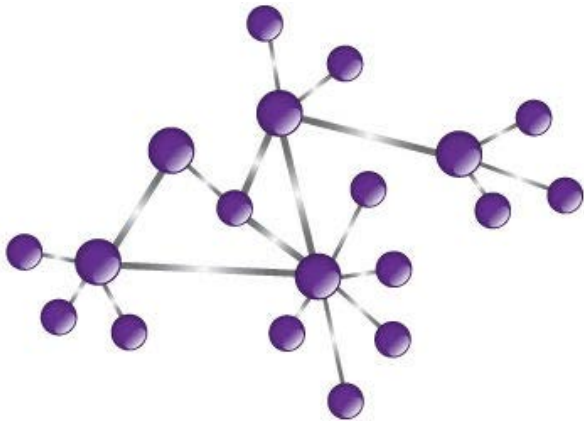
WebEOC®

API Technical Guide

Version 7.3 and higher



ESi Acquisition, Inc.

For ESi sales and operations:
ESi Acquisition, Inc.
823 Broad Street, Augusta, Georgia 30901-1400
USA
Tel: (706) 823-0911 or (800) 596-0911
Fax: (706) 826-9911
sales@esi911.com

http://www.esi911.com/

Revision History

Revision 0	February 11, 2008
Revision 1	October 26, 2011

Introduction

WebEOC 7 introduces application programming interface (API) functionality for third-party applications to exchange data with WebEOC. The interface is built from a group of widely adopted standards that are natively supported by many mainstream development environments.

This technical guide is intended for software developers to make use of the WebEOC API, and assumes the developer is skilled in basic object oriented programming concepts and has an advanced, administrator level understanding of WebEOC.

Overview

The API is a web service implemented on the Simple Object Access Protocol (SOAP) specification. A Web Services Description Language (WSDL) file exists for code generation utilities to automatically create helper proxy classes and the underlying SOAP bindings. Code generation utilities exist for many popular development environments or languages (e.g. Java, .NET, Python, Ruby). An example of code generation for .NET will be included below in the walkthrough section.

The following methods are currently available with the WebEOC version 7.3 or higher edition of the API. Examples of specific usage will be given in the walkthrough.

Setup

- Using the WebEOC Credentials object

Incidents

- **AddIncident** – Adds an incident to WebEOC. Requires administrator credentials for Incidents.
- **GetIncidents** – Get a list of all the WebEOC incidents available to a particular user.

Boards

- **AddData** – Add a new entry to a WebEOC board.
- **UpdateData** – Update an existing entry in a WebEOC board.
- **AddAttachment** – Attaches a file to an existing board entry.
- **UpdateDataBatch** – Adds or updates any number of record entries at one time in a WebEOC board.
- **AddRelatedData** - Add a new entry to a WebEOC board that is related to a record in a different table.
- **UpdateRelatedData** – Update an existing entry in a board that is related to a record in a different table.
- **GetData** – Get all the entries contained within a WebEOC board.
- **GetFilteredData** – Get the entries contained within a WebEOC board that meet certain requirements.
- **GetDataById** – Retrieve a single, specific, entry from a WebEOC board.
- **PostStandardData** – Currently not used with core WebEOC. Reserved for future interfaces posting specific standardized data that must be transformed before adding to a WebEOC board.
- **GetBoardNames** – Get a list of all the boards in WebEOC. Requires administrator credentials.
- **GetInputViews**– Get a list of the input views for a particular board. Requires administrator credentials.
- **GetDisplayViews** – Get a list of the display views for a particular board. Requires administrator credentials.
- **GetViewFields** - Returns a list of all the fields on an input or display view. Requires administrator credentials.

Lists

- **GetLists** – Get a *DataTable* of lists. Requires administrator credentials.
- **GetListItems** – Get a *DataTable* of list items in a WebEOC list. Requires administrator credentials.
- **GetListItemsByListName** – Gets a *DataTable* of list items in a WebEOC list that a specific user has access to. Requires administrator credentials.
- **SaveList** – Save or rename a WebEOC list. Requires administrator credentials.
- **SaveListItem** – Save or rename a WebEOC list item. Requires administrator credentials.
- **DeleteList** – Delete a WebEOC list. Requires administrator credentials.
- **DeleteListItem** – Delete a WebEOC list item. Requires administrator credentials.

WebEOCUser Object

The WebEOCUser object encapsulates information describing a single WebEOC user. Properties of this object are listed in the description area.

User and Position methods

Some of the user and position methods use the WebEOCUser object from above.

- **AddUser** – Adds a new user to WebEOC. Requires administrator credentials.
- **DeleteUser** – Deletes a user from WebEOC. Requires administrator credentials.
- **DeleteUserIfNoPositions** – Deletes a user if it is not assigned to any positions. Requires administrator credentials.
- **AddUserToPosition** – Assigns a user to a position. Requires administrator credentials.
- **RemoveUserFromPosition** – Un-assigns a user from a position. Requires administrator credentials.
- **GetPositions** – Returns all of the positions contained within WebEOC. Requires administrator credentials.
- **GetPositionsForUser** – Returns the positions assigned to a user. Requires administrator credentials.
- **GetUsersByPosition** – Returns the users that belong to a particular position. Requires administrator credentials.

Miscellaneous methods

- **Ping** – Tests connectivity to the WebEOC instance and correctness of the credentials.

Walkthrough

In the following sections, each of the steps involved with using the web service and specific examples of calling each of the web methods will be covered. The Shelters and Press Release boards will be used in some of the examples below. These boards can be obtained from running WebEOC board installers included with WebEOC versions prior to version 7.4.

Note: in the examples below, the requirement to be a "Lists administrator" means that the user must be assigned at least the "Lists" admin profile with the level of list permissions specified, **if** it is not an administrator. An administrator automatically has full access to all admin managers.

Automatic Code Generation

A WSDL file describing the WebEOC API web service is available in the root of the WebEOC virtual directory, so, if the WebEOC server name is MyWebEOCServer and WebEOC is running in the virtual directory /eoc7, the path to the WSDL would be http(s)://MyWebEOCServer/eoc7/api.asmx?wsdl.
Many popular languages have utilities for generating SOAP proxy classes by inspecting the WSDL.

The .NET code generator is a command line executable called WSDL.exe and is distributed with the .NET SDK. With the above WSDL path, WSDL.exe would be invoked like this:

wsdl /out:WebEOCServiceAPI.cs http://MyWebEOCServer/eoc7/api.asmx?wsdl

Alternatively, proxy classes can be generated directly within Visual Studio by right clicking on the project and selecting "Add Web Reference."

Setup

Using the WebEOCCredentials object

Authentication values for the web service are passed through the WebEOCCredentials object. This object contains all the information WebEOC needs to verify if it should perform a requested operation. Generally, the credentials object contains all of the information that a user would have to supply when logging into WebEOC. The credentials object is a required parameter for all of the web methods and contains the following properties:

Username (Required) – the WebEOC user to perform the requested operation as
Password (Required) – the password for the user specified by Username
Position (Required) – the position that the WebEOC user is assigned to
Incident (Conditionally required) – the WebEOC incident to log into while performing the requested operation
Jurisdiction (Optional) – the WebEOC jurisdiction to log into while performing the requested operation

The API follows the standard WebEOC login process. Thus, values contained within the credentials object cannot circumvent normal WebEOC permissions. For example, the user in Username must have access to the named incident and cannot be currently locked out for the operation to succeed.

Creating the WebEOCCredentials Object

```
//Instantiate the WebEOCCredentials object
WebEOCCredentials credentials = new WebEOCCredentials();

//Set our values
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";
credentials.Incident = "Downtown Flood";
credentials.Jurisdiction = "Chatham County";
```

Incidents

Adding an incident

The AddIncident method adds an incident to WebEOC. This method has several parameters.

WebEOCCredentials – user credentials (must be an Incidents administrator with at least Edit permissions)
IncidentName – the name of the incident
DateTime – an object of type *DateTime* that specifies the incident's creation time
Default – a Boolean value specifying whether the incident is the default incident or not

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Add an incident to WebEOC, set to June 1, 2011,
//not set as the default incident
api.AddIncident(credentials, "incident1",
                    new DateTime(2011, 6, 1), false);
```

Getting the Incident List

One of the simplest API operations is retrieving the WebEOC incident list for a particular user. This method has only a single parameter, a **WebEOCCredentials** object, populated with the user info to retrieve the incident list for. It will return an array of incident names. These incident names are valid possible values for the Incident property for any other calls to the web service.

```
//Create the api service object.
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Retrieve the incident array for the user contained in the WebEOCCredentials
object.
string[] incidents = api.GetIncidents(credentials);
```

Boards

Adding a New Board Entry

The AddData method is used to add a new entry to a board. In this example, a new shelter will be added to the Shelters board. AddData has the following parameters:

WebEOCCredentials – user credentials
BoardName – the name of the board to add the data to
InputViewName – the input view to use when adding the data
XmlData – a string of xml data that contains the values to be added to the board

This method returns an integer value called the dataid which is the database primary key for the newly added record. The return value will be used later for updating the new record. The xml data string should contain a root element called data with children elements matching the WebEOC input view field names. The input view for shelters has the following fields:

name	specialneeds
address	petfriendly
county	title
capacity	contactname
status	Contactnumber
arc	Remarks

The input xml data string should look similar to this:

```
<data>
        <name>City High School</name>
        <address>1201 Main St.</address>
        <county>Chatham</county>
        <capacity>75</capacity>
        <status>OPEN</status>
        <arc>Yes</arc>
        <specialneeds>Yes</specialneeds>
        <petfriendly>No</petfriendly>
        <title>Director</title>
        <contactName>Jack Jones</contactName>
        <contactNumber>555-5555</contactNumber>
        <Remarks>here are some comments</Remarks>
</data>
```

Not all of the fields have to be included in the xml string. Any fields not in the string will remain empty in the new record.

```
string boardName = "Shelters";
string inputViewName = "Input";

//Build the xml data string
string xmlData = "<data>" +
                    "<name>City High School</name>" +
                    "<address>1201 Main St.</address>" +
                    "<county>Chatham</county>" +
                    "<capacity>75</capacity>" +
                    "<status>OPEN</status>" +
                    "<arc>Yes</arc>" +
                    "<specialneeds>Yes</specialneeds>" +
                    "<petfriendly>No</petfriendly>" +
                    "<title>Director</title>" +
                    "<contactName>Jack Jones</contactName>" +
                    "<contactNumber>555-5555</contactNumber>" +
                    "<Remarks>here are some comments</Remarks>" +
                "</data>";

//And post to WebEOC
int dataid = api.AddData(credentials, boardName, inputViewName,
xmlData);
```

Updating an Existing Board Entry

Existing records can be modified using the UpdateData method. In this example, the shelter record created in the AddData example will be updated. UpdateData takes the same parameters as AddData plus one additional one. It also needs the dataid which was returned by AddData. This is how UpdateData determines which record to update. The xml data string is structured the same as for AddData and should supply all the fields contained on the view. This means that if you specify all twelve fields specified in the prior AddData example, but only specify four of those fields in the XML data string passed to UpdateData, the eight unspecified fields will be overwritten with empty data.

In the example below the status and contactNumber fields are updated to the newly specified values, and all other fields remain the same.

```
string boardName = "Shelters";
string inputViewName = "Input";

//Build the xml data string that will be used
//to update the existing record
string xmlData = "<data>" +
                   "<name>City High School</name>" +
                   "<address>1201 Main St.</address>" +
                   "<county>Chatham</county>" +
                   "<capacity>75</capacity>" +
                   "<status>CLSD</status>" +
                   "<arc>Yes</arc>" +
                   "<specialneeds>Yes</specialneeds>" +
                   "<petfriendly>No</petfriendly>" +
                   "<title>Director</title>" +
                   "<contactName>Jack Jones</contactName>" +
                   "<contactNumber>444-4444</contactNumber>" +
                   "<Remarks>here are some comments</Remarks>" +
                 "</data>";

//And post to WebEOC
api.UpdateData(credentials, boardName, inputViewName, xmlData, dataid);
```

Adding an attachment to a post

The AddAttachment method allows an attachment to be added to a pre-existing post. The "PIO Admin Input" view of the Press Release board has an attachment field in it, and that input view is used for the example below. The AddAttachment method has several parameters:

WebEOCCredentials – the credentials of any WebEOC user with access to the board
BoardName – the name of the WebEOC board
InputViewName – the name of the WebEOC input view
AttachmentFieldName – the name of the attachment field. This can be found on the board input view editor.
Dataid – The dataid of the post that you are adding the attachment to. The post must have already been made prior to the AddAttachment call.
AttachmentFileData – the file converted into a byte array.
AttachmentFileName – This is the default filename provided to a user when the user saves the file attachment to a local drive.

```
//Board with attachment field
string board = "Press Release",
        inputViewName = "PIO Admin Input",
        AttachmentFieldName = "press_release",
        filePath = @"C:\test.jpg";

//dataid of the board post
int dataid = 1;

//Setup to get image in correct format for API call
MemoryStream ms = new MemoryStream();
Image myImage = Image.FromFile(filePath);
myImage.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);
byte[] image = ms.ToArray();

//API reference and credentials
APIExample.WebEOCAPI.API api = new API();
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";
credentials.Incident = "Setup";

//Call API, adding attachment
api.AddAttachment(credentials, board, inputViewName,
                    AttachmentFieldName, dataid, image, "filename.jpg");
```

Making or updating multiple posts at once

The UpdateDataBatch method allows one or more posts to be added or updated in WebEOC at one time. The method has several parameters.

WebEOCCredentials – any user with access to the board through their group.
BoardName – the name of the board
InputViewName – the name of the input view that the XML data is used in.
XmlData – one or more records in the form below, where the <record></record> section can be repeated multiple times.

```
<data>
    <record>
        <xmltag1>data1</xmltag1>
        <xmltag2>data2</xmltag2>
        <xmltag3>data3</xmltag3>
        …
    </record>
    …
</data>
```

Below is the main code and supporting functions. Note that the third parameter of the buildXMLDataString function takes an integer, which is the dataid of the record to update. If a zero or a negative number is supplied (as below), a new record is made. If the dataid of a pre-existing record is supplied, then that existing record is updated with the new data.

Note: the *Main code* section below can add one or more records to a board, or update one or more records already in a board. The choice of two records to create is done as an example.

Note: in the example below, the board being used (Shelters) is by default an incident-independent board. If a board in use is incident-independent, the WebEOCCredentials object can specify an incident, but if it is incident-dependent, the WebEOCCredentials object **must** specify an incident.

Main code:

```
//Create the api service object
APIExample.WebEOCAPI.API apiReference = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";
credentials.Incident = "Setup";

//specify board and input view name
string boardName = "Shelters";
string inputViewName = "Input";

//Make two new XML strings, merge them into a single XML string
String[] concatenatedXMLData = new String[2];
for (int i = 0; i < concatenatedXMLData.Length; i++)  {
    concatenatedXMLData[i] =
        XMLBoardParsing.buildXMLDataString(XMLBoardParsing.UNIQUE_SHELTERS_XML, i, -1);
}
String xmlData = mergeXMLData(concatenatedXMLData);

//display XML data to be added
System.Console.WriteLine(xmlData);

//Add the two new records
int[] resultDataIds =
    apiReference.UpdateDataBatch(credentials, boardName, inputViewName, xmlData);

//Display the retrieved record data.
//A display view name has to be added for use by GetDataByDataId
String displayViewName = "Details";
System.Console.WriteLine("The records retrieved are: ");
System.Console.WriteLine(apiReference.GetDataByDataId(credentials, boardName,
                                                       displayViewName, resultDataIds[0]));
System.Console.WriteLine(apiReference.GetDataByDataId(credentials, boardName,
                                                       displayViewName, resultDataIds[1]));
```

Supporting Code:

Builds a single XML record based on the parameters provided. The code example below generates XML for the Shelters board. The dataid is only included in the generated XML when this is an existing record (i.e. a positive dataid). Unique records can be made by passing in unique integers for *appendText*.

```
public static string buildXMLDataString(int type, int appendText, int dataID)
{
    // if a dataID is greater than 0, add a dataid including it
    String appendDataID = "";
    if (dataID > 0)
        appendDataID = "<dataid>" + dataID + "</dataid>";

    switch(type)
    {
        case UNIQUE_SHELTERS_XML:
            string xmlCustomData =
                "<data>" +
                   "<record>" +
                        appendDataID +
                        "<label>Label" + appendText + "</label>" +
                        "<address>Address" + appendText + "</address>" +
                        "<capacity>75</capacity>" +
                        "<status>OPEN</status>" +
                        "<arc>Yes</arc>" +
                        "<specialneeds>Yes</specialneeds>" +
                        "<petfriendly>Yes</petfriendly>" +
                        "<title>Director" + appendText + "</title>" +
                        "<contactname>Contact" + appendText + "</contactname>" +
                        "<contactnumber>Number" + appendText + "</contactnumber>" +
                        "<remarks>Remark" + appendText + "</remarks>" +
                   "</record>" +
                "</data>";
            return xmlCustomData;

        ...
```

Supporting code:

Merges any number of XML records together

```
private String mergeXMLData(String[] separateRecords)
{
    String returnedData;
    // strip out beginning and ending data tags
    for (int i = 0; i < separateRecords.Length; i++)
    {
        separateRecords[i] = separateRecords[i].Replace("<data>", "");
        separateRecords[i] = separateRecords[i].Replace("</data>", "");
    }

    // build new string with a single starting and ending data tag
    returnedData = "<data>";
    for (int i = 0; i < separateRecords.Length; i++)
    {
        returnedData += separateRecords[i];
    }
    returnedData += "</data>";

    // return XML string
    return returnedData;
}
```

Adding a Related Board Entry

The AddRelatedData method adds a new record to a board that should be related to an existing record. In this example, a new resident record will be added to an existing shelter. In the Shelters board used below (this is not the Shelters board distributed with the pre-7.4 WebEOC board installers), the input view for adding new shelter residents (Sheltered) has a foreignkey tag to the Shelters table. So each new entry in the Sheltered table should be related to an existing record in the Shelters table. Parameters for AddRelatedData are very similar to AddData. There are two additional parameters for indicating which table and dataid to relate to.

WebEOCCredentials – user credentials
BoardName – the name of the board to add the data to
InputViewName – the input view to use when adding the data
XmlData – a string of xml data that contains the values to be added to the board
RelatedTable – the table that the new record will be related to
RelatedDataId – the data id of an existing record that the new record will be related to

```
//The board that we are adding data to.
string boardName = "Shelters";

//The input view that we are adding data with.
string inputViewName = "Sheltered";

//The table to relate to.  Our new resident record should be related to an
entry in the Shelters Table.
string sheltersTable = "Shelters Table";

//We should already know the data id of the shelter to add to.  Either saved
//from initially adding the shelter or retrieved through GetData or
//GetFilteredData.
int shelterDataId = 1;

//Build the xml data string
string xmlData = "<data>" +
                    "<Last_Name>Jones</Last_Name>" +
                    "<First_Name>Mickey</First_Name>" +
                    "<dob>03-21-58</dob>" +
                "</data>";

//Post the new record.  The data id for our new resident record will be
//returned.
newDataId = api.AddRelatedData(credentials, boardName, inputViewName,
                                xmlData, sheltersTable, shelterDataId);
```

Updating a Related Board Entry

The UpdateRelatedData method modifies a record that has been related to an existing entry through AddRelatedData. In this example, an existing shelter resident will be moved to a different shelter. UpdateRelatedData takes the same parameters as AddRelatedData.

```
//The board that we are modifying.
string boardName = "Shelters";

//The input view that we are using.
string inputViewName = "Sheltered";

//The table to relate to.  Our resident record should be related to an entry
in the Shelters Table.
string sheltersTable = "Shelters Table";

//We should already know the data id of the shelter to move to.  Either saved
//from initially adding the shelter or retrieved through GetData or
//GetFilteredData.
int shelterDataId = 2;

//Our updated data is empty. We're not changing the resident's personal data.
//Just which shelter record they are related to.
string xmlData = "<data></data>";



//Post the change.
api.UpdateRelatedData(credentials, boardName, inputViewName,
                                  xmlData, sheltersTable, shelterDataId);
```

Retrieving Board Data

The GetData method returns an xml string of the data contained within a board, based on a particular display view. If the board is incident independent, all data will be returned. If the board is incident dependent, only the records entered through the incident passed through the WebEOC credentials object will be returned. GetData accepts three parameters:

WebEOCCredentials – user credentials
BoardName – the board to retrieve data from
DisplayViewName – the specific display view to retrieve data with.

```
//The board that we are using.
string boardName = "Shelters";

//The display view that we are using.
string displayViewName = "List";

//Get the data.
string results = api.GetData(credentials, boardName, displayViewName);
```

Example returned XML:

```
<data>
	<record>
		<name>City High School</name>
		<address>1201 Main St.</address>
		<county>Chatham</county>
		<capacity>75</capacity>
		<status>OPEN</status>
		<arc>Yes</arc>
		<specialneeds>Yes</specialneeds>
		<petfriendly>No</petfriendly>
		<title>Director</title>
		<contactName>Jack Jones</contactName>
		<contactNumber>555-5555</contactNumber>
		<Remarks>here are some comments</Remarks>
	</record>
	<record>
		<name>St. Mary's Prep</name>
		<address>638 East Bay St.</address>
		<county>Chatham</county>
		<capacity>125</capacity>
		<status>OPEN</status>
		<arc>No</arc>
		<specialneeds>Yes</specialneeds>
		<petfriendly>No</petfriendly>
		<title>Director</title>
		<contactName>Shelly McTavish</contactName>
		<contactNumber>555-5555</contactNumber>
		<Remarks>here are some remarks</Remarks>
	</record>
</data>
```

Retrieving Filtered Board Data

The GetFilteredData method retrieves board data similar to GetData but allows filtering parameters to be specified for the returned data. GetFilteredData supports two filter types. If a view filter has been created within WebEOC, that filter can be invoked through the GetFilteredData. Alternatively, an XML string can be passed to GetFilterData to allow dynamic filtering. View filters and a user filter can be used in conjunction. GetFilteredData accepts five parameters:

WebEOCCredentials – user credentials
BoardName – the board to retrieve data from
DisplayViewName – the specific display view to retrieve data with.
ViewFilterNames – an array of any view filters to apply to the returned data
XmlUserFilter – an xml string of data to filter by.

The following two examples call GetFilteredData.

GetFilteredData Using a View Filter.

In this example, we will retrieve all of the open shelters in the shelters board by invoking a previously defined view filter called “Open Shelters”.

```
//The board that we are getting data for.
string boardName = "Shelters";

//The display view that we are using.
string displayViewName = "List";

//Set up the view filters array.
string[] viewFilters = new string[] { "Open Shelters"};

//Get the xml results
string results = api.GetFilteredData(credentials, boardName, displayViewName,
                                     viewFilters, null);
```

GetFilteredData Using an XML User Filter

In this example, we will retrieve the data for a specific shelter based on the shelter’s name. The XML filter to apply looks like this:

```
<data>
        <name>City High School</name>
</data>
```

```
//The board that we are getting data for.
string boardName = "Shelters";

//The display view that we are using.
string displayViewName = "List";

//Build the xml filter string
string xmlFilter = "<data><name>City High School</name></data>";


//Get the xml results
string results = api.GetFilteredData(credentials, boardName, displayViewName,
                                     null, xmlFilter);
```

Retrieving Board Data By Data Id

The GetDataByDataId is used to retrieve a specific board entry when the data id for that entry is already known. For example, a system interfacing with WebEOC could add a new shelter through the AddData method. The AddData method would return the data id for that newly created shelter record. The third party system could use that data id with GetDataByDataId in the future to retrieve the current shelter information without having to create a filter for that shelter. GetDataByDataId accepts four parameters:

WebEOCCredentials – user credentials
BoardName – the board to retrieve data from
DisplayViewName – the specific display view to retrieve data with.
DataId – the dataid for the entry to retrieve

```
//The board that we are getting data for.
string boardName = "Shelters";

//The display view that we are using.
string displayViewName = "List";

//The data id
int dataId = 1;

//Get the xml results
string results = api.GetDataByDataId(credentials, boardName, displayViewName,
                                     dataId);
```

Getting the WebEOC Board List

The GetBoardNames method returns a list of all that boards that exist within WebEOC. Like GetIncidents this method has only a single parameter, a populated WebEOCCredentials object. The user specified by the WebEOCCredentials object must be a WebEOC administrator or be a member of an admin profile that has permissions to the Boards admin manager.

```
//Create the api service object.
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Retrieve the array of board names.
string[] boards = api.GetBoardNames(credentials);
```

Getting the View List for a Board

The API provides two methods for retrieving the list of views for a particular board. GetInputViews which retrieves all of the input views for a board and GetDisplayViews which retrieves all of the display views. Both methods accept the same parameters:

WebEOCCredentials – user credentials (user must be a Board administrator)
BoardName – the name of the board to retrieve the view list for.

```
//Create the api service object.
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

string boardName = "Shelters";

//Retrieve the array of input view names.
string[] inputViews = api.GetInputViews(credentials, boardName);

//Retrieve the array of display view names.
string[] displayViews = api.GetDisplayViews(credentials, boardName);
```

Getting the List of View Fields

The GetViewFields method returns a list of all the fields on a particular view. The view can be a display or input view. GetViewFields accepts the following parameters:

WebEOCCredentials – user credentials (user must be a Board administrator)
BoardName – the name of the board to retrieve fields for.
ViewName – the specific view to retrieve fields for.

```
//Create the api service object.
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

string boardName = "Shelters";
string viewName = "Input";

//Retrieve the array of field names.
string[] fields = api.GetViewFields(credentials, boardName, viewName);
```

List methods

Getting all lists in WebEOC

The GetLists method returns all of the lists that exist within WebEOC. Each DataRow within the returned DataTable contains the list identifier and the list name.

WebEOCCredentials – user credentials (user must be a List administrator)

```
//Create the api service object.
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object.
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Retrieve the DataTable of all lists on the WebEOC system.
DataTable lists = api.GetLists(credentials);

//Each DataRow has the ID of the list and the list name
System.Console.WriteLine("Lists in the system: ");
foreach (DataRow row in lists.Rows)
{
    Console.WriteLine("list identifier: " + row[0].ToString() +
                      ", list name: " + row[1].ToString());
}
```

Getting all the list items in a list, using the list identifier

The GetListItems method returns the list items in a WebEOC list, based on the list identifier. This method has two parameters:

WebEOCCredentials – user credentials (user must be a List administrator)
listId – the ID of the list to retrieve. List Ids can be obtained with the GetLists method.

Each DataRow in the DataTable of list items contains the list item identifier, the list identifier, and the name of the list item. The below example will get all of the list items for the list with listId 1.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

int listId = 1;

//Retrieve the list items by list identifier
DataTable listItems = api.GetListItems(credentials, listId);
foreach (DataRow row in listItems.Rows)
{
    Console.WriteLine("list item identifier: " + row[0].ToString() +
                      ", list identifier: " + row[1].ToString() +
                      ", list item name: " + row[2].ToString());
}
```

Getting the list items in a list by list name

The GetListItemsByListName method returns the list items in a WebEOC list based on the list name. This method has two parameters:

WebEOCCredentials – user credentials (user must be a Lists and a Boards administrator)
listName – the name of the list.

Each DataRow in the DataTable of list items contains the list item identifier, the list identifier, and the list item name. The below example will get list item information based on the list name.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username =  "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

String listName = "Shelter Status";

//Retrieve the list items by list name
DataTable listItems = api.GetListItemsByListName(credentials, listName);
foreach (DataRow row in listItems.Rows)
{
    Console.WriteLine("list item identifier: " + row[0].ToString() +
                      ", list identifier: " + row[1].ToString() +
                      ", list item name: " + row[2].ToString());
}
```

Making or renaming a list

The SaveList method creates a new WebEOC list with the provided list name or renames the list with the specified listId to the new list name. This method has three parameters:

WebEOCCredentials – user credentials (user must be a List administrator)
listId – 0 if this will be a new list, or the listId of a pre-existing list if this is a list rename.
listName – the name of the list to create, or the new name to rename the list identified by the listId

The SaveList method returns the listId of the list that was created or renamed.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

String listName = "List1";

//Create the list
int listid = api.SaveList(credentials, 0, listName);
```

Making or renaming a list item

The SaveListItem command creates a new list item with the specified name or renames a list item to the specified name. This command also sets or resets the color for the list item. This method has five parameters:

WebEOCCredentials – user credentials (user must be a List administrator).
listId – the listId of a previously created list to which this list item will be added.
listItemId – 0 if this is a new list item, or the listItemId of the pre-existing list item to be renamed.
listItemName – the name of the new list item or what to rename the pre-existing list item.
listItemColor – the 6-character hexadecimal value to set the list item color to.

Note: ListItemIds are not allocated per list; they are allocated over all WebEOC lists. This means that if you make three list items in one list, followed by three in another list, each of the listItemIds will be distinct from all other listItemIds in the WebEOC install. In order to determine the listItemIds for a given list, use the GetListItemsByListName method.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Specify the listId of the list that will have this listItemId added to it
int listId = 1;

//Create a new list item
int listItemId = api.SaveListItem(credentials, listId, 0, "item1", "FFF000");
```

Deleting a list

The DeleteList method deletes a list from WebEOC. The method has two parameters:

WebEOCCredentials – user credentials (must be a List administrator with Delete permissions)
listId – the list identifier of the list to delete

Note: Deleting a list using the API functions the same as deleting the list through the WebEOC user interface: in both cases, deleting the list deletes all of the list items in the list.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//The list identifier of the list to be deleted
int listId = 1;

//Delete the list
api.DeleteList(credentials, listId);
```

Deleting a list item

The DeleteListItem method deletes a list item from a list. The method has two parameters:

WebEOCCredentials – user credentials (must be a List administrator)
listItemId – the list item identifier of the list item to delete.

In the example below, the first list item on this WebEOC system is deleted.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//The specific list item to be deleted
int listItemId = 1;

//Delete the list item
api.DeleteListItem(credentials, listItemId);
```

WebEOCUser Object

The WebEOCUser object encapsulates information describing a single WebEOC user. This object is used when retrieving user information from the API, or when adding a user to WebEOC via the API. The WebEOCUser object contains the following fields:

Username (String) – The user name for the WebEOC user.
IsAdministrator (Boolean – false by default) – A flag indicating if the WebEOC user is an administrator or not.
IsMultipleUserLogin (Boolean – false by default) – A flag indicating if the WebEOC user is set for multiple user login or not.
IsAccountDisabled (Boolean – false by default) – A flag indicating if the WebEOC user account is currently disabled.
ChangePasswordAtNextLogin (Boolean – false by default) - A flag indicating if the WebEOC user will have to change their password on the next login.
IsDualCommitUser (Boolean – false by default) – A flag indicating if the WebEOC user is a dual commit user.
PrimaryEmail (String) – The primary email address of the WebEOC user.
Color (String) – A 6 character hexadecimal value indicating the default color for the WebEOC user.

User and Position methods

Adding a user

The AddUser method adds a user to WebEOC. The method has three parameters:

WebEOCCredentials – user credentials (must be a User administrator with at least Edit permissions)
WebEOCUser – must have at least a username specified in this object
Password – the password that will be assigned to this user

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//create the user object and assign some properties to it
WebEOCUser user = new WebEOCUser();
user.Username = "user1";
user.IsAdministrator = false;
user.IsMultipleUserLogin = true;
user.PrimaryEmail = "user@useremail.com";

//add the user to WebEOC with the specified password
api.AddUser(credentials, user, "userpassword");
```

Deleting a user

The DeleteUser method deletes a user. This method has two parameters.

WebEOCCredentials – user credentials (must be a User administrator with Delete permissions)
UserName – the username to delete

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Delete user2
api.DeleteUser(credentials, "user2");
```

Deleting a user if the user is not assigned to any positions

The DeleteUserIfNoPositions method deletes a user if it is not assigned to any positions. This method has two parameters.

WebEOCCredentials – user credentials (must be a User administrator with 'Delete' permissions)
UserName – the username to delete

If there are no positions assigned to the user, the method will delete the user and return the bool value "true". If there are position(s) assigned to the user, the user will not be deleted and the bool value "false" will be returned.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Delete user2 if it is not assigned to any positions
api.DeleteUserIfNoPositions(credentials, "user2");
```

Adding a user to a position

The AddUserToPosition method assigns a user to a position. The position must exist in WebEOC prior to this call. This method has three parameters:

WebEOCCredentials – user credentials (must be a User Administrator with at least Edit permissions)
UserName – the user to assign to the position.
Position – position the user will be assigned to

The example below adds a new user to WebEOC, followed by assigning that user to a pre-existing position.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//create the user object and assign some properties to it
//(if the user already exists, this step can be skipped)
WebEOCUser user = new WebEOCUser();
user.Username = "user2";
user.IsAdministrator = false;
user.IsMultipleUserLogin = true;
user.PrimaryEmail = "user2@useremail.com";

//add the user to WebEOC with the specified password
api.AddUser(credentials, user, "user2password");

//add the user to the specified position
api.AddUserToPosition(credentials, user.Username, "EOC Director");
```

Removing a user from a position

The RemoveUserFromPosition method removes a user from a position. This method has three parameters:

WebEOCCredentials – user credentials (must be a User administrator with at least Edit permissions)
UserName – the user to un-assign from the position
Position – the position name that the user is to be unassigned from

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Remove user2 from the "EOC Director" position
api.RemoveUserFromPosition(credentials, "user2", "EOC Director");
```

Getting all positions in WebEOC

The GetPositions method gets all of the positions in WebEOC. This method has one parameter:

WebEOCCredentials – position credentials (must be a Positions administrator)

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Get all of the positions in WebEOC
string[] allPositions = api.GetPositions(credentials);
foreach (string position in allPositions)
{
    Console.WriteLine(position);
}
```

Getting all the positions assigned to a user

The GetPositionsForUser method gets all of the positions assigned to a user. Note that administrators have access to all positions. This method has two parameters:

WebEOCCredentials – user credentials (must be a User administrator)
UserName – the name of the user with the assigned positions

Note: the example below uses a non-admin user to demonstrate this API call, since administrators have access to all positions without being explicitly assigned to them.

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Get all the positions assigned to some non-admin user
string[] positions = api.GetPositionsForUser(credentials, "non-admin user");
foreach (string positionName in positions)
{
    Console.WriteLine(positionName);
}
```

Getting all the users assigned to a position

The GetUsersByPosition method gets all of the users assigned to a given position. Note that administrators are by default considered members of all positions. This method has two parameters:

WebEOCCredentials – user credentials (must be a User administrator)
PositionName – the name of the position with assigned users

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Get all of the users assigned to the "EOC Director" position
WebEOCUser[] webeocUsers = api.GetUsersByPosition(credentials, "EOC Director");
foreach (WebEOCUser user in webeocUsers)
{
    Console.WriteLine(user.Username);
}
```

Miscellaneous methods

Checking connectivity and credentials in WebEOC

The Ping method provides a convenient way to determine if WebEOC is accessible and if the credentials being used are correct. The method takes a single parameter, and returns back the string “Pong” on success.

WebEOCCredentials – any credentials of an admin or non-admin user

```
//Create the api service object
APIExample.WebEOCAPI.API api = new API();

//Create the credentials object
WebEOCCredentials credentials = new WebEOCCredentials();
credentials.Username = "WebEOC Administrator";
credentials.Password = "Password";
credentials.Position = "EOC Director";

//Call the Ping method; the expected result is "Pong"
string result = api.Ping(credentials);
System.Console.WriteLine(result);
```