table: **[CANARY_TESTNG].[TEST_INPUT3_RB]**	
input format: **row based**	
input fields:	
time-step: **TIME_STEP**	# IO4-1
parameter tag: **SCADA_TAG**	
parameter value: **RAW_VALUE**	
output table: **[CANARY_TESTING].[TEST_OUTPUT3_EXTND]**	
output format: **default**	
output fields:	
time-step: **TIME_STEP**	# IO4-2
instance id: **EDS_NAME**	
station id: **STN_CFG_ID**	
event code: **STATUS_CODE**	
event probability: **EVT_PROB**	
contributing parameters: **PARAM_LIST**	# IO4-3
login info:	
prompt for login: **yes**	

1. Although the *time-step* parameter is listed here, because the default value of "**TIME_STEP**" was used, it could have been omitted.
2. As noted above, the *time-step* parameter could have been omitted.
3. Notice that the *comments* parameter is missing from the *output fields*. Because *output format:* **default** was used instead of *output format:* **custom**, a field named "**COMMENTS**" is still expected to be defined in the output table; to be clear - the *comments* parameter is not required, but the database field should still be there! This may prove confusing for end users, so it is always advisable to include all field names if any are customized, just for clarity's sake.

5.3.5 Example IO5: A complete *data sources* section, with three entries – two input tables and one output table.

This example shows a data source section of a configuration file using two inputs sources (one for water quality data, one for operational data) and a single output to a different database. Both input data sources indicate that an unprivileged account is being used to read the data. The output data source uses an interactive login since CANARY would need an account with write privileges to output to the database.

This example also shows the differences between a SQL server data source configuration (as in examples IO2 through IO4) and a MySQL or Oracle database

configuration. These drivers would also need to be added to the *drivers* list in the *canary* control section.

As these sections of the configuration file can be long, be sure to follow the formatting rules for YAML (see Appendix A). Some quick tips: (1) blank lines can be added between the *data sources* list entries, (2) comments on separate lines should start in the same column as the list entries (align the # with the - character), (3) and lists should be aligned starting with – symbols, (4) do not use tab characters but use spaces instead!.

Table 13. Complete configuration for a three-database system, reading from two different input databases and outputting to a third.

YAML Configuration of Data sources	Comments
data sources:	# Ex. IO5
- *id*: **EXMPL_DB_INPUT1**	# IO5-1
type: **JDBC**	
location: **jdbc:mysql://wq-scada:3306/water_quality**	# IO5-1a
enabled: **yes**	
timestep options:	
field: **TIME_STEP**	
conversion function: **"str_to_date"**	# IO5-1b
conversion format: **"%Y-%m-%d %H:%i:%s"**	# IO5-1b
database options:	
time drift: **0.0020833**	# IO5-2
JDBC2 class name: **com.mysql.jdbc.jdbc2.optional.MysqlDataSource**#1c	
input table: **samples**	# IO5-1d
input format: **row based**	
input fields:	
time-step: **TIME_STEP**	
parameter tag: **SCADA_TAG**	
parameter value: **RAW_VALUE**	
login info:	# IO5-3
username: **CANARY_READ_ONLY**	# IO5-4
password: **C4n4rY**	
- *id*: **EXMPL_DB_INPUT2**	# IO5-5
type: **JDBC**	
location: **jdbc:oracle:thin:@//192.168.11.293:1521/orcl**	# IO5-5
enabled: **yes**	
timestep options:	
field: **TIME_STEP**	
conversion function: **"To_Date"**	# IO5-5
conversion format: **"YYYY-MM-DD HH24:MI:SS"**	# IO5-5
database options:	
time drift: **0.0020833**	
JDBC2 class name: **oracle.jdbc.pool.OracleDataSource**	# IO5-5
input table: **"OPERATIONS"."RAW_SAMPLES"**	# IO5-5
input format: **custom**	
input fields:	
time-step: **SAMPLE_TIME**	

YAML Configuration of Data sources	Comments
parameter tag: **TAG**	
parameter value: **VALUE**	
parameter quality: **QUALITY**	# IO5-6
login info:	
username: **CANARY_LIMITED**	
password: **C4n4rY**	
- *id*: **EXMPL_DB_OUTPUT**	# IO5-7
type: **JDBC**	
location: **jdbc:sqlserver://localhost;database=scada4**	
enabled: **yes**	
timestep options:	
field: **TIME_STEP**	
conversion function: **"CONVERT(datetime,"**	
conversion format: **"120"**	
database options:	
time drift: **0.00**	
JDBC2 class name: **com.microsoft.sqlserver.jdbc.SQLServerDataSource**	
output table: **[DASHBOARD].[EDS_OUTPUTS]**	# IO5-8
output format: **custom**	# IO5-9
output fields:	
time-step: **TIME_STEP**	
station id: **STN_CFG_ID**	
algorithm id: **ALG_CFG_ID**	
event code: **STATUS_CODE**	
event probability: **EVT_PROB**	
contributing parameters: **PARAM_LIST**	# IO5-9
login info:	
prompt for login: **yes**	# IO5-10

1. The first input database in this example is a MySQL database. Remember to add the JAR file for the driver to the *drivers* list in the control section! Some of the differences are:
    a. The *location* URL
    b. The *conversion function* and *conversion format*
    c. The *JDBC2 class name*
    d. The table names (notice the style difference from the other examples)
2. This *time drift* means that the computer running CANARY is approximately three (3) minutes faster than the database. In reality, they might be exactly synchronized, but by adding in a 3-minute delay, there is less chance of missing data due to database write latency. It is advisable to always delay by one time-step, just to be safe.
3. This data sources section does not contain an *output table* parameter. This means that it cannot be used for writing results. It might be desirable to separate

input and output databases this way for security reasons, or simply because there are multiple databases to be used.

4. The username listed here gives the impression that the account has no write privileges. This is a good way to ensure that the CANARY database user cannot accidentally do anything it should not do (and makes the read-only usernames and passwords slightly less sensitive).
5. The second input database is an example Oracle DB database. Notice the differences in the same places as in comment 1.
6. This data source uses a **custom** input format so that the *parameter quality* parameter can be used.
7. The output database is the SQL server example used previously.
8. Note the lack of *input table*, *input format*, or *input fields* parameters. This means that the data source can only be used for output.
9. While the *output fields* are identical to the parameters listed in example IO4, notice that the *output format* is now **custom**. This means that the *comments* parameter is not used, whereas it was implied in example IO4.
10. Because this is a more privileged account (it needs to write to a table), the example requires that the user enter their login information every time.

5.4 Signals Definition Section

This section of the configuration file starts with *signals*, and describes the associated parameters. This section is used to designate which signals of different types to be used in the CANARY analysis. A signal is an incoming water quality, operational, or alarm value measured by a sensor. Signals can also be combinations of data values constructed as composite signals.

- *signals:* – This starts the signals section. Below this entry, each signal used in a CANARY analysis is defined. This section is a list type entry, and each new signal must start with a minus sign (**-**) character and all the entries should then have the same indentation.

  Each list entry is composed of:

  - *id:* – This parameter is a string that serves as the internal CANARY identifier for an individual signal. This parameter is case sensitive and must start with a letter. It cannot include spaces or symbols. The signal id cannot be a subset of the beginning characters of another signal name. For example, if there is an existing signal id of **TEST_CL_STATION1** then **TEST_CL** is not a permissible name for another signal, since it is a subset of an existing name. However, **CL_STATION1** is permissible.
  - *SCADA tag:* – This parameter specifies the signal name as given in the CSV file or database. It must exactly match the CSV column header

name, the database field name, or the database table value. This parameter connects CANARY with the specific location of the signal data within the data source.

- *evaluation type:* – This parameter defines the type of data signal. The options are **WQ** (water-quality), **OP** (operations), **ALM** (data alarm), and **CAL** (calibration). Only signals defined as **WQ** are used to detect events. Only one **CAL** signal can be used by each station. To use multiple calibration signals, define the signals as **OP** signals, then combine them into a single **CAL** signal using a composite signal.
- *parameter type:* – This parameter specifies the signal description short text, such as **PH** for pH or **CL2** for free chlorine value. This text is used as the axis label for CANARY plots. However, when **EDDIES** mode is used, this parameter must be consistent with the parameter definitions within EDDIES.
- *ignore changes:* – This parameter specifies if the data signal should be ignored in specific instances. The options are: **all**, **increases**, **decreases**, **both**, or **none**. This parameter should be left as **none** for any water quality signal, unless the user has advanced experience with CANARY. If this parameter is omitted, the default of **none** is used.
    - For most signals, and by default, this field should be **none** or be omitted.
    - For operations signals, however, this field has additional options: does a change in this operations type signal mean that the algorithm should do a short-term recalibration? For example, when a pump comes online, it might be necessary to allow a short time period for water quality parameters to re-stabilize. Setting this field to **increases**, **decreases**, or **both** activates this feature for operations type signals. If one of the activating values is selected, the trigger threshold is based on the slope of the data across the past five time steps. If the slope is greater than or less than the negative of one unit of precision (see *precision*), then no alarms occur because the operations signal has triggered a short-term recalibration
- *data options:* – This parameter starts the data type subsection for a specific signal. It is needed for all water quality (**WQ**) and operational (**OP**) signal types.
    - *precision:* – This parameter specifies a noise threshold for the signal. Signal changes less than this value are ignored by CANARY. The precision value is typically set based on sensor precision information from the sensor manufacturer. It can be adjusted to better manage very noisy or extremely constant signal values.

- *units:* – This parameter specifies the units that are used to label CANARY plots. CANARY is able to recognize LaTeX character strings.
- *valid range:* – This parameter specifies the physical valid range for the data. It is a two-value vector defined with brackets and separated by a comma. Any data outside this range is treated as having originated from a malfunctioning sensor. This entry also controls the minimum and maximum value range on the Y-axis of the CANARY water quality plots. The default value is **[-.inf, .inf]**, which corresponds to –infinity and + infinity and places no limits on a valid data value. An example valid range for chlorine residuals is **[0.0, 2.5]**, as measured in mg/L.
- *set points:* – This parameter specifies the set points for the signal. It is defined the same way as *valid range*. These values are only used for event detection, if a set point algorithm is defined. If either set point value is outside of the *valid range*, then a set point alarm never occurs.

o *alarm options:* – This parameter starts the alarm type subsection for a specific signal. It is needed for alarm (**ALM**) and calibration (**CAL**) signal types. It describes the sensor diagnostic alarms. Alarm type signals are defined similarly to water quality and operations signals. A calibration signal (**CAL**) applies to an entire monitoring station, while an alarm signal (**ALM**) applies to a single signal.

- *value when active:* – This parameter specifies the diagnostic alarm value. The options are either **1** or **0**. The majority of utilities define the normal (operating) state as **0** and the alarm/calibration state as **1**, but the opposite situation can also be handled also.
- *scope:* – This parameter specifies which signal (of **WQ** or **OP** type) the alarm signal applies. If the current signal type is **CAL**, then this is left blank. If the signal type is **ALM**, then the scope value is the *SCADA Tag* of the signal producing the alarm

o *composite rules:* – This parameter defines a composite signal. It should be followed by a pipe (**|**) character, then indented text. Each line is a command executed in Reverse Polish Notation (RPN) – like the old HP calculators. An operation acts on the preceding entries. For example, the standard “4 + 5” would instead be “4 5 +” in RPN. Once a command is reached, such as the “+” sign in the example, the command entry containing the result is used by the next operation. For example, “(4 + 5) / 2” would be written as “4 5 + 2 /,” while “2 / (4 + 5)” would be written as “2 4 5 + /.” NOTE: a composite signal has a limit of around 10-15 operations per signal. To use more than this number of operations, the calculation needs to be broken into multiple signals.

Each line of the rules has one of the following formats:

- **@SIGNAL_ID[SHIFT]** – The @ symbol indicates that this is a name of a signal. The **SIGNAL_ID** is the id of a signal that is defined elsewhere. A number between square brackets, such as **[1],** defines a shift in time backwards an integer number of time steps. Some examples:
  - **@STATION1_CL2_VAL[3]**
  - **@P_0x0002481[0]**
- **(CONSTANT)** – A number between two parentheses is used as a constant. For example:
  - **(0)**
  - **(-2.3)**
  - **(99.922)**
- Any other text is interpreted as a command. The valid commands are:
  - **+** plus
  - **-** minus
  - ***** multiply
  - **/** divide
  - ****** or **^** exponentiation
  - **exp** for the natural number raised to a power (use a constant or signal on the line prior)
  - **log10** base-10 logarithm
  - **log** natural logarithm
  - **sin** sine (radians)
  - **cos** cosine (radians)
  - **tan** tangent (radians)
  - **abs** absolute value
  - **sqrt** square root
  - **>**, **<**, **>=**, **<=**, **==,** and **!=** logical tests
  - **gt**, **lt**, **ge**, **le**, **eq,** and **ne** equivalent logical tests
  - **max** maximum of two values

- **min** minimum of two values

5.4.1 Example S1: A water quality signal for a chlorine sensor.

This example shows a signal section of a configuration file using a water quality signal (in this case chlorine concentration) with only the minimum entries. The defaults that are not shown are an infinite range of valid values, no set points, and no data ignored.

Table 14. Configuration for a single WQ type signal; includes the *signals* key.

YAML Configuration of Signals	Comments
signals:	# S1-1
- *id*: **SIG_S1_CL2**	# Ex. S1
SCADA tag: **LOCAA_WQ_CL2_VAL**	# S1-2
evaluation type: **WQ**	# S1-3
parameter type: **CL2**	
data options:	
precision: **0.01**	# S1-4
units: **mg/L**	

1. The *signals* key can only be specified once per file. If there is a failure on startup reading the configuration file, it is likely that bad indentation or a repeated main key, like *signals* is to blame.
2. The *SCADA tag* is the string that uniquely identifies a particular sensor within the SCADA system. These strings do not have a standard format, and the user needs to talk with the SCADA database administrator to identify the tag names.
3. An *evaluation type* of **WQ** means that the signal is included in the event detection algorithms at a particular station. Nothing in CANARY limits **WQ** type signals to water chemistry signals. Pressure, valve status, or any measurable quantity could be labeled as a **WQ** type signal and CANARY would use that signal to determine if an event has occurred. Pressure specifically might be a useful **WQ** type signal if the user wants CANARY to look for main breaks or pressure spikes that would change pressures, but not enough to cross set-point limits.

5.4.2 Example S2: A water quality signal for a pH sensor, with set points and a valid range.

This example shows the signal section of a configuration file using a water quality signal with all available parameters defined. A pH sensor in particular should have a valid range, since pH is limited by definition to values between 0 and 14. One could even use pH as a surrogate for bad data quality, creating calibration signal like the one in example S5 that would turn on calibration mode if the pH sensor gave a reading of 0.0, since this would mean that something has gone very wrong with the sensor station.

Table 15. Configuration for a single WQ type signal with set points and a valid range.

YAML Configuration of Signals	Comments
- *id*: **SIG_S2_PH**	# Ex. S2
SCADA tag: **LOCAA_WQ_PHX_VAL**	
evaluation type: **WQ**	
parameter type: **PH**	# S2-1
ignore changes: **none**	
data options:	
precision: **0.1**	# S2-2
units: **pH**	
valid range: **[0.1, 13.9]**	# S2-3
set points: **[-.inf, 9]**	# S2-4

1. The *parameter type* string is mostly used for information purposes only. It is used in the "contributing parameters" output string, or in the "parameter type" output string that is optionally available. It should be short, probably an abbreviation, but any string less than 10 characters should be okay.
2. The *precision* value determines the minimum change necessary before a change can possibly be an outlier. It may be necessary to play with these slightly to find the right balance of sensitivity.
3. The *valid range* is usually written as two values separate by a comma surrounded by square brackets; however, it can be specified as a multi-line list similar to *signals*. To allow any values, use **-.inf** or **.inf** to specify negative infinity or positive infinity, respectively.
4. The *set points* are optional low and/or high set points for a given parameter. To turn one or the other off, use **-.inf** or **.inf** for the value.

5.4.3 Example S3: A hardware alarm for a chlorine sensor.

Many physical sensors might have extra SCADA signals output to indicate sensor malfunction, low supplies, or other problems. This example shows a signal configuration using a hardware alarm associated with the chlorine sensor used in example S1.

Table 16. Configuration for a single ALM type signal associated the WQ signal in example S1.

YAML Configuration of Signals	Comments
- *id*: **SIG_S3_CL2_ALM**	# Ex. S3
SCADA tag: **LOCAA_WQ_CL2_MAINT_REQ**	
evaluation type: **ALM**	
parameter type: **PH_ALM**	
alarm options:	
value when active: **1**	# S3-1
scope: **LOCAA_WQ_CL2_VAL**	# S3-2

1. The *value when active* parameter is very important for **ALM** type signals. If possible, it is much better to have a 1 signify a problem than to use any other value (in this case, 0 would be the “sensor ok” signal).
2. The *scope* of the alarm is the *SCADA tag* of the data signal and not the ID.

5.4.4 Example S4: A set of operations signals – two normal operations signals and one composite signal.

Operations signals with *evaluation type* **OP** are signals that aid CANARY’s performance, but which are not directly used to identify events. These signals are not necessarily an indicator of plant or distribution system operations (though they often are, and this is where the name comes from). For example, the flow direction at a junction or the status of a tank (draining or filling) might be defined as OP signals and used to eliminate false alarms that accompany these types of operational changes.

While **OP** type signals do not play a direct role in the event detection algorithms, they can be included in pattern libraries. Thus, a drop in chlorine at the same time as a change in flow direction would have a different pattern than just the drop in chlorine alone.

This example shows a signal section of a configuration file using a composite signal that averages pressure. Any type of signal, **WQ**, **OP**, **ALM**, or **CAL** can be a composite signal. The fact that only **OP** and **CAL** type composite signals are shown in the examples should not limit the user in the application of composite signals.

Table 17. Configuration for several OP type signals, including a composite signal made from signals in both examples S4 and S1.

YAML Configuration of Signals	Comments
- *id*: **SIG_S4_PRES**	# Ex. S4A
SCADA tag: **P_0x92871A3E5**	# S4-1
evaluation type: **OP**	
parameter type: **PRES**	
data options:	
precision: **0.01**	
units: **PSI**	
valid range: **[-.inf, .inf]**	# S4-2
set points: **[-.inf, .inf]**	
- *id*: **SIG_S4_VALVE**	# Ex. S4B
SCADA tag: **P_0x2482914**	
evaluation type: **OP**	
parameter type: **VALVE_STAT**	
data options:	
precision: **0.1**	
units: **n/a**	
valid range: **[0,1]**	
- *id*: **SIG_S4_COMPOSITE**	# Ex. S4C
SCADA tag: **SIG_S4_AVE_PRES**	# S4-3

YAML Configuration of Signals	Comments
evaluation type: **OP**	
parameter type: **PRES**	
data options:	
precision: **0.01**	
units: **PSI**	
valid range: **[-.inf, .inf]**	
set points: **[-.inf, .inf]**	
composite rules: **|**	# S4-4
@SIG_S4_PRES[5]	# S4-4a
@SIG_S4_PRES[4]	# S4-4b
+	# S4-4c
@SIG_S4_PRES[3]	# S4-4d
+	# S4-4e
@SIG_S4_PRES[2]	# S4-4f
@SIG_S4_PRES[1]	# S4-4g
@SIG_S4_PRES[0]	# S4-4h
+	# S4-4i
+	# S4-4j
+	# S4-4k
(6.0)	# S4-4l
/	# S4-4m

1. An example of a non-intuitive SCADA tag, and the reason why the *id* parameter is used internally to CANARY.
2. An example of an infinite range.
3. If a signal does not have real data within the SCADA system, as is the case with composite signals, any unique tag is acceptable for use in this field.
4. The composite rules are a string of text. Thus, a pipe character (|) comes after the *composite rules* parameter and the rules are indented below the parameter. A line without indentation ends the text string. For this example, each rule is discussed line by line below.
    a. The first rules entry states “read the data from the signal with id = SIG_S4_PRES at 5 time steps ago, and add it to the stack.” This value is referred to as VAL5, since it is the value from 5 time steps ago.
        i. The stack is now: [VAL5]
    b. The second line retrieves the value from SIG_S4_PRES at 4 time steps ago, and adds it to the stack. The stack now contains two values.
        i. The stack is now: [VAL5, VAL4]
    c. The first command adds the two values on the stack together, and replaces them with the result. This result is referred to as NEW1, since it is the first new addition result.
        i. The values are removed and added: NEW1 = (VAL5 + VAL4)

    ii. The stack becomes [NEW1]

d. The fourth line retrieves and adds the value from SIG_S4_PRES at 3 time steps ago to the stack.
    i. The stack now contains [NEW1, VAL3]

e. The fifth line adds the two values currently on the stack, NEW1 and VAL3, together.
    i. The values are removed and added: NEW2 = (NEW1 + VAL3)
    ii. The stack becomes [NEW2]

f. The sixth line retrieves and adds the value from SIG_S4_PRES at 2 time steps ago to the stack.
    i. The stack becomes [NEW2, VAL2]

g. The seventh line retrieves and adds the value from SIG_S4_PRES at 1 time step ago to the stack.

h. The stack becomes [NEW2, VAL2, VAL1The eighth line retrieves and adds the current value from SIG_S4_PRES to the stack.
    i. The stack becomes [NEW2, VAL2, VAL1, VAL0]

i. The ninth line adds the last two values on the stack together (VAL0 and VAL1) and the result is added back to the stack.
    i. The values are removed and added: NEW3 = (VAL1 + VAL0)
    ii. The stack becomes [NEW2, VAL2, NEW3]

j. The tenth line adds the last two values on the stack together and the result is added back to the stack.
    i. The values are removed and added: NEW4 = (VAL2 + NEW3)
    ii. The stack becomes [NEW2, NEW4]

k. The eleventh line adds the last two values on the stack together.
    i. The values are removed and added: NEW5 = (NEW2 + NEW4)
    ii. The stack becomes [NEW5]

l. The twelfth line adds a constant value of 6.0 to the stack.
    i. The stack becomes [NEW5, 6.0]

m. The thirteenth line divides the two values on the stack and the result is added back to the stack.
    i. The values are removed and divided: NEW6 = (NEW5 / 6.0)
    ii. The stack becomes [NEW6]

Since no more rules are provided, the last value on the stack becomes the current value of the signal.

i. Current value of signal: NEW6

ii. The complete formula for this rule set can be written as: (((VAL5+VAL4)+VAL3)+(VAL2+(VAL1+VAL0))) / 6.0

The number of operations that can be performed in a single signal is limited to about 12, because of the limitations on the number of parentheses that can be used internally in the code. If a composite signal is failing for seemingly no apparent reason, try splitting it into two different composite signals to reduce the number of operations.

5.4.5 Example S5: Two calibration signals, a direct SCADA signal and a composite signal.

A monitoring station, discussed in Section 5.6, can only have one calibration signal. If more than one is needed, the user needs to create a composite signal to combine the other signals. Signals of type **CAL** or **ALM** should not be used as input to composite signals.

This example shows a signals section of a configuration file using a hardware calibration signal and a composite signal, which defines a calibration mode if the pressure drops below 10.0 psi.

Table 18. Configuration for two CAL signals, a direct hardware signal and a composite signal.

YAML Configuration of Signals	Comments
- *id*: **SIG_S5_CAL_HW**	# Ex. S5
SCADA tag: **LOC_AA_CAL_SWITCH**	
evaluation type: **CAL**	
parameter type: **CAL**	
alarm options:	
value when active: **1**	# S5-1
- *id*: **SIG_S5_CAL_COMP**	
SCADA tag: **SIG_S5_CAL_COMP**	
evaluation type: **CAL**	
parameter type: **CAL**	
alarm options:	
value when active: **1**	
composite rules: **|**	# S5-2
@SIG_S4_COMPOSITE	
(10.0)	
<	

1. A calibration signal does not need a *scope* parameter under the *alarm options* section.
2. The composite rules for this signal say “if the current value of SIG_S4_COMPOSITE (the average pressure defined in example S4) is less than 10.0, then set the value to 1 (true) else set the value to 0 (false).

Logical operations return an integer value of 1 for true and 0 for false when used in the composite rules. These can then be used as real values in other signals.

5.5 Algorithms Definition Section

This section of the configuration file defines the event detection algorithms that can be used for analysis. In the *monitoring stations* section, the algorithms defined here are assigned to a group of signals.

- *`algorithms:`* – This starts the algorithms section. Below this entry, each algorithm used in a CANARY analysis is defined.

  Each list entry is composed of:

  - *`id:`* – This parameter defines an unique algorithm name to use in the analysis. It is a text string with no spaces. Multiple algorithms can be defined within a single configuration file.
  - *`type:`* – This parameter specifies the event detection algorithm to use in the analysis. The options include **`LPCF`**, **`MVNN`**, **`SPPE`**, **`SPPB`**, **`JAVA`**, **`CAVE`**, or **`CMAX`**. For more details on the different algorithms, see Section 2.4.2.
  - *`history window:`* – This parameter specifies the number of prior time steps of water quality data used to predict the next water quality value. As a general rule of thumb, the value of history window should be long enough to include 1.5 to 2.0 days of previous data. Note: the value of history window applies equally to all signals that are input to this algorithm. For more details, see Section 2.4.
  - *`outlier threshold:`* – This parameter specifies the number of standard deviations of prediction error that must be met or exceeded before an outlier is declared. For more details, see Section 2.4.
  - *`event threshold:`* – This parameter specifies the probability of an event (P(event)) threshold that must be exceed before a series of outliers are declared an event.
  - *`event timeout:`* – This parameter specifies the number of consecutive time steps after an event is found before the alarm is silenced automatically.
  - *`event window save:`* – This parameter specifies the amount of time to be saved prior to an event being reported. It does not impact the event detection and is only used for plotting the identified events in post processing.
  - *`BED:`* – This parameter starts the BED subsection. It indicates that the Binomial Event Discriminator is being used.

- *window:* – This parameter specifies the size of the binomial window (defined in time steps). It needs to be less than the size of history window and typical values are 4 to 18 time steps.
- *outlier probability:* – This parameter specifies the probability of failure for each binomial trial. It should be left at **0.5**.

o *external algorithm class:* – This parameter defines an external event detection algorithm. It is only used if an external algorithm has been created. It is the class name for the algorithm being called.

o *external algorithm config:* – This parameter defines external algorithm configuration. It is only used if an external algorithm has been created. The XML configuration for the algorithm should be added here as a text entry in double-quotes.

o *cluster library file:* – This parameter specifies the filename of the clustering data file to use with the algorithm.

o *use algorithm inputs:* – This parameter specifies which algorithms to include in the **CAVE** or **CMAX** consensus algorithms. It is a list of the algorithm ids, each line starting with a (-) character.

Each list entry is composed of:

- The *id* of the algorithm that should be used as input.

5.5.1 Example A1: An LPCF configuration with BED

This example shows a LPCF algorithm configuration using BED. This algorithm is one of the most commonly used, since it provides a good starting point for testing algorithm parameters on historical data.

Table 19. Configuration code for the LPCF algorithm with the BED enabled.

YAML Configuration of Algorithms	Comments
algorithms:	# Ex. A1
- *id*: **ALG_A1**	
type: **LPCF**	
history window: **720**	# A1-1
outlier threshold: **1.0**	
event threshold: **0.90**	
event timeout: **20**	# A1-2
event window save: **20**	
BED:	
window: **15**	# A1-3
outlier probability: **0.5**	

1. A data interval of 2 minutes means that the history window is one day long.

2. The *event timeout* of 20 time steps give the operators 40 minutes to start the investigation process.
3. The BED parameters equate to 12 outliers out of 15 time steps before CANARY declares an event.

5.5.2 Example A2: A set-point algorithm (SPPE)

This example shows a set-point algorithm configuration. The set-point algorithms only operate on signals that have set-points that are within the valid range.

Table 20. Configuration code for a set-point algorithm.

YAML Configuration of Algorithms	Comments
- *id*: **ALG_A2**	# Ex. A2
type: **SPPE**	
history window: **15**	
outlier threshold: **15**	
event threshold: **0.85**	
event timeout: **90**	
event window save: **15**	

5.5.3 Example A3: A consensus algorithm that combines two algorithms (CMAX)

To combine two different algorithms, a consensus algorithm must be used. This example shows a consensus algorithm configuration using the two previously defined algorithms. The **CMAX** algorithm uses the maximum probability value from each algorithm and combines them into a single result.

Table 21. Configuration code for a consensus algorithm.

YAML Configuration of Algorithms	Comments
- *id*: **ALG_A3**	# Ex. A3
type: **CMAX**	
history window: **15**	
outlier threshold: **1.0**	
event threshold: **0.95**	
event timeout: **30**	
event window save: **30**	
use algorithm inputs:	# A3-1
- ALG_A1	
- ALG_A2	

1. The *id* of the input algorithms are provided as *use algorithm inputs* as a list. The list could be exactly as shown or it could be input as **[ALG_A1, ALG_A2]**. It is presented as a multi-line here to maintain consistent format with the other lists

in the examples. This list is not limited to two entries – it could have any number of entries.

5.5.4 Example A4: An example of an LPCF algorithm with a cluster library.

This example shows a LPCF algorithm configuration using a cluster library. It is the same algorithm as example A1, but with the addition of a cluster library. If the algorithm has a cluster library, it should be added at the end.

Table 22. Configuration code of an algorithm with a pattern matching cluster library.

YAML Configuration of Algorithms	Comments
- *id*: **ALG_A4**	# Ex. A4
type: **LPCF**	
history window: **180**	
outlier threshold: **1.0**	
event threshold: **0.90**	
event timeout: **20**	
event window save: **15**	
BED:	
window: **15**	
outlier threshold: **0.5**	
cluster library file: **MyClusters.edsc**	# A4-1

1. The cluster library must have already been created and saved with the filename listed. The same path and filename rules apply here as apply to the *location* parameter in the data source section.

5.5.5 Example A5: An example external Java algorithm.

This example shows an external Java algorithm configuration. It uses the MVNN algorithm as implemented in the default CanarysCore.jar that is included in the CANARY distribution. The algorithm configuration must be included as a string to be passed to the Java class. In this case, the values passed to the Java object are also listed in the appropriate YAML configuration keys; the values in the XML string are the controlling parameters, but the values in the YAML cannot be left completely blank.

Table 23. Configuration code for an example external Java based algorithm.

YAML Configuration of Algorithms	Comments
- *id*: **ALG_A5**	# Ex. A5
type: **JAVA**	
history window: **360**	
outlier threshold: **1.0**	
event threshold: **0.95**	

YAML Configuration of Algorithms	Comments
event timeout: **15**	
event window save: **15**	
external algorithm class: **org.teva.canary.agls.MVNN**	# A5-1
external algorithm configuration: **|**	# A5-2
<edsAlgorithm type="org.teva.canary.algs.MVNN" >	
<history-window>360</history-window>	
<outlier-threshold>1.0</outlier-threshold>	
<event-threshold>0.95</event-threshold>	
<rule-option>LINK_USE_RAW</rule-option>	
<rule-option>ON_OUTLIER_KEEP_LAST</rule-option>	
<rule-option>ON_EVENT_KEEP_LAST</rule-option>	
</edsAlgorithm>	

1. The class name of the object to use must be given here, and the containing JAR file should have been listed in the *drivers* section of the control section.
2. The algorithm configuration is provided as text. The object must be able to parse the text provided to configure itself. CANARY does not try to configure the algorithm using the values provided in the other keys. The configuration text cannot have YAML comments in it. They are interpreted as literal text and passed to the object during configuration.

5.6 Monitoring Stations Section

This section describes the configuration of monitoring stations. For a given station, signals and algorithms are chosen from those defined previously in the *signals* and *algorithms* sections. Additionally, an input data source must be chosen from one of those defined in the *data sources* section. If an output to a database is desired, it must be added to the station definition as well. The description of this section and the parameters and values is given below along with examples.

- *monitoring stations:* – This starts the station section of the configuration file. Below this entry, the station is defined, including the signals and algorithms to use in the analysis.

  Each list entry is composed of:

  - *id:* – This parameter defines the name of the station. It is used as the basis for naming all of the output files. It is a text string and cannot contain spaces.
  - *location id number:* – This parameter specifies a user-defined physical location number. It must be an integer.
  - *station tag name:* – This parameter defines a station tag name for use in the LOCATION_ID field of output tables. If omitted, the station id string is used instead.

- *`station id number:`* – This parameter specifies a configuration number for the station. It can be used to differentiate between two different stations at the same physical location, or two different algorithms using the same signals.
- *`enabled:`* – This parameter defines if the station is going to be used in the analysis. The options are **yes** or **no**. It allows multiple stations to be listed within a configuration file, so additional stations can be analyzed with minimal editing of the configuration file.
- *`inputs:`* – This parameter starts the inputs subsection. It is a list, and each entry must be preceded by a minus sign (`-`) character.

  Each entry is composed of:

  - *`id:`* – This parameter specifies the data source. It has to be consistent with the *id* defined in the *data sources* section.
- *`outputs:`* – This parameter starts the outputs subsection. It is unnecessary if the EDSD file is sufficient output. This is a list, and each entry must be preceded by a minus sign (`-`) character.

  Each entry is composed of:

  - *`id:`* – This parameter specifies the data source where the event results are written. It has to be consistent with the *id* defined in the *data sources* section.
- *`signals:`* – This parameter starts the signals subsection. Below this entry, the signals to include in this station are listed. This is a list, and each entry must be preceded by a minus sign (`-`) character.

  Each entry is composed of:

  - *`id:`* – This parameter specifies the signal for this station. It has to be consistent with the *id* defined in the *signal* section.
  - *`cluster:`* – This parameter specifies if pattern matching is to be used for this signal. It is an optional parameter. The options are **yes** or **no**. If **yes**, the signal is not included in the pattern recognition algorithm. This is associated with an *id* entry and should not be preceded by a minus sign.
- *`algorithms:`* – This parameter starts the algorithms subsection. Below this entry, the algorithms to include in this station are listed. This is a list, and each entry must be preceded by a minus sign (`-`) character.

  Each entry is composed of:

  - *`id:`* – This parameter specifies the algorithm for this station. It has to be consistent with the *id* defined in the *algorithm* section.
  - *`cluster:`* – This parameter specifies if pattern matching is to be used for this algorithm. It is an optional parameter. The options are

yes or **no**. If **yes**, the pattern library defined in the *algorithm* section is accessed as part of the event detection. This is associated with an *id* entry and should not be preceded by a minus sign.

5.6.1 Example MS1: A simple monitoring station

This example shows a monitoring station configuration using only a few signals, a file-based input, and a simple algorithm.

Table 24. Configuration code for a simple monitoring station entry.

YAML Configuration of Monitoring Stations	Comments
monitoring stations:	# MS1-1
- *id*: **STATION_AAA**	# Ex. MS1
location id number: **28**	
station tag name: **STATION_AAA_NOCLUST**	
station id number: **1**	
enabled: **yes**	
inputs:	
- *id*: **EXMPL_CSV_IN**	# See IO1
signals:	
- *id*: **SIG_S1_CL2**	# MS1-2
- *id*: **SIG_S2_PH**	
- *id*: **SIG_S4_PRES**	# MS1-3
- *id*: **SIG_S5_CAL_HW**	# MS1-4
algorithms:	
- *id*: **ALG_A1**	# See A1

1. The *monitoring stations* key can only occur once per file.
2. By using SIG_S1_CL2, the monitoring station automatically adds in the associated alarm signal, SIG_S3_CL2_ALM. The user does not need to add in any alarm signals to a monitoring station entry.
3. The pressure signal is an **OP** type signal, so the algorithm does not analyze it for events.
4. Only one **CAL** signal can be defined per monitoring station.

5.6.2 Example MS2: A monitoring station with database I/O and an algorithm using clustering

This example shows a monitoring station configuration using a database and an algorithm with clustering enabled. Most of the other sections only include the section title key in the first example of the section (i.e., the *monitoring stations* key on the first line below); please be aware that this example does include this line.

Table 25. Configuration code for a multi-input and output monitoring station with complex algorithms.

YAML Configuration of Monitoring Stations	Comments
monitoring stations:	# Ex. MS2
- *id*: **MS_PLANT23**	
location id number: **23**	
station tag name: **MONTERREY_PLANT**	
station id number: **1**	
enabled: **yes**	
inputs:	# MS2-1
- *id*: **EXMPL_DB_INPUT1**	
- *id*: **EXMPL_DB_INPUT2**	
outputs:	
- *id*: **EXMPL_DB_OUTPUT**	
signals:	
- *id*: **PLANT23_CL2**	
- *id*: **PLANT23_PH**	
- *id*: **PLANT23_COND**	
- *id*: **PLANT23_TOC**	
- *id*: **PLANT23_PRES**	
cluster: **no**	# MS2-2
- *id*: **PLANT23_CAL_COMPOSITE**	
cluster: **no**	# MS2-3
algorithms:	
- *id*: **ALG_A4**	

1. This example uses the multiple inputs from example IO5 and shows how to combine them.
2. The algorithm ALG_A4 has a pattern-matching component, but pressure was not included in the creation of the pattern library, so it needs to be excluded.
3. Calibration signals are excluded from the clustering automatically, but it does not hurt to make sure.

5.7 Complete Configuration Files

To run CANARY, a combination of the example sections of configuration files used in this section of the CANARY User Manual is required. Samples of complete configuration files are provided in the CANARY installation, under the directory "${USERHOME}\My CANARY\Examples."

6 Training CANARY and Choosing Parameter Settings

General information on event detection systems, false alarms versus false negatives, and other significant issues are discussed in a comprehensive report, Water Quality Event Detection Systems for Drinking Water Contamination Warning Systems: Development, Testing, and Application of CANARY (Murray et al, 2010), available from the EPA web site: http://www.epa.gov/nhsrc. . While this user's manual describes how to pick various parameters, the implications of those choices are better discussed in the report, and users are encouraged to review the information provided in that document prior to deploying CANARY in real-time. In addition, training and tutorials are available at the CANARY Trac-site: https://software.sandia.gov/trac/canary.

7 References

Bezdek, J 1981, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, New York.

Dunn, JC 1973, "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters", Journal of Cybernetics, vol 3, pp. 32-57.

Gaffney, SJ 2004, "Probabilistic Curve-Aligned Clustering and Prediction with Regression Mixture Models", PhD Dissertation, Department of Information and Computer Science, University of California, Irvine.

Hall, J, Zaffiro, AD, Marx, RB, Kefauver, PC, Krishnan, ER, Haught, RC & Herrmann, JG 2007, "On-line water quality parameters as indicators of distribution system contamination", Journal of the American Water Works Association, vol 99, no. 1, pp. 66-77.

Hart, DB, McKenna, SA, Klise, KA, Cruz, VA & Wilson, MP 2007, "CANARY: A water quality event detection algorithm development and testing tool", Proceedings of ASCE World Environmental and Water Resources Congress 2007, ASCE, Tampa FL.

Klise, KA & McKenna, SA 2006a, "Multivariate applications for detecting anomalous water quality", Proceedings of the 8th Annual Water Distribution Systems Analysis (WDSA) Symposium, ASCE, Cincinnati OH.

Klise, KA & McKenna, SA 2006b, "Water quality change detection: multivariate algorithms", Proceedings of SPIE Defense and Security Symposium 2006, International Society for Optical Engineering (SPIE), Orlando FL.

McKenna, SA, Hart, DB, Klise, KA, Cruz, VA & Wilson, MP 2007, "Event detection from water quality time series", Proceedings of ASCE World Environmental and Water Resources Congress (EWRI) 2007, ASCE, Tampa FL.

McKenna, SA, Klise, KA & Wilson, MP 2006, "Testing water quality change detection algorithms", Proceedings of the 8th Annual Water Distribution Systems Analysis Symposium (WDSA), ASCE, Cincinnati OH.

Murray, R., Haxton, T., McKenna, SA, Hart, DB, Klise, KA, Koch, M, Vugrin, ED, Martin, S, Wilson, MP, Cruz, VA, and Cutler, L. 2010. Water Quality Event Detection Systems for Drinking Water Contamination Warning Systems: Development, Application, and Testing of CANARY. EPA/600/R-10/036.

The MathWorks 2002, Signal Processing Toolbox User's Guide, 6th edn, The MathWorks.

US EPA 2005, "WaterSentinel Online Water Quality Monitoring as an Indicator of Drinking Water Contamination", EPA, U.S. Environmental Protection Agency, 817-D-05-002.

Vugrin, E., S.A. McKenna and D. Hart, 2009, “Trajectory Clustering Approach for Reducing Water Quality Event False Alarms”, Proceedings of ASCE World Environmental and Water Resources Congress (EWRI), Kansas City, Missouri, May 17-21.

Appendix A - YAML Tips

YAML (YAML Ain't Markup Language) is a text format for representing data in a human readable manner. The official website for YAML is at http://www.yaml.org. Very good documentation is available, but a simple overview of the basic structure is presented here, along with some advice on how to avoid common mistakes.

YAML has three types of data objects: **scalars**, **lists**, and **named entries**. A value can take on any of these three types.

- **Scalars** – scalar values are numbers, text strings, or Boolean values (true or false). Anything between single- or double-quote characters is a string. If a value cannot be changed into any other type of object, it reverts to being treated as a string.
- **Lists** – lists are either a list of objects on separate lines (much like the list here) where each item is indicated with a single dash (-) character and the dashes all have the same indentation. Alternately, a list can be a set of comma-separated values on a single line with the whole group surrounded by square brackets.
- **Named entries** – a named entry, or mapping, is a collection of ***key*: value** pairs. These can be on separate lines, with the same indentation, or comma-separated between curly braces. The ***key*** must be a valid scalar. The **value** can be of any type.

Table 26. Examples of the three YAML data types.

Scalars	Lists	Named entries

`9.0`
`true`
`off`
`.inf`
`"This is a string"`
`This is also a string`
	`[ 1, [2, 2.5, 2.8, 3]`
	
	`- 1`
	`- - 2`
	`  - 2.5`
	`  - 2.8`
	`- 3`
		`{ A:1, B:2}`
		`A: 1`
		`B: 2`
		
		`my children:`
		`  eldest: 15`
		`  youngest: 12`

- **Special words** – If the following words are not part of a larger string, then they have special meaning:
  - **Boolean values** – true, false, yes, no, on, off
  - **Numeric values** – .inf, .nan, null
- **Control characters** – do not start a string with !, &, or *. Three dashes (---) and three periods (...) at the beginning of a line have special meaning. Finally, a single pipe character (|) indicates that a multi-line string follows.

- **Comments** – a comment is anything after the # symbol until the end of the line (unless it is inside quotes denoting a string, or it is in a multi-line string)
- **Indentation and spacing** – this is the biggest "gotcha!" in YAML. ***Do not use tabs!*** Tab characters do not evaluate well when mixed with spaces. It is much better to use spaces to indent different lines.
    - Different levels of indentation are used to indicate nested objects. When the indentation returns to the previous level, the object is closed.
    - Comments should be indented to the same level as the surrounding object. Adding a comment with less indentation may close an object before the user wants to do so.

In the next table, an example showing a complete YAML document is given, with line-by-line comments in the second column.

Table 27. A full YAML document with annotations.

`---`	Special characters that indicate the start of a YAML document.
`# This is a comment line`	A comment at the top of the file is usually a good idea so the user can remember why they wrote it.
`my first key:`	The root level of the document (unindented) has to be either a list (every line starts with a dash) or a mapping (every line starts with a key:). Configuration files almost always start with keys at the root level.
`- 12.0`	The value of "my first key" is a list. A value that is a list can have the dash character at the same level of indentation as the first character of the key.
`- 13.2`	Second element in the list. Any number with a decimal point is automatically cast as a real number.
`- .inf`	The special value for representing IEEE Infinity
`- 12`	Unlike the first value in the list, this is an integer, not a real number.
`my second key:`	The value of "my second key" is going to be a mapping, so each of the child keys needs to be indented.
`  name: David`	The value of "name" is a string, even without quotes.
`  age: "32"`	The value of "age" is also a string, not an integer, because it is inside quotes.
`  favorite food: |`	The |-character signals the start of a multiline string.
`    Pizza is good,`	Note that the string needs to be indented from the first character of the key preceding it.
`    but only with`	The second line will be added to the first before processing
`      pepperoni!`	The extra spaces before "pepperoni" will be saved in the string when it is processed, because of the extra indentation
`  my list: [ 1, 2, 3]`	Going back to the old indentation closes the string, and keeps "my list" as part of the "my second key" mapping. Note how the list uses the bracketed format this time.
`finally: null`	The "null" special word means that the value of "finally" has nothing in it. It is empty.

Finally, here is an example of some bad YAML that will break and explanations why. In this table, each row is its own example, and may contain multiple lines of YAML code.

Table 28. YAML that will not work!

`my key: is fun` `  to break`	WRONG! Multi-line text has to start with the |-character as the only character on the line with the – or |
`value: 2` `value: 4`	WRONG! An object cannot have the same key twice.

`my list:`
`  - 12`
`  - 13`
`- 15`
`  - 29`
	WRONG! Notice how the third entry is not indented to the same level as the others.

`0.2934` `address: 2020 Some Street`	WRONG! The number at the beginning is a scalar, not a list or mapping. The whole document would break.
`my list: 35` `- 29`	WRONG! The first list entry has to be on the next line, or the entire list on one line surrounded by square brackets.

`my text string: |`
`  This is a test`
`  of a multiline # bbb`
`  string.`
	Not wrong, but will not give the user what he or she expects. The resulting string is literally: “This is a test of a multiline # bbb string.” This is probably not what is desired.

In the end, YAML is not a terribly complicated format to learn. If the reader wants more information, please visit the YAML documentation pages on its website. The most common error is indentation. As long as the user makes sure to use spaces, not tabs, and lines things up, they should have few problems. Many good, free text editors are available that will highlight the YAML code in different colors for keys, values, lists, strings, and numbers, and which will help check the indentation. It is probably worth the time to try a few and find one that the user likes to use when editing these types of files, but even Notepad will work well in a pinch.