WQX Web API - 9/28/2018

WQX Web Application Programming Interface (API)

Introduction

EPA's Water Quality eXchange team, the Chesapeake Bay Program, and EPA Contractor Gold Systems have developed, tested, and utilized a new way to submit data to WQX and the Water Quality Portal. Until very recently the only two methods to publish your data via WQX were to set up an automated node to node communication via EPA's Exchange Network or to manually upload your data using the WQX Web user interface. The Chesapeake Bay Program recently began publishing their data using automated API services through WQX Web. The new WQX Web API services allow a data submitter to automate data submissions like a node but through WQX Web. The API is intended to provide programmatic access to WQX Web data submission functions and procedures. The intended audience is programmers who are familiar with the concepts and techniques of WQX data submissions and Web Services. This project is innovative because maintenance on the data mapping is managed within the WQX Web. The interface makes maintaining data translation rules easy and user friendly. WQX Web maintains compatibility with new business rules, elements and xml schema validation.hing their data using automated API services through WQX Web.

Access to the WQX Web API

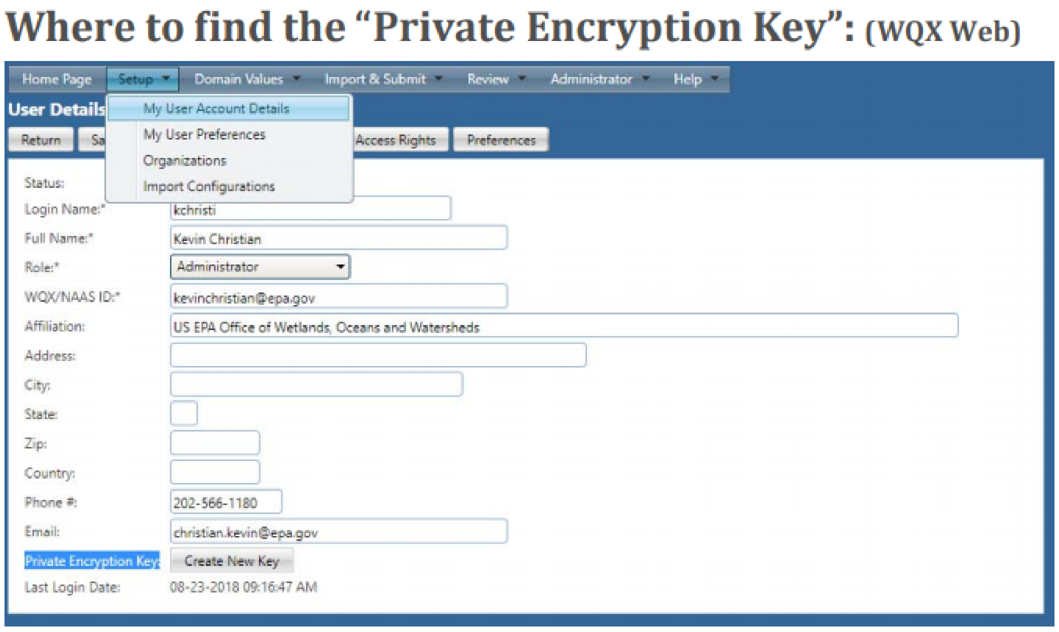
The API is available to registered users within the WQX Web application. Before using the API, users must obtain a unique 88-character "Private Encryption Key": associated with a registered WQX Web user account. Developers / programmers may initial use their private and personal account. Later for releasing production applications one may email a request to register an application (WQX Web user) account

All API access is through an authenticated Uniform Resource Identifier (URI). Data is submitted by sending an HTTP GET/POST to the URI with appropriate parameters supplied. The minimum parameters for every request include the UserID, Timestamp, and Signature, which includes the name of the method being invoked.

Where to find the "Private Encryption Key": (WQX Web)

Click "Setup" Menu,

Click "My User Account Details" menu item



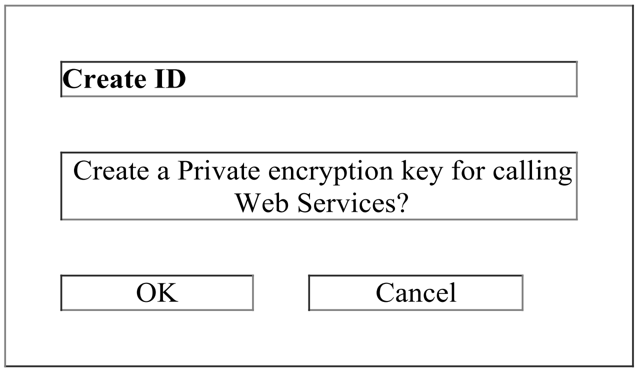
Status:	
Login Name:*	kchristi
Full Name:*	Kevin Christian
Role:*	Administrator
WQX/NAAS ID:*	kevinchristian@epa.gov
Affiliation:	US EPA Office of Wetlands, Oceans and Watersheds
Address:	
City:	
State:	
Zip:	
Country:	
Phone #:	202-566-1180
Email:	christian.kevin@epa.gov
Private Encryption Key:	Create New Key
Last Login Date:	08-23-2018 09:16:47 AM

Home Page	**Setup**	**Domain Values**	**Import & Submit**	**Review**	**Administrator**	**Help**

User Details

Return	Save	Cancel	User Access Rights	Preferences

Status:	
Login Name:	
Full Name:	
Role:	
WQX/NAAS ID:	
Affiliation:	
Address:	
City:	
State:	
Zip:	
Country:	
Phone #:	
Email:	
Private Encryption Key:	Create New Key
Last Login Date:	



Create a Private encryption key for calling Web Services?

1. DEVELOPERS and PROGRAMMERS click <<OK>>
2. Graphical User Interface or App Users click <<Cancel>>

WQX Domain Value Services and Downloads

EPA makes available domains of Water Quality eXchange (WQX) data elements for both submission and retrievals. Domains are publicly accessible to assist others in conforming to a consistent nomenclature. The domain value web service can be queried to determine the EPA-supplied values of the domains used for both submission and retrievals.

GetDomainValues

To download the domain lists (as zipped CSV files), click the links below:

- All - The Entire Domain Lists (ZIP) | (XML)

Individual Domain Values Lists: ActivityGroupType (ZIP) | (XML) ActivityMedia (ZIP) | (XML) ActivityMediaSubdivision (ZIP) | (XML) ActivityRelativeDepth (ZIP) | (XML) ActivityType (ZIP) | (XML) AddressType (ZIP) | (XML) AnalyticalMethod (ZIP) | (XML) AnalyticalMethodContext (ZIP) | (XML) Assemblage (ZIP) | (XML) BiologicalIntent (ZIP) | (XML) CellForm (ZIP) | (XML) CellShape (ZIP) | (XML) Characteristic (ZIP) | (XML) CharacteristicAlias (ZIP) | (XML) CharacteristicGroup (ZIP) | (XML) CharacteristicWithPickList* (ZIP) | (XML) Country (ZIP) | (XML) County (ZIP) | (XML)	DetectionQuantitationLimitType (ZIP) | (XML) ElectronicAddressType (ZIP) | (XML) FrequencyClassDescriptor (ZIP) | (XML) Habit (ZIP) | (XML) HorizontalCollectionMethod (ZIP) | (XML) HorizontalCoordinateReferenceSystemDatum (ZIP) | (XML) MeasureUnit (ZIP) | (XML) MethodSpeciation (ZIP) | (XML) MetricType (ZIP) | (XML) MetricTypeContext (ZIP) | (XML) MonitoringLocationType (ZIP) | (XML) NetType (ZIP) | (XML) Organization (ZIP) | (XML) ReferenceLocationType (ZIP) | (XML) ResultDetectionCondition (ZIP) | (XML) ResultLaboratoryComment (ZIP) | (XML) ResultMeasureQualifier (ZIP) | (XML) ResultMeasureValuePickList* (ZIP) | (XML) ResultSampleFraction (ZIP) | (XML) ResultStatus (ZIP) | (XML) ResultTemperatureBasis (ZIP) | (XML) ResultTimeBasis (ZIP) | (XML) Note: CharacteristicWithPickList* and	ResultValueType (ZIP) | (XML) ResultWeightBasis (ZIP) | (XML) SampleCollectionEquipment (ZIP) | (XML) SampleContainerColor (ZIP) | (XML) SampleContainerType (ZIP) | (XML) SampleTissueAnatomy (ZIP) | (XML) SamplingDesignType (ZIP) | (XML) State (ZIP) | (XML) StatisticalBase (ZIP) | (XML) Taxon (ZIP) | (XML) TaxonAlias (ZIP) | (XML) TaxonGroup (ZIP) | (XML) TelephoneNumberType (ZIP) | (XML) TimeZone (ZIP) | (XML) ThermalPreservativeUsed (ZIP) | (XML) ToxicityTestType (ZIP) | (XML) Tribe (ZIP) | (XML) VerticalCollectionMethod (ZIP) | (XML) VerticalCoordinateReferenceSystemDatum (ZIP) | (XML) Voltinism (ZIP) | (XML) WellFormationType (ZIP) | (XML) WellType (ZIP) | (XML) ResultMeasureValuePickList* are the same.

Program Language

The API may be used with any language capable of issuing HTTPS requests with an HMAC-SHA256 encryption algorithm (Java, PHP, Perl, Python, C, etc.

Data Return Format

The API returns data in format(s): JSON, XML.

Documentation for the WQX Web API services can be referenced via WQX Web Help Menus. Example Code is also available via both WQX Web Help Menu and STORET FTP Site URL: ftp://ftp.epa.gov/storet/wqx/api/Example_code.txt

WQX Web API Described below are the RESTful web services for WQX Web. These services are designed and used to submit water quality data to WQX. The basic process flow is as follows:
1. Upload a file and optional attachment.
2. Request that WQX Web start importing the uploaded file. There are up to five for overriding generated element/value parameter combinations. These parameters can be used to alter any existing generated elements in the import configuration.
3. Get the status for that dataset, periodically, to confirm whether it imported without any errors
4. If the final status is "Imported", then ask WQX Web to submit that dataset to CDX. During that process, the data from your flat file or spreadsheet will be converted to XML (via an import configuration that you previously created) and submitted to CDX. Alternatively, you can request that WQX Web automatically export and submit your dataset to CDX upon completion of the import. It's important to understand that data is not final/permanent until it has been submitted to CDX and has received a final status of "Completed at CDX"
5. If a dataset receives a status of Failed, you can get a list of documents relating to the dataset, and download the "Import Log.xlsx" or "ProcessingReport.zip" (among other files) to review the issues with your import file or the submission to CDX.

At end of this document, you will find an example of how to upload a file and check the status of a dataset.

RESTful Web Service Methods:

- **Upload:** upload a file to the web server (to be imported).
  - Action
    - POST
  - File Name, Allowed file extension:
    - txt
    - csv
    - xlsx
    - xls
    - xml
    - zip
  - returns a fileId
- **UploadAttachment:** upload an attachment to the web server (to be imported).
  - Action
    - POST
  - File Name, Allowed file extension:
    - zip
  - returns a attachmentFileId

- **StartImport:** start importing a file and attachment that was previously uploaded

- Action
  - GET
- Parameters
  - importConfigurationId
  - fileId
  - attachmentFileId
    - Optional
  - fileType
    - Allowed values:
      - CSV
      - TAB
      - TILDE
      - PIPE
      - XLS
      - XLSX
  - newOrExistingData
    - Allowed values:
      - 0: file may contain new and/or existing data
      - 1: file contains new data only
      - 2: file contains existing data only (to be replaced)
  - uponCompletion
    - Allowed values
      - 0: do nothing
      - 1: start export
      - 2: start export and submit to CDX
  - uponCompletionCondition
    - Allowed values:
      - 0: not applicable
      - 1: start export only if no import errors
      - 2: start export only if no import errors and no warnings
      - 3: start export even when there are import errors.
        - WARNING: The only way you will know if there were errors will be to review the import log.
  - worksheetsToImport
    - comma delimited list of values (1-based), e.g. "1,3"
    - (this parameter value will be ignored when the fileType is not XLS or XLSX)
    - If no value is passed in (and it's applicable) then we'll use the value from the import configuration
  - ignoreFirstRowOfFile
    - true/false
    - (this parameter value will be ignored for Expert Mode Import Configurations)
  - generatedElementName1
    - Optional
      - Allowed values:
        - string
        - Must match an existing generated element name from the import configuration
  - generatedElementValue1
    - Optionally required if generatedElementName is populated
      - Allowed values:
        - string
  - generatedElementName2
    - Optional
      - Allowed values:
        - string
        - Must match an existing generated element name from the import configuration
  - generatedElementValue2
    - Optionally required if generatedElementName is populated
      - Allowed values:
        - string
  - generatedElementName3
    - Optional

            - Allowed values:
                - string
                - Must match an existing generated element name from the import configuration
    - generatedElementValue3
        - Optionally required if generatedElementName is populated
            - Allowed values:
                - string
    - generatedElementName4
        - Optional
            - Allowed values:
                - string
                - Must match an existing generated element name from the import configuration
    - generatedElementValue4
        - Optionally required if generatedElementName is populated
            - Allowed values:
                - string
    - generatedElementName5
        - Optional
            - Allowed values:
                - string
                - Must match an existing generated element name from the import configuration
    - generatedElementValue5
        - Optionally required if generatedElementName is populated
            - Allowed values:
                - string
  - returns a datasetId

- **StartXmlExport:** start creating the XML submission file (for CDX)
  - Action
    - GET
  - Parameters
    - datasetId
    - uponCompletion
      - Allowed values:
        - 0: do nothing
        - 1: submit to CDX
  - returns the intial status for the dataset

- **SubmitDatasetToCdx:** submit a dataset to CDX
  - Action
    - GET
  - Parameters
    - datasetId
  - returns the intial status for the dataset

- **SubmitFileToCdx:** submit a previously uploaded WQX XML file to CDX
  - Action
    - GET
  - Parameters
    - fileId
  - returns a datasetId

- **GetStatus:** get the status for a dataset. To avoid undue burden on the server, it is recommended that you not call this service more often than every 10 seconds. For large imports (longer than 20 minutes), calling this service periodically will guarantee that the server will not shutdown before the import completes.
  - Action

        - GET
    - Parameters
        - datasetId
    - returns the status, percent complete, and position in queue for the dataset
        - Possible Statuses:
            - Waiting to Import
            - Importing
            - Imported
            - Import Failed
            - Waiting to Export
            - Waiting to Export and Submit
            - Exporting
            - Exported
            - Export Failed
            - Processing at CDX
            - Completed at CDX
            - Failed at CDX
            - Waiting to Delete
            - Deleting
            - Delete Failed
            - Waiting to Update WQX
            - Updating WQX
            - Updated WQX
            - Update Failed

- **GetDocumentList:** get the list of available documents for a dataset
    - Action
        - GET
    - Parameters
        - datasetId
    - returns a list of document URLs which can be used to download the documents
    - documents that are typically available for a dataset include:
        - the original import file
        - event logs (for the import and/or export in WQX Web)
        - validation report (for the XML submission file - from CDX)
        - processing report (for the XML submission file - from WQX)

- **Projects:** returns projects for an organization
    - Action
        - GET
    - Parameters
        - OrganizationIdentifiersCsv
            - Comma delimited (e.g. "id1,id2,id3")
            - NOTE: No spaces
        - ProjectIdentifiersCsv
            - Comma delimited (e.g. "id1,id2,id3")
            - NOTE: No spaces
        - TransactionIdentifier
            - (e.g. "_23090c89-c6a6-4dd1-b16f-73f8ac36fac1"
        - LastChangeDateMin
            - Format: (mm/dd/yyyy or mm-dd-yyyy)
        - LastChangeDateMax
            - Format: (mm/dd/yyyy or mm-dd-yyyy)
        - StartRow
            - NOTE:
                - Zero Based
                - Conditionally required if RowsToRetrieve is supplied
        - RowsToRetrieve

    - NOTE:
        - Conditionally required if StartRow is supplied
- returns WQX Schema 2.2 element values including the transaction id for the project

- **MonitoringLocations:** returns locations for an organization
    - Action
        - GET
    - Parameters
        - OrganizationIdentifiersCsv
            - Comma delimited (e.g. "id1,id2,id3")
            - NOTE: No spaces
        - MonitoringLocationIdentifiersCsv
            - Comma delimited (e.g. "id1,id2,id3")
            - NOTE: No spaces
        - MonitoringLocationName
            - Wildcards are supported (e.g. "Location%" means anything starting with "Location")
        - MonitoringLocationType
            - Wildcards are supported (e.g. "Location%" means anything starting with "Location")
            - Allowed values:
                - Atmosphere
                - BEACH Program Site-Channelized stream
                - BEACH Program Site-Estuary
                - BEACH Program Site-Great Lake
                - BEACH Program Site-Lake
                - BEACH Program Site-Land
                - BEACH Program Site-Land runoff
                - BEACH Program Site-Ocean
                - BEACH Program Site-River/Stream
                - BEACH Program Site-Storm sewer
                - BEACH Program Site-Waste sewer
                - Borehole
                - Canal Drainage
                - Canal Irrigation
                - Canal Transport
                - Cave
                - CERCLA Superfund Site
                - Channelized Stream
                - Combined Sewer
                - Constructed Diversion Dam
                - Constructed Tunnel
                - Constructed Water Transport Structure
                - Constructed Wetland
                - Estuary
                - Facility Industrial
                - Facility Municipal Sewage (POTW)
                - Facility Other
                - Facility Privately Owned Non-industrial
                - Facility Public Water Supply (PWS)
                - Floodwater non-Urban
                - Floodwater Urban
                - Gallery
                - Gas-Condensate
                - Gas-Engine
                - Gas-Extraction
                - Gas-Flare
                - Gas-Monitoring Probe
                - Gas-Passive Vent
                - Gas-Subslab
                - Gas-Temporary

    - Great Lake
    - Lake
    - Land
    - Land Flood Plain
    - Land Runoff
    - Landfill
    - Leachate-Extraction
    - Leachate-Head Well
    - Leachate-Lysimeter
    - Leachate-SamplePoint
    - Local Air Monitoring Station
    - Mine Pit
    - Mine/Mine Discharge
    - Mine/Mine Discharge Adit (Mine Entrance)
    - Mine/Mine Discharge Tailings Pile
    - Mine/Mine Discharge Waste Rock Pile
    - National Air Monitoring Station
    - Ocean
    - Oil and Gas Well
    - Other-Ground Water
    - Other-Surface Water
    - Pipe, Unspecified Source
    - Playa
    - Pond-Anchialine
    - Pond-Sediment
    - Pond-Stock
    - Pond-Stormwater
    - Pond-Wastewater
    - Reservoir
    - River/Stream
    - River/stream Effluent-Dominated
    - River/Stream Ephemeral
    - River/Stream Intermittent
    - River/Stream Perennial
    - Riverine Impoundment
    - Seep
    - Spigot / Faucet
    - Spring
    - State/Local Air Monitoring Station
    - Storm Sewer
    - Survey Monument
    - Test Pit
    - Waste Pit
    - Waste Sewer
    - Well
    - Wetland Estuarine-Emergent
    - Wetland Estuarine-Forested
    - Wetland Estuarine-Scrub-Shrub
    - Wetland Lacustrine-Emergent
    - Wetland Palustrine Pond
    - Wetland Palustrine-Emergent
    - Wetland Palustrine-Forested
    - Wetland Palustrine-Moss-Lichen
    - Wetland Palustrine-Shrub-Scrub
    - Wetland Riverine-Emergent
    - Wetland Undifferentiated
- TransactionIdentifier
  - e.g. "_23090c89-c6a6-4dd1-b16f-73f8ac36fac1"
- LastChangeDateMin
  - Format: (mm/dd/yyyy or mm-dd-yyyy)

        - LastChangeDateMax
            - Format: (mm/dd/yyyy or mm-dd-yyyy)
        - StartRow
            - NOTE:
                - Zero Based
                - Conditionally required if RowsToRetrieve is supplied
        - RowsToRetrieve
            - NOTE:
                - Conditionally required if StartRow is supplied
    - returns WQX Schema 2.2 element values including the transaction id for the monitoring location

Each web service call will include a header with the following items:

1. X-UserID
    - The caller's "User ID" to determine rights to private data
2. X-Stamp
    - Timestamp when the caller made the request. This must be UTC Time (i.e. Greenwich Mean Time), so you must convert from your local time zone.
    - Format is (mm/dd/yyyy hh:mi:ss AM)
3. X-Signature
    - Signature is made up of the following pieces of information (concatenated together as a single string value):
        - User ID
        - Timestamp
        - URI
        - Request Method
    - Signature is then encrypted using the HMAC-SHA256 encryption algorithm and placed in the header element.

The following example demonstrates how to call a web service with the required header information.

This example uses C#, but you can use any language that provides an HMAC-SHA256 encryption algorithm:

```
using System;
using System.Collections.Generic;
using System.Linq;
using  System.Text;
using System.Security.Cryptography;
using System.Net;


namespace WebServiceConsole {


    class Program {

        static void Main (string[] args) {
            // This user id and private encryption key are for illustration purposes only.
            string userID = "wqxweb_userid";
            string privateKey =
"zHPJBAZbMaPOIVqmz1afDRW0HhovKuExPlvKsg8SIdIXCYeoHeI/bQaG0HBahYTCaGno5EljBJmzd5AKjfHC2w==";
```

```
        string timeStamp = "";
        string data = "";
        string fileId = "";
        byte[]  signature;
        Uri uri;

        using (WebClient client = new WebClient()) {
            try {
                //***************************UPLOAD A FILE***********************
                // The URI of the upload webservice with parameters
                uri = new Uri("https://cdx.epa.gov/WQXWeb/api/Upload/Projects.xlsx");
                client.Headers.Add("X-UserID", userID);
                timeStamp = DateTime.UtcNow.ToString();
                client.Headers.Add("X-Stamp", timeStamp);
                // The encrypted data must be in this order and format
                data = String.Format("{0}{1}{2}{3}", userID, timeStamp, uri.ToString(), "POST");
                signature = Encoding.UTF8.GetBytes(data);
                using (HMACSHA256 hmac = new HMACSHA256(Convert.FromBase64String(privateKey))) {
                    byte[] signatureBytes = hmac.ComputeHash(signature);
                  client.Headers.Add("X-Signature", Convert.ToBase64String(signatureBytes));
                }
                // Decide what type of content to be returned (text/plain, json, xml)
                client.Headers.Add("Content-Type", "text/plain");
                FileStream sourceFile = new FileStream("c:\users\user\Documents\ImportFile\Projects.xlsx", FileMode.Open);
                BinaryReader binReader = new BinaryReader(sourceFile);
                byte[] fileInput = new byte[sourceFile.Length]; //create byte array of size file
                for (i = 0; i < sourceFile.Length; i++)
                    fileInput[i] = binReader.ReadByte(); //read until done
                sourceFile.Close(); //dispose of stream
                binReader.Close(); //dispose of reader
                fileInput = client.UploadData(uri, fileInput);
                fileId = System.Text.Encoding.UTF8.GetString(fileInput);
                fileId = fileId.Substring(1, fileId.Length - 2);

                //***************************GET STATUS OF DATASET***********************
                client.Headers.Clear();
                // The URI of the get dataset status webservice with parameters
                uri = new Uri("https://cdx.epa.gov/WQXWeb/api/GetStatus?datasetId=12345");
                client.Headers.Add("X-UserID", userID);
                string timeStamp = DateTime.UtcNow.ToString();
                client.Headers.Add("X-Stamp", timeStamp);
                // The encrypted data must be in this order and format
                string data = String.Format("{0}{1}{2}{3}", userID, timeStamp, uri.ToString(), "GET");
                byte[] signature = Encoding.UTF8.GetBytes(data);
                using (HMACSHA256 hmac = new HMACSHA256(Convert.FromBase64String(privateKey))) {
                    byte[] signatureBytes = hmac.ComputeHash(signature);
                    client.Headers.Add("X-Signature", Convert.ToBase64String(signatureBytes,
    Base64FormattingOptions.None));
                }
                // Decide what type of content to be returned (json, xml)
                client.Headers.Add("Content-Type", "application/xml");
                //client.Headers.Add("Content-Type", "application/json");
                // Process the response
                Byte[] httpMessage = client.DownloadData(uri);
                if (httpMessage.Length > 0) {
                    Console.WriteLine(Encoding.ASCII.GetString(httpMessage));
                }
            }
            catch (Exception e) {
                Console.WriteLine(e.Message);
```

```
                    Console.Read();
                }
            }
            Console.Read();
        }
    }
}
```