



AQUATIC  
INFORMATICS™

FASTER ANALYSIS.BETTER DECISIONS.



## AQUARIUS 3.10 Acquisition Service API

April 8, 2016

Copyright © 2016, Aquatic Informatics Inc.



<b>1 Introduction .....</b>	<b>3</b>
<b>2 Programming Environment .....</b>	<b>3</b>
<b>3 General Description of API .....</b>	<b>3</b>
<b>4 Data Types and Method Descriptions .....</b>	<b>4</b>
Method: <i>GetAuthToken</i> .....	5
Method: <i>GetAllLocations</i> .....	6
Method: <i>GetLocationId</i> .....	7
Method: <i>GetLocation</i> .....	8
Method: <i>CreateLocation</i> .....	9
Method: <i>ModifyLocation</i> .....	11
Method: <i>GetFieldVisitsByLocation</i> .....	12
Method: <i>GetFieldVisitsByLocationChangedSince</i> .....	13
Method: <i>GetFieldVisitsByLocationAndDate</i> .....	14
Method: <i>SaveFieldVisit</i> .....	15
Method: <i>CreateTimeSeries</i> .....	16
Method: <i>CreateTimeSeries2</i> .....	17
Method: <i>GetTimeSeriesID</i> .....	18
Method: <i>GetTimeSeriesID2</i> .....	19
Method: <i>GetTimeSeriesList</i> .....	20
Method: <i>GetTimeSeriesListForLocation</i> .....	21
Method: <i>AppendTimeSeriesFromBytes</i> .....	22
Method: <i>AppendTimeSeriesFromBytes2</i> .....	24
Method: <i>AppendAndMerge</i> .....	24
Method: <i>DeleteTimeSeriesPointsByTimeRange</i> .....	25
Method: <i>UndoAppend</i> .....	26
Method: <i>DeleteTimeSeries</i> .....	27
Method: <i>IsConnectionValid</i> .....	28
Method: <i>KeepConnectionAlive</i> .....	29
<b>Appendix A: Field Visit Data Model .....</b>	<b>30</b>
<b>Appendix B. Sample Code .....</b>	<b>31</b>



## 1 Introduction

This document describes the AQUARIUS Acquisition Service API. This software interface is intended for an acquisition system to access and update time series and field visit data in AQUARIUS. The interface uses industry standard SOAP web services.

## 2 Programming Environment

Refer to the AQUARIUS System Requirements document for details about the AQUARIUS Server requirements.

External tools can connect to the AQUARIUS Acquisition web service using the SOAP protocol. If Microsoft Visual Studio is used, then the web service can be referenced directly, otherwise interfaces can be generated from the WSDL description or coded by hand.

## 3 General Description of API

This API allows an acquisition system to perform functions with Aquarius data, including:

- Creation of new time series
- Query of information about time series
- Append new data to time series

Each time series in AQUARIUS is associated with a parent location (sometimes known as a station). This parent location serves as a “container” for one or more time series. Typically, the location is configured to correspond with a real physical measuring location.

A location or time series in AQUARIUS may be identified in various ways:

- **Numeric Identifier.** This is a unique numeric identifier (essentially a database primary key) which is assigned by AQUARIUS.
- **Unique Text Identifier.** For a location, this is the full name of the location (the full text name as seen in Springboard in the “Location Name” column). For a time series, this identifier contains the parameter, label, and parent location, for example [HG.Telemetry@02AB014](#).



## 4 Data Types and Method Descriptions

- ▶ Unless otherwise specified, all string data is UTF8 Unicode and all byte arrays are UTF8 encoded strings.
- ▶ Any string that contains a comma must be wrapped in double quotes.
- ▶ The SOAP interface is found at the following URL:  
<http://<yourserver>/AQUARIUS/AQAcquisitionService.svc>  
And the WSDL is here:  
<http://<yourserver>/AQUARIUS/AQAcquisitionService.svc?WSDL>

Note that the WSDL contains additional methods which are not described in this document. These methods are for internal use only; these methods are not supported or required for external use.

Authentication to the web service is token based. To acquire a token you use the GetAuthToken method. Passwords passed to this method can be either clear-text or RSA encrypted. If using RSA encryption, the RSA public key is:

```
<RSAKeyValue>  
<Modulus>  
3GmH/grhUyuiWlBDhNliZN3PYy2JPxJ5nG4t1CmQW8ZGPPSsWY1AYIEU3b0A7eKOL3BbAiQzn0GpaSU9  
HT452zKhqXE+G2nxw7Y0tJYfBzsTbBSPcHaUTHd/YyqDwDhuuJ+RrCqQfgnkq+YXnlCU1CJwVlF5HUy  
LWWHaUgaNlEqs=  
</Modulus>  
<Exponent>AQAB</Exponent>  
</RSAKeyValue>
```

This token is embedded in the HTTP header information in each subsequent SOAP call as AQAuthToken (see code sample in Appendix B).

Acquisition Service allows two types of binding: BasicBinding and WSHttpBinding. If you are using WSHttpBinding the following should be added to <wsHttpBinding> in <ProgramFiles>\Aquatic Informatics\AQUARIUS Server\WebServices\Web.config file:

```
<binding name="WSHttpBinding_IAQAcquisitionService"  
maxReceivedMessageSize="2147483647">  
</binding>
```

## Method: GetAuthToken

This method is used to log in to the AQUARIUS Acquisition web service and obtain a token for accessing the API.

An authentication token is required to call any other API methods (i.e., it must be passed in to all subsequent API calls in order to authenticate).

Authentication of the API may be disabled by using AQUARIUS Manager to change the Global Setting with SettingGroup = "AquariusAuthentication" and SettingKey = "AQAcquisitionService." This setting is set to "true" by default.

### Interface

SOAP method: `string GetAuthToken (string user, string encodedPassword`

#### Input

string user	AQUARIUS user name (as configured in AQUARIUS Manager)
string encodedPassword	Hashed password of the AQUARIUS user. May be passed as clear text, or as RSA-encrypted hash (see above for RSA public key).

#### Output

string authToken	The token is a SHA signature. The server caches this token for use during the session. Tokens expire after 60 minutes of inactivity.
------------------	--



## Method: GetAllLocations

This method is used to retrieve a list of all locations in the system.

This method returns an array of Data Transfer Objects (DTOs) which contains information about the locations.

### Interface

SOAP method: `LocationDTO[] GetAllLocations ()`

#### Input

(none)

n/a.

#### Output

DTO containing information about all locations in the system. This DTO contains the following information:

- long LocationId: unique numeric identifier of the location.
  - string Identifier: Identifier of the location.
  - string LocationName: Name of the location.
  - float Longitude: Longitude of the location.
  - float Latitude: Latitude of the location.
  - float UtcOffset: Time zone offset from UTC, in *hours*. For example, PST would have a value of -8.
  - string LocationPath: This is the full path of the folder where the location lives in the primary folder structure. Parent and child folders are separated with ".". Example: "All Locations.BC.Vancouver Office".
  - string LocationTypeName: Name of the AQUARIUS location type table (e.g., "Hydrology Station").
  - ExtendedAttributes: Extended attributes represented as key-value pairs. Key is the column name of the table/view which is specified in the LocationType table and from
- LocationDTO[]



the built-in common attribute table: Location\_Extension. Value is the value in that column. If the value is null in the table, there will be no entry for that column.

- float Elevation: The elevation of the location.
- string ElevationUnits: Units that are used for elevation (e.g. “m” and “ft”).

## Method: GetLocationId

This method is used to retrieve the numeric identifier of a specific location, given the text identifier.

### Interface

SOAP method: `long GetLocationId (string locationIdentifier)`

#### Input

string locationIdentifier	Text identifier of the location. For example “02AB014”
---------------------------	--

#### Output

long locationId	Numeric identifier of the location.
-----------------	-------------------------------------



## Method: GetLocation

This method is used to retrieve information about a specific location.

This method returns a Data Transfer Object (DTO) which contains information about the location.

### Interface

SOAP method: `LocationDTO GetLocation (long locationId)`

#### Input

long locationId	Numeric identifier of the location.
-----------------	-------------------------------------

#### Output

LocationDTO	DTO containing information about all locations in the system. See <b>GetAllLocations</b> for a description of the DTO.
-------------	---



## Method: CreateLocation

This method is used to create a new location.

This method takes as input a Data Transfer Object (DTO) which contains information about the location.

### Interface

SOAP method: `long CreateLocation (LocationDTO newLocation)`

#### Input

Information about the location to be created. See GetAllLocations for a description of the DTO. Notes on parameter usage:

- LocationId: should be NULL. The unique numeric identifier will be generated by AQUARIUS.
- Identifier: Cannot be NULL or empty and must be unique across the whole database. An exception is thrown if not unique.
- LocationName: Cannot be NULL or empty and must be unique within the folder.
- LocationPath: The full path of the primary folder where the location is to be created, with each folder level separated by a period (e.g. "All Locations.Alaska.Kodiak"). If this parameter is empty or null, the location is created in the root folder (i.e. "All Locations").
- LocationTypeName: Must match an existing AQUARIUS LocationType.
- Longitude: Nullable. But if Latitude has value, Longitude must also be assigned a value. Default datum is "WGS 84."
- Latitude: Nullable. But if Longitude has value, Latitude must also be assigned a value. Default datum is "WGS 84."
- UtcOffset: The time offset from Coordinated Universal Time (UTC) in hours. Default is 0.
- ExtendedAttributes: Extended attributes represented as key-value

LocationDTO newLocation



pairs. Key is the column name of the table/view which is specified in the LocationType table and from the built-in common attribute table: Location\_Extension. If a column in the attribute table does not allow null and a value is not specified for that column, an exception is thrown.

- float Elevation: The elevation of the location.
- string ElevationUnits: Units that are used for elevation (e.g. “m” and “ft”).

## Output

Long LocationId	Numeric identifier of the newly created location.
-----------------	---



## Method: ModifyLocation

This method is used to modify an existing location.

This method takes as input a Data Transfer Object (DTO) which contains information about the location, including the identifier of the location.

### Interface

SOAP method: `void ModifyLocation (LocationDTO location)`

#### Input

LocationDTO location	Information about the location to be modified. See GetAllLocations for a description of the DTO. Notes on parameter usage: <ul style="list-style-type: none"><li>• LocationId: Must be valid and cannot be changed.</li><li>• LocationPath: Must be valid and cannot be changed.</li></ul>
----------------------	--

#### Output

(none)	n/a
--------	-----



## Method: GetFieldVisitsByLocation

This method is used to retrieve a list of all field visits for a given location identifier.

### Interface

SOAP method: `FieldVisit[] GetFieldVisitsByLocation (long locationId)`

#### Input

long locationID	Numeric identifier of the location for which to retrieve field visits.
-----------------	--

#### Output

FieldVisit[] fieldVisits	Array of field visits for the location that is specified via the locationId parameter. Returns null if no field visits are found.
--------------------------	---



## Method: GetFieldVisitsByLocationChangedSince

This method is used to retrieve a list of all field visits for a given location identifier, which have changed since a given time.

The changedSince time is relative to the server local time.

The “changedSince” criteria refer to any part of the field visit (including the field visit itself, as well as any contained activities, observations or results).

### Interface

SOAP method: `FieldVisit[] GetFieldVisitsByLocationChangedSince (long locationId, string changedSince)`

#### Input

long locationID	Numeric identifier of the location for which to retrieve field visits. If locationId is negative or equals 0, an exception is thrown. If the specified locationId does not exist, an empty list is returned.
String changedSince	Last modified time for a field visit. Must not be null or empty. The field visits modified on or after the time will be retrieved. The format is of ISO time format ("yyyy-MM-ddTHH:mm:ss.fffz" , e.g., "1994-11-05T08:15:30.000-05:00" corresponds to November 5, 1994, 8:15:30 AM, US Eastern Standard Time). An exception is thrown if the input is of a bad format or has invalid values.

#### Output

FieldVisit[] fieldVisits	Array of field visits modified on and after the specified time for the specified location. Returns an empty array if no field visits are found.
--------------------------	---



## Method: GetFieldVisitsByLocationAndDate

This method is used to retrieve a list of field visits for a particular date and location identifier.

### Interface

SOAP method: `FieldVisit[] GetFieldVisitsByLocationAndDate (long locationId, DateTime startDate)`

#### Input

<code>long locationId</code>	Numeric identifier of the location for which to retrieve field visits.
<code>DateTime startDate</code>	Date and time indicating when the field visit was started.

#### Output

<code>FieldVisit[] fieldVisits</code>	Array of field visits that match the specified criteria. Returns an empty array if no field visits are found.
---------------------------------------	---



## Method: SaveFieldVisit

This method is used to save field visit data. Saving encompasses insertion, deletion and updates of field visit records. Please refer to the description of the fv input parameter in Appendix A. For an example of how to enter a field visit in code (C#), refer to Appendix B 'Field Visit Creation Example (C#)'.

### Interface

SOAP method: `FieldVisit SaveFieldVisit (FieldVisit fv)`

#### Input

Field visit to save. FieldVisit object is a compound object.

To insert a new field visit set the FieldVisitId to 0.

To delete an existing field visit set the FieldVisitId to its negative counterpart (i.e. 100 would become -100).

To update an existing field visit leave its FieldVisitId unchanged.

FieldVisit fv

To insert a new field visit measurement set its MeasurementID to 0.

To delete an existing field visit measurement set the MeasurementID to its negative counterpart (i.e. 100 would become -100).

To update an existing field visit measurement leave its MeasurementID unchanged.

To insert a new field visit result set its ResultID to 0.

To delete an existing field visit result set ResultID to its negative counterpart (i.e. 100 would become -100).

To update an existing field visit result leave its ResultID unchanged.

#### Output

FieldVisit fieldVisit

Copy of the updated object. Returns NULL if the field visit was deleted.

## Method: CreateTimeSeries

This method is used to create a new time series in the AQUARIUS system.

*This method will be deprecated in a future release. Use the newer method CreateTimeSeries2, as it allows better control of metadata such as units, parameters, etc.*

### Interface

SOAP method: `long CreateTimeSeries (string identifier)`

#### Input

String identifier	Text identifier of the time series to create. Note that this is the full text identifier which contains the parameter, label, and parent location. For example "HG.Telemetry@02AB014"
-------------------	---

#### Output

long timeSeriesID	Numeric identifier of the created time series.
-------------------	--



## Method: CreateTimeSeries2

This method is used to create a new time series in the AQUARIUS system.

### Interface

SOAP method: `long CreateTimeSeries2(long parentId, string label, string comment, string description, string parameter, int utcOffsetInMinutes, string unit, double maxGap)`

#### Input

long parentId	Numeric identifier of the “parent” location.
string label	Text label of the time series to create. This will be used as part of the time series identifier and therefore should not be blank. For example “Telemetry”.
string comment	Comments for the time series to create. May be blank.
string description	Text description for the time series to create. May be blank.
string parameter	Name of the parameter for the new time series to create. This must correspond with the list of parameters in the AQUARIUS system, for example “HG”.
int utcOffsetInMinutes	Time zone offset from UTC, in minutes. For example, PST would have a value of -480.
string unit	Name of the unit for the new time series. This must correspond with the list of units in the AQUARIUS system, for example “m”.
double maxGap	Max gap tolerance (in units of days). A value of 0 means no gap processing.

#### Output

long timeSeriesID	Numeric identifier of the created time series.
-------------------	--



## Method: GetTimeSeriesID

Get the numeric identifier of a time series given its text identifier.

### Interface

SOAP method: `long GetTimeSeriesID(string identifier)`

#### Input

String identifier

Text identifier of the requested time series. Note that this is the full text identifier which contains the parameter, label, and parent location, for example "HG.Telemetry@02AB014".

#### Output

long timeSeriesID

Numeric identifier of the time series. 0 if not found. In case there are multiple time series found with the specified identifier, an exception is thrown.



## Method: GetTimeSeriesID2

Get the numeric identifier of a time series given its text label and parent label.

### Interface

SOAP method: `long GetTimeSeriesID2(string parentLabel, string identifier, string parameterType)`

#### Input

String parentLabel	Text name of the “parent” location. Note that this is the full text name as seen in Springboard in the “Location Name” column.
String identifier	Text label of the time series. For example “Telemetry”. Note that this is <i>not</i> the full text identifier.
String parameterType	Parameter type of the time series. For example “HG”. May be set to null or empty if this is not to be used in the criteria for identifying the time series.

#### Output

long timeSeriesID	Numeric identifier of the time series. The function will return 0 if the requested time series cannot be found. In case there are multiple time series found with the specified identifier, an exception is thrown.
-------------------	---



## Method: GetTimeSeriesList

This methods returns a list of time series description objects at a given location for a given parameter type.

### Interface

SOAP method: `TimeSeriesDescription[] GetTimeSeriesList(long locationID, string parameterType)`

#### Input

long locationID	Numeric identifier of the location for which to retrieve a list of time series.
String parameterType	Type of the parameter. For example “HG”.

#### Output

TimeSeriesDescription[]	An array of TimeSeriesDescription objects. Each object contains summary information about the time series including its AQDataID, total samples, end time and end value.
-------------------------	--



## Method: GetTimeSeriesListForLocation

This method returns a list of time series description objects at a given location.

### Interface

SOAP method: `TimeSeriesDescription[] GetTimeSeriesListForLocation(long locationID)`

#### Input

`long locationID` Numeric identifier of the location for which to retrieve a list of time series.

#### Output

`TimeSeriesDescription[]` An array of `TimeSeriesDescription` objects. Each object contains summary information about the time series including its `AQDataID`, total samples, end time, and end value.



## Method: AppendTimeSeriesFromBytes

This method is used to append new data to an existing time series in the AQUARIUS system.

### Interface

SOAP method: `int AppendTimeSeriesFromBytes(long id, byte[] csvbytes, string userName, string comment)`

#### Input

long id	Numeric identifier of the time series.
byte[] csvbytes	<p>Bytes representing data to append to the time series. The format is as follows:</p> <p>YYYY-MM-DD HH:MM:SS, nnn.mmm, fff, ggg, iii, aaa, note</p> <p>Optional data may be omitted by leaving it blank. Commas can be omitted if there are no fields on the rest of the line.</p> <p>Each point should be separated by a newline ('\n') character.</p> <p>Where:</p> <ul style="list-style-type: none"><li>• YYYY-MM-DD HH:MM:SS represent the timestamp of the point, relative to the UTC offset of the given time series. Note that timestamps should be in order (time increasing), and not contain any duplicates.</li><li>• nnn.mmm represents the value of the point in decimal format (optional – if the value is omitted then AQUARIUS will show a timestamp with no value, i.e. an “Empty” point).</li><li>• fff represents a numeric flag value (optional).</li><li>• ggg represents a numeric grade value (optional).</li><li>• iii represents a numeric interpolation code (optional).</li><li>• aaa represents a numeric approval code (optional).</li><li>• “note” represents a text note which can be attached to the point (optional).</li></ul> <p>For example, the following string indicates value= 2.8, flag=192, approval code=1; grade, interpolation type and note are unspecified:</p>



2011-01-17 23:52:33, 2.8, 192, , , 1

String userName

Username to be associated with this append.

String comment

Comment to be associated with this append. Currently this is not used.

### Output

int numPointsAppended

Number of points appended.



## Method: AppendTimeSeriesFromBytes2

This method is used to append new data to an existing time series in the AQUARIUS system. It returns an AppendResult which includes an undo token. This undo token can be used in a subsequent call to UndoAppend.

### Interface

SOAP method: `AppendResult AppendTimeSeriesFromBytes2(long id, byte[] csvbytes, string userName)`

#### Input

long id	Numeric identifier of the time series.
byte[] csvbytes	Bytes representing data to append to the time series. The format is as described for AppendTimeSeriesFromBytes.
String userName	Username to be associated with this append.

#### Output

AppendResult appendResult	An object containing information about the append, including append token (can be used for undo), number of points appended and text time series identifier.
---------------------------	--

## Method: AppendAndMerge

(This method has been deprecated and will not be supported in future releases of AQUARIUS).



## Method: DeleteTimeSeriesPointsByTimeRange

This method is used to delete a range of points from a given time series. The points to delete are identified by the time range.

This operation cannot be reversed. The points will be permanently deleted from the time series, and cannot be restored.

### Interface

SOAP method: `int DeleteTimeSeriesPointsByTimeRange(long id, System.DateTime startTime, System.DateTime endTime)`

#### Input

<code>long id</code>	Numeric identifier of the time series.
<code>System.DateTime startTime</code>	Start date/time of the time range to delete.
<code>System.DateTime endTime</code>	End date/time of the time range to delete.

#### Output

<code>Int numPointsDeleted</code>	Number of points deleted.
-----------------------------------	---------------------------



## Method: UndoAppend

This method is used to undo a previous append, deleting the points associated with that append. The points to delete are identified by an “append token” which is returned from AppendTimeSeriesFromBytes2. Only appends performed via AppendTimeSeriesFromBytes2 can be undone.

This operation cannot be reversed. The points will be permanently deleted from the time series, and cannot be restored.

### Interface

SOAP method: `int UndoAppend(string identifier, string appendToken)`

#### Input

String identifier	Text identifier of the time series.
String appendToken	Append token used to identify which append to un-do. This is the token returned by the corresponding call to AppendTimeSeriesFromBytes2.

#### Output

Int numPointsDeleted	Number of points deleted.
----------------------	---------------------------



## Method: DeleteTimeSeries

This method deletes a given time series.

This operation cannot be reversed. The time series will be permanently deleted from the database, and cannot be restored.

### Interface

SOAP method: `void DeleteTimeSeries(long timeSeriesId)`

#### Input

long timeSeriesId	Text identifier of the time series.
-------------------	-------------------------------------

#### Output

(none)	n/a
--------	-----



## Method: IsConnectionValid

This method checks if authentication token is still valid for the created connection.

### Interface

SOAP method:

```
bool IsConnectionValid()
```

### Input

<none>

### Output

bool

Returns true if the token is still valid



## Method: KeepConnectionAlive

This method keeps authentication token valid for the created connection.

### Interface

SOAP method:

```
bool GetReportData()
```

REST method:

```
/IsConnectionValid?token={string}
```

### Input

<none>

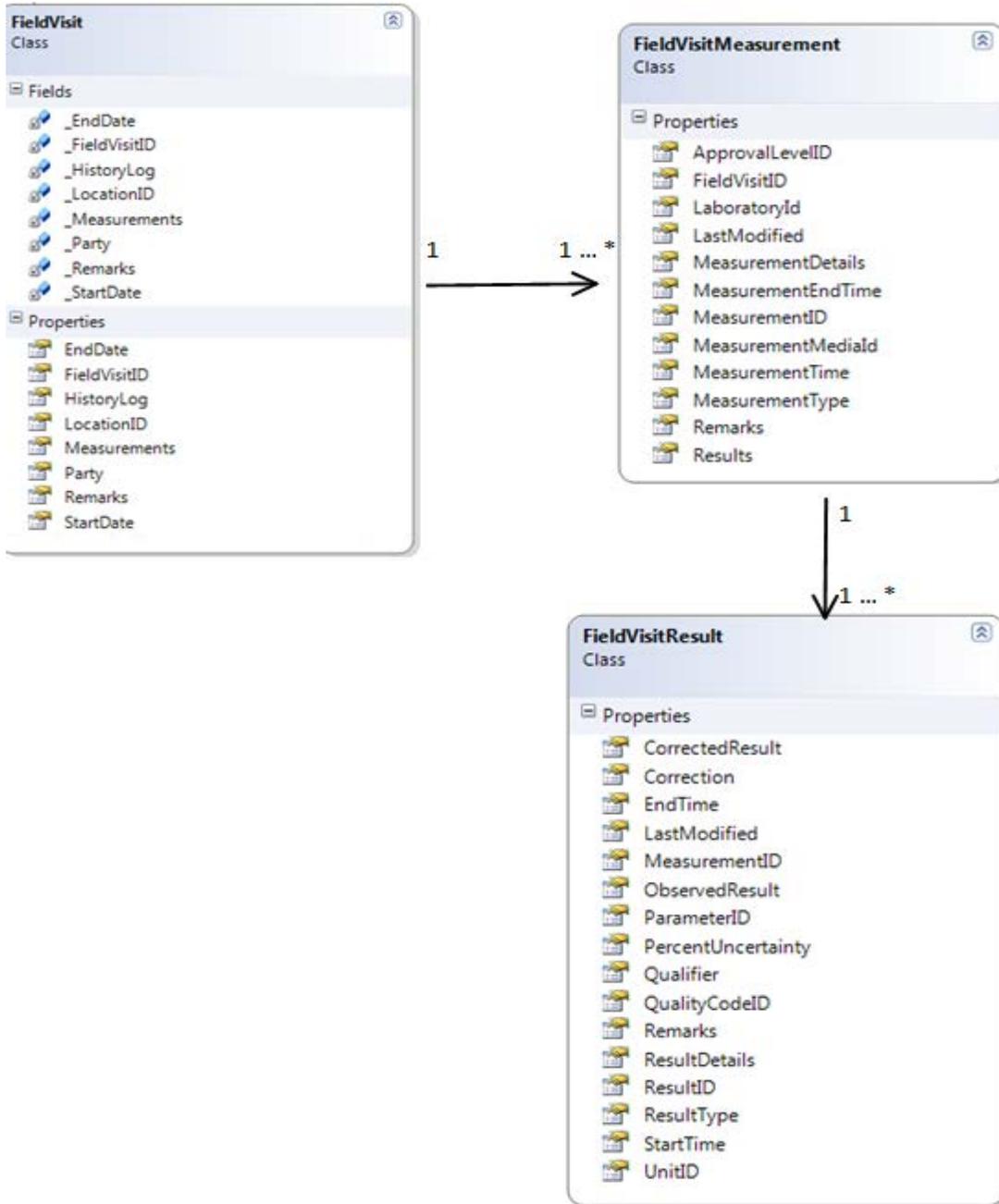
### Output

bool

Returns true if the token is still valid



## Appendix A: Field Visit Data Model



### SOAP Client Example: Authentication (C#)

This example shows sample code for using the SOAP interface. If you are using Visual Studio you can create the client proxy class by using the “Add Service Reference” feature. Otherwise you can generate a proxy class from the WSDL some other way (not shown). In AQUARIUS 3.3, and newer versions, the WSDL indicates that the authentication is required in the SOAP header, so you no longer have to manually add it to the header. In .NET the authentication token must be passed as the first parameter of each method. Other frameworks/languages may use an alternate means of setting the token, depending on the implementation.

```
//Add web reference
AquariusAcqServiceClient client = new AQAcquisitionServiceClient (
    "WSHttpBinding_IAQAcquisitionService",
    "http://<mysrvrname>/AQUARIUS/AQAcquisitionService.svc");

//Get token
string authToken = client.GetAuthToken("dummy_user", "dummy_pw");

// Call the method
Client.GetTimeSeriesID(authToken, time_series_identifier);
```

### Field Visit Creation Example (C#)

The example code excerpt below shows how one can create a field visit in AQUARIUS using C#:

```
FieldVisit fv = createFieldVisit(
    prog.GetLocationID("AQW1121"),
    0,
    DateTime.Parse("January 1, 2008"),
    DateTime.Parse("January 1, 2008"),
    "Bob and Doug MacKenzie",
    0);

FieldVisit myFieldVisit = client.SaveFieldVisit(authToken, fv);

public static FieldVisit createFieldVisit(long locationId, int FVID, DateTime start, DateTime?
end, string party, Int32 measID)
{
    //Create the field visit that will contain the measurement(s)
    FieldVisit visit = new FieldVisit();
    visit.LocationID = locationId;
    visit.FieldVisitID = FVID;
    visit.StartDate = start;
    visit.EndDate = end;
    visit.Party = party;
    visit.Remarks = "";

    //Create a measurement that will contain the observations
    var measurement = new FieldVisitMeasurement
```



```
{
    MeasurementID = measID,
    FieldVisitID = visit.FieldVisitID,
    //set the "magic string" that tells the generic
    //field visit plug in to display this measurement
    MeasurementType = "AqFvtPlugin_MeasurementActivity",
    MeasurementTime = visit.StartDate,
    MeasurementEndTime = visit.EndDate
};

//Create a measurement ('stage' in this example)
var stage = new FieldVisitResult
{
    MeasurementID = measurement.MeasurementID,
    ResultID = 0,
    StartTime = visit.StartDate,
    ParameterID = "HG",
    UnitID = "m",
    ObservedResult = 5.0
};

//Create another measurement ('discharge' in this example)
var discharge = new FieldVisitResult
{
    MeasurementID = measurement.MeasurementID,
    ResultID = 0,
    StartTime = visit.StartDate,
    ParameterID = "QR",
    UnitID = "m^3/s",
    ObservedResult = 35.0
};

// set measurement results
measurement.Results = new[] { stage, discharge };

//Create an observation ('stage' in this example)
var stageObservation = new FieldVisitResult
{
    MeasurementID = measurement.MeasurementID,
    ResultID = 0,
    StartTime = visit.StartDate,
    ParameterID = "HG",
    UnitID = "m",
    ObservedResult = 3.0
};

//Create another observation ('discharge' in this example)
var dischargeObservation = new FieldVisitResult
```



```
{
    MeasurementID = measurement.MeasurementID,
    ResultID = 0,
    StartTime = visit.StartDate,
    ParameterID = "QR",
    UnitID = "m^3/s",
    ObservedResult = 16.0
};

//set the measurement observations
measurement.Observations = new[] { stageObservation, dischargeObservation };

visit.Measurements = new[] { measurement };

return visit;
}
```