WHATCOUNTS

WhatCounts, Inc.

WebServices API User Guide
Version 1.4

September 6, 2013

WhatCounts, Inc.
101 Yessler Way, Suite 500
Seattle, WA 98104
206.709.8250
206.709.9210 fax

Table of Contents

1. Overview and Architecture

Capabilities

The WhatCounts API allows programmers to control a number of aspects of the permission-based e-mail management. Features available through this API include:

- List creation and editing
- Template creation and editing
- List and template enumeration
- Batch subscribe and unsubscribe
- Subscriber find, update, and delete
- Simple data collection
- Import and Export data
- One-off message deployment
- Campaign deployment

Interface

The API is available as a web service via SOAP or XML-RPC messaging. A downloadable Java client library contains client stub files that make using the API very easy for Java users. For .NET and other clients, the WSDL file that describes the API SOAP interfaces in a standard XML format can be read from:

```
http://[siteURL]/wsdl/WhatCountsAPI.wsdl
```

Where `[siteURL]` is the domain you use to access your system.

Security

The WebServices API requires a configuration license key file. It is also session based. Finer grained user authentication can be done using permission-based constraints. If the messaging itself needs to be encrypted, the SOAP and XML-RPC messaging can be done over SSL.

Version

This is version 1.4 of the API Guide. It is based upon the 1.0 version of the WhatCounts API for Java. Some of the method signatures may have changed, such as return values and error handling. Please use the JavaDocs for the most current method calls and return values.

JavaDocs

The most recent JavaDoc for your specific system is available under:

```
http://[siteURL]/api/docs/javadoc/
```

Where `[siteURL]` is the domain you use to access your system.

Several constant values are used in code examples throughout the help files, usually following the form `APIConstants.CONSTANT_NAME`. You can find those values defined, as well as the values for all integer return codes used by the API, in the JavaDocs.

Between system releases, some method signatures may change, such as return values and error handling. Please use the JavaDocs for the most current method calls and return values.

2. Creating a License/Key File

The requirements for preparing a license/key file for use with the API are simple. You first need to have a valid WhatCounts account and be registered as a site administrator for your company.

To take advantage of added security, you will also need to know a little about your company's network.

Accessing the API

Every API request must include an authentication code that identifies and authenticates the user against the proper realm within the platform. This is identified in the WebServices as the license or key. The license file is a plain-text ASCII string that appears to XML with three elements.

- **Version**: version identifier, used to support different types of encryption and updates to the encoding method.
- **Encryption key**: The key is a unique authentication token for the realm.
- **Encrypted data**: The data block contains other information required by the API, including host information so the API will know which system to communicate with.

Changing the license file invalidates any files or API calls using the old key. If the license already exists and you regenerate it, make sure to notify everyone using the WebServices API.

To generate the license:

1. Login to the platform and then go to **CUSTOMER CENTER > API MANAGEMENT > SETUP API**. You must have the API feature enabled in your realm in order to use it. If you do not see an option to Setup your API, please contact support.
2. Click **Generate License File**. The page will refresh and display a message indicating the license file will be emailed to the email account with which you logged in.
3. When you receive the email, copy the specified portion of text and paste it into a new file.
4. Save the file somewhere your application (and the API) will be able to read it.

Restricting Access

As added security, you can limit access to the API to certain systems in your network. To do this:

1. Go to **CUSTOMER CENTER > API MANAGEMENT > SETUP API**.

2. Enter the IP address in the **Restrict IP Address** field. Separate multiple addresses with commas, spaces or semicolons. To support a range of addresses, enter the IP range. For example, to restrict access through the API to one system at 216.39.173.94, enter that address directly. To allow access from any system on the .173 network, enter only 216.39.173.

3. Click **UPDATE** to save the setting.

Sample License/Key File

Your file may appear on one, long line. The API removes all new lines and carriage returns when it reads this file, so any particular formatting style is irrelevant.

```
<version>1</version>
<key>CD4671A6A61A7DEF3939FE9BADFC6BF</key>
<data>F43D1956D399EB74EEF61074F46696710146
41BAB6B3F1EE97ADC59956656C74EBBB661FADA6BF
CB5730E33C74A611DFD74E6BC4FBCDC5B6E6905C99
94C4FCED54C74A11BEFE396BDC61BDB6407F0A5E</data>
```

3. Installing the API

In the most general case there is nothing to install since the API uses standardized SOAP and XML-RPC messaging protocols that can be implemented on the client side in most programming languages and platforms. There are many free and commercial libraries available that make this easier.

WhatCounts provides some files and utilities that make it very simple to use with .NET on a windows platform and Java on any platform that supports the Java 1.4.2 (or higher) runtime (i.e. Mac OS-X, Linux, Windows, Solaris, etc.).

.NET

If you use Visual Studio to develop for the .NET platform, you can easily generate proxy client code for the WhatCounts Web Services API by creating a Web Reference in your Visual Studio project. The URL for the Web Reference is:

```
http://api.whatcounts.com/wsdl/WhatCountsAPI.wsdl
```

You can also use the wsdl.exe command line tool that comes with .NET Visual Studio:

```
C:> wsdl http://api.whatcounts.com/wsdl/WhatCountsAPI.wsdl /language:cs
```

This command will create a source file `WhatCountsAPIService.cs` in the current directory that can then be compiled into a DLL:

```
C:> csc /t:library /r:System.Web.Services.dll WhatCountsAPIService.cs
```

Some additional documentation concerning this tool can be found at:

```
http://samples.gotdotnet.com/quickstart/aspplus/doc/webservicesintro.aspx
```

Java

For Java programmers there is a library called `whatcountsapi.jar` that contains stub interfaces for the entire API. This is implemented using Apache Axis, an open source SOAP library. This makes it very easy to use the WhatCountsAPI in Java language projects. There is sample code and details on using this library in the appendix.

You can download the `whatcountsapi.jar` for Java at:

```
https://[siteURL]/api/downloads/whatcountsapi.jar
```

Where `[siteURL]` is the domain you use to access your system.

Setting the Endpoint for Broadcasters

The WSDL file default URL points to `http:// api.whatcounts.com`, the main WhatCounts SAAS platform. If your account is not hosted on the main WhatCounts SAAS platform, you must programmatically change the endpoint URL to point to your broadcaster. The last line should change from:

```
<wsdlsoap:address location="http://api.whatcounts.com/webservices/WhatCountsAPI"/>
```

To:

```
<wsdlsoap:address location="http://[siteurl]/webservices/WhatCountsAPI"/>
```

Where `[siteURL]` is the domain you use to access your system.

In .NET (C#, VB.NET), this change can be made in code, or in web.config. If this change is not made you will experience authentication issues.

Session Management

To login and start a session, read in the License Key and pass it as a String value to the `WhatCountsAPI.beginSession(String)` method. A session token (cookie) is returned. Pass along this cookie to the other API methods for the duration of the session. Once you are done, close the session with `WhatCountsAPI.endSession (cookie)` to remove the session data on the server side.

To see an example of session management and using the libraries, see the code examples below.

Making Sure It Works

You can compile and run the following simple example programs to make sure your license file is valid and that SOAP endpoint connectivity is functioning.

C# Example

You will need to create a Web Reference as described above or reference the WhatCountsAPIService.dll in your project to compile and run this example. Also, copy your license file to the same directory that you will run the example from.

To compile this program you can use the .NET command line compiler:

```
csc /r:System.Web.Services.dll /r:WhatCountsAPIService.dll Example1.cs
```

Running the program should produce results similar to:

```
WhatCounts API version = 1.1
Session started, cookie = E4F3F9A0D0247600004D9BD052
Session ended.
```

The session cookie will be different each time you run the program.

```
using System;
using System.IO;

namespace WhatCountsAPIExample
{
    class Example1
    {
        [STAThread]
        static void Main(string[] args)
        {
            // Read the license file
            StreamReader sr = File.OpenText( "whatcounts_api_license.txt" );
            string license = sr.ReadToEnd();

            // Create a proxy endpoint to the remote SOAP API
            WhatCountsAPIService api = new WhatCountsAPIService();

            // if you are not using api.whatcounts.com, you'll need
            // to set an explicit endpoint URL, e.g.:
            // api.Url = "http://our.broadcaster.com/webservices/
                          WhatCountsAPI";

            // Get the current API version
            string version = api.getVersion();
            Console.WriteLine( "WhatCounts API version = {0}", version );
```

```
                // Start a session and then end the session just to
                // validate the license file.
                string cookie = api.beginSession( license );
                if ( cookie != null )
                {
                    Console.WriteLine( "Session started, cookie = {0}", cookie );

                    //------------------------------
                    // API calls can be done here...
                    //------------------------------

                    api.endSession( cookie );
                    Console.WriteLine( "Session ended." );
                }
                else
                    Console.WriteLine( "Can't open an API session." );
            }
        }
    }
```

Java Example

This simple Java console example requires the open source Apache Axis SOAP client libraries. These can be downloaded from http://ws.apache.org/axis/. Also, the client proxy stubs are in whatcountsapi.jar which can be obtained from WhatCounts.

The following jar files are required and need to be added to the classpath:

```
whatcountsapi.jar
axis.jar
commons-discovery.jar
common-logging.jar
jaxrpc.jar
saaj.jar
xml-api.jar
xercesImpl.jar
```

When executed from a console this example should produce results similar to:

```
WhatCounts API version = 1.1
```

```
Session started, cookie = E4F3F9A0D0247600004D9BD052
Session ended.
```

The session cookie will be different each time you run the program.

```
import java.io.*;
import java.net.*;
import whatcounts.api.client.soap.*;

/**
 * Example showing basic usage of the WhatCountsAPI using SOAP with Java
bindings.
 * It uses the Axis (Apache SOAP) client package.
 */

public class Example1
{
    public static void main( String[] args ) throws Exception
    {
        // Read the license file used to authenticate a session
        File file = new File( "whatcounts_api_license.txt" );
        byte[] buf = new byte[(int)file.length()];
        try {
            FileInputStream fs = new FileInputStream( file );
            int n = fs.read( buf );
        } catch ( IOException e ) {
            // Error - file not found...
            throw e;
        }

        String license = new String( buf );
        // the serviceURL may be different if your company has its
        // own WhatCounts Broadcaster

        String serviceUrl =
"http://api.whatcounts.com/webservices/WhatCountsAPI";

        // Create an API stub that is bound to the SOAP service endpoint.
        WhatCountsAPI api = null;
        try {
            URL endpoint = new java.net.URL( serviceUrl );
            api = (WhatCountsAPI) new
```

```
                    WhatCountsAPIServiceLocator().getWhatCountsAPI(endpoint);
            } catch ( javax.xml.rpc.ServiceException e ) {
                // Error - API not available
                throw e;
            }

            // Get the current API version
            String version = api.getVersion();
            System.out.println( "WhatCounts API version = " + version );

           // Begin a session
           String cookie = api.beginSession( license );
           if ( cookie != null )
            {
                System.out.println( "Session started, cookie = " + cookie );

                // -------------------
                // Make API calls here
                // -------------------

                // End the session
                api.endSession( cookie );
                System.out.println( "Session ended." );
            }
            else
                System.out.println( "Can't open an API session." );
        }
    }
```

4. Subscriber Services

Searching for a Subscriber

Searching for a subscriber is easy with the API. The following code snippet demonstrates how to locate a specific individual, in this case a subscriber with the e-mail address *myemail@mydomain.com.*

```
System.out.println ("Searching for myemail@mydomain.com...");
boolean exact = true;
boolean return_xml = false;
String results = api.userSearch(sess, "myemail@mydomain.com", exact,
return_xml);
System.out.println (results);
```

The userSearch method takes four arguments. The first is the session code obtained through the initialize method. The second is the e-mail address pattern. It doesn't have to be an exact address. For example, to return all addresses for subscribers working at Microsoft you could provide the pattern "microsoft.com". However, this needs to be done in concert with the third argument which says "search for this exact address" if true. So, to find the Microsoft subscribers you'd want to set this value to false. The last argument is a flag to specify whether you want the results returned in XML. If false, as in this example, the results will be returned in csv format. If true, in XML. Here's what the output of this test would look like in both formats:

Comma Separated Values (CSV)

```
myemail@mydomain.com, Jane, Doe, 123 Main Street, , Seattle, WA, 98104, US
```

XML

```
<user>
    <email>myemail@mydomain.com</email>
    <first>Jane</first>
    <last>Doe</last>
    <address_1>123 Main Street</address_1>
    <address_2></address_2>
    <city>Seattle</city>
    <state>WA</state>
    <zip>98104</zip>
    <country>US</country>
    <subscription>
        <list_id>43</list_id>
```

```
            <format>1</format>
        </subscription>
        <subscription>
            <list_id>18</list_id>
            <format>1</format>
        </subscription>
    </user>
```

Notice that with the XML option you're automatically returned subscription information – which indicates which lists the user is subscribed to and in which format they've requested to receive their e-mails (1 = plain text, 2 = HTML, 99 = MIME).

Subscribing and Unsubscribing a User

Subscribing and unsubscribing users is straightforward using two methods in the API. Here's a code snippet demonstrating how to subscribe a user to a particular list:

```
int rc = api.userSubscribe(session_code, "myemail@mydomain.com", 15, 1);
```

This request will sign up the user associated with the e-mail address *myemail@mydomain.com* to list 15 and set their format to plain text. If you wanted to sign them up to receive your content in HTML you'd simply set the last parameter to 2.

If you wanted to unsubscribe the same user from the same list, you'd use a piece of code similar to:

```
int rc = api.userUnSubscribe(session_code, "myemail@mydomain.com", 15);
```

You'll notice that for the userUnSubscribe method there's no need to pass in a format setting.

Note: if you attempt to subscribe a user that didn't previously exist in any of your lists a new record will automatically be created.

Setting User Data

Through this API you can set, get and clear user data, the default fields and any custom fields you have defined. Java integers (Integer), strings (String) and date (Date) objects are supported. Once data are stored with subscriber records you can perform segmentation filtering against your lists. You can also treat your subscriber database like a big contact management system – setting and getting data as you like.

As an example, support you wanted to store a subscriber's age. You could use the following method to do that:

```
int rc = api.userSetDataInt (session_code, "myemail@mydomain.com","age", 35);
```

If you wanted to set a string value, such as their favorite color, you could use the same method but with a different type of fourth argument. For example:

```
int rc = api.userSetDataString (session_code, "myemail@mydomain.com",
      "favorite_color", "green");
```

If you wanted to set a date value, you could do that as well, provided you used a java.util.Calendar object for the fourth argument.

All the setData methods (except the MapEntry version below), return APIResponseCodes.API_RESPONSE_OK (0) if the value can be set and APIResponseCodes.API_RESPONSE_FAILURE (1) if it can't.

Setting user data needn't be done one at a time. You can fill a MapEntry array with keys (data names) and associated values and use one function to set all those values at once. For example:

```
MapEntry[] data = new MapEntry[3];
data[0] = new MapEntry();
data[0].setKey("age");
data[0].setValue(new Integer(35));
data[1] = new MapEntry();
data[1].setKey("favorite_color");
data[1].setValue("green");
data[2] = new MapEntry();
data[2].setKey("fave_holiday");
Calendar cal = new GregorianCalendar( 2007, Calendar.MARCH, 17 );
data[2].setValue ( cal );


int rc = api.userSetData (session_code, "myemail@mydomain.com", data);
```

This method returns the number of successful elements set from the collection.

Please note that successive set operations against the same element name for the same subscriber overwrite one another. In the following example the final age associated with the subscriber will be 55.

```
int rc = api.userSetDataInt (session_code, "myemail@mydomain.com", "age", 35);
int rc = api.userSetDataInt (session_code, "myemail@mydomain.com", "age", 55);
```

Also note that spaces should not be used for data names. We recommend you use the underbar ('_') character instead. So "favorite color" would be come "favorite_color."

Clearing User Data

Data can be cleared using two methods. One clears an element at a time, the other all elements for a specified subscriber.

To clear a single element use the *userClearData* method. Example:

```
int rc = userClearData (session_code, "mymailbox@mydomain.net", "age");
```

To clear all element (user data) for a subscriber the method *userClearAllData* can be used. Example:

```
int rc = userClearAllData (session_code, "mymailbox@mydomain.net");
```

Getting User Data

Retrieving data associated with subscriber records is performed using the methods *userGetData* and *userGetAllData*. To retrieve a single data element specify a subscriber's email address, the name of the data element and the format you'd like to receive the data in. For example, to get the user's age (provided it was previously set) in XML you'd call:

```
boolean return_xml = true;
String xml = userGetData (session_code, " mymailbox@mydomain.net", "age",
return_xml);
```

The results would look something like:

```
<age>55</age>
```

If your last argument was false, indicating you didn't want the results to be returned in XML, you'd simple get the string "55." XML formatting is probably only useful when you're asking for all the data associated with a subscriber.

For example:

```
boolean return_xml = true;
String xml = userGetAllData (session_code, " mymailbox@mydomain.net",
return_xml);
```

This method might return output similar to:

```
<age>55</age>
<favorite_color>green</favorite_color>
<DOB>June 15, 1995</DOB>
```

If you don't specify XML for this method the data will be returned in CSV (comma separated values) format with a header row. Example:

```
"age", "favorite_color", "DOB"
"35", "green", "June 15, 1995"
```

5. Template Services

Enumerating Template Names

Getting a list of current template names is performed using the method *enumTemplates*. You can receive template information in numerical or text format – and the text can be comma separated or XML.

To retrieve template numbers, separated by commas, call the *enumTemplates* method with the option APIConstants.ENUM_TYPE_NUMBERS.

```
String results = api.enumTemplates (session_code,
    APIConstants.ENUM_TYPE_NUMBERS);
```

To retrieve a list of template names, separated by commas, call the same method but use APIConstants.ENUM_TYPE_NAMES.

```
String results = api.enumTemplates (session_code,
    APIConstants.ENUM_TYPE_NAMES);
```

To retrieve the same list, but in XML, use the option APIConstants.ENUM_TYPE_XML.

```
String results = api.enumTemplates (session_code,
    APIConstants.ENUM_TYPE_XML);
```

Creating a new Template

Creating a new template is performed through the *templateCreate* method. It takes two arguments: a session code and the name you would like to assign to the template. The name can be anything you like, provided it doesn't already exist. Both the session code and template name are String objects. Here's a sample:

```
int rc = api.templateCreate (session_code, "Monthly Sale");
```

This request will attempt to create a new template named "Monthly Sale." If successful, the return string will contain the word "SUCCESS." Any other response will represent a failure message. One possible failure message might be "FAILURE: this template name is already in use."

Editing a Template

Controlling what goes "into" a template is performed through the *templateEdit* method. It takes four arguments: session code, template name, content type and content.

The template name is the same name you might have provided through the *templateCreate* method, or manually through the web site. It's not case sensitive.

The content type is an integer used to tell the method which element of the template you're editing. Remember, WhatCounts templates have multiple components such as: plain text, HTML, AOL and WAP. Future components may be created. For that reason, the *templateEdit* method requires you to identify the part of the template you're changing. Right now there are the following possible choices:

```
TEMPLATE_PART_SUBJECT
TEMPLATE_PART_DESCRIPTION
TEMPLATE_PART_NOTES
TEMPLATE_PART_PLAINTEXT
TEMPLATE_PART_HTML
TEMPLATE_PART_AOL
TEMPLATE_PART_WAP
```

Here's an example of editing the HTML component of a template named "Monthly Sale":

```
int rc = api.templateEdit (session_code, "Monthly Sale",
      APIConstants.TEMPLATE_PART_HTML, "<H1>Hello World!</H1>");
```

In this example we're setting the HTML component of the template *Monthly Sale* so that it contains a tiny bit of example HTML.

If the editing operation is successful the method will return API_RESPONSE_OK. Otherwise, an error code will be returned. One such error might be API_RESPONSE_NOT_FOUND if the template *Monthly Sale* does not exist.

If you wanted to set the subject line for the template (which is the same subject that your e-mail recipients will see in the e-mail inbox) you might do the following:

```
int rc = api.templateEdit (session_code, "Monthly Sale",
      APIConstants.TEMPLATE_PART_SUBJECT,
      "Hey %%FIRST%%, check out these great new sales!");
```

Getting Template Content

If you'd like to read the contents of a template you can use the method *templateGetData*. It works similarly to the *templateEdit* method in that you specify which element within the template you'd like to read. This method takes three arguments: the session code, name of the template and the element within the template you want to read.

For example, to read the HTML component, you could call the method like:

```
String result = api.templateGetData (session_code, "Monthly Sale",
    APIConstants.TEMPLATE_PART_HTML);
```

The output from this method is the piece of the template you've asked for. If it's empty or there's an error, this method will return a blank string.

Deleting a Template

Deleting templates is handled by the *templateDelete* method. For example, to delete a template named "Monthly Sale" you'd might using the following snippet of code:

```
int rc = api.templateDelete (session_code, -1, "Monthly Sale");
```

To delete a template by ID instead of name, the code would look like this:

```
int rc = api.templateDelete (session_code, 81, null);
```

The output from this method is API_RESPONSE_OK unless there's an error, in which case it may return API_RESPONSE_NOT_FOUND or API_RESPONSE_FAILURE.

Copying a Template

Templates can also be copied. To make a copy of an existing template, use the *templateCopy* method. It takes there arguments: a session code, the name of the original (source) template and the name of the template you'd like to make that will be a copy of the source one.

Here's an example copying a template named "Monthly Sale" to one named "Annual Sale":

```
int rc = api.templateCopy (session_code, "Monthly Sale", "Annual Sale");
```

The output from this method is API_RESPONSE_OK unless there's an error, in which case it may return API_RESPONSE_NOT_FOUND or API_RESPONSE_FAILURE.

6. List Services

Enumerating List Names

Getting a list of current lists is performed using the method *enumLists*. You can receive list information in numerical or text format – and the text can be comma separated or XML.

To retrieve list numbers, separated by commas, call the *enumLists* method with the option APIConstants.ENUM_TYPE_NUMBERS.

```
String results = api.enumLists (session_code, APIConstants.ENUM_TYPE_NUMBERS);
```

To retrieve a list of names, separated by commas, call the same method but use the option APIConstants.ENUM_TYPE_NAMES.

```
String results = api.enumLists (session_code, APIConstants.ENUM_TYPE_NAMES);
```

To retrieve the same list, but in XML, use the option APIConstants.ENUM_TYPE_XML.

```
String results = api.enumLists (session_code, APIConstants.ENUM_TYPE_XML);
```

Creating Lists

To create a new list you'll need to fill out fields within an object named *WhatCountsAPIList*. Here's a list of fields:

Field Name	Type	Description
list_id	int	This field is set by the getListData method and is not meant to be set by the API client.
name	String	Name of the list
from_address	String	From-address (can contain template tags)
reply_to_address	String	Reply-to-address (can contain template tags)
errors_to_address	String	Errors-to-address
description	String	Description for the list
template_id	int	Id of the template used by this list
multipart	boolean	Deprecated, do not use
use_aol	boolean	Deprecated, do not use
wrap_plain_text	boolean	Deprecated, do not use
wrap_html	boolean	Deprecated, do not use

track_html_reads	boolean	If you want to perform HTML read tracking for campaigns launched with this list, set this field to true.
track_click_throughs	boolean	If you want to enable click-through tracking for all the URLs used by the templates with this list, set this field to true.
opt_in	boolean	Would you like to enable double opt-in support for this list? If so, set this field to true. Subscribers will automatically receive a confirmation message when they subscribe. Unless they positively acknowledge the special confirmation e-mail, their original request will be ignored.
opt_out	boolean	Would you like to enable double opt-out support for this list? If so, set this field to true. Subscribers will automatically receive a confirmation message when they unsubscribe. Unless they positively acknowledge the special confirmation e-mail, their original request will be ignored and they'll remain subscribed to the list.
send_signup_ack	boolean	Would you like subscribers to receive a courtesy message confirming their signup request? Set this field to true to enable this feature.
send_cancel_ack	boolean	Would you like subscribers to receive a courtesy message confirming their cancellation request? Set this field to true to enable this feature.
signup_template_id	int	ID of the template to be used to generate the signup acknowledge message. If set to zero (default), a system-generated message will be used.
cancel_template_id	int	ID of the template to be used to generate the cancellation acknowledge message. If set to zero (default), a system-generated message will be used.
external_signup_link	String	URL for an external page that the subscriber will be directed to after their signup form has been processed. This should be a fully qualified URL.
external_cancel_link	String	URL for an external page that the subscriber will be directed to after their cancellation form or link has been processed. This should be a fully qualified URL.
tracking_base_url	String	Base URL for tracking links. This is valid for customer that have created a DNS host alias record that points to one of the WhatCounts servers and allows the tracking URLs to carry customer-specified branding (domain name).

After you've created a *WhatCountsAPIList* object you can call the method *listCreate* to create the list. Example:

```
WhatCountsAPIList details = new WhatCountsAPIList ();
details.setName("My Test List");
details.setOptIn(true);
details.setTemplateId(83);
int rc = api.listCreate (session_code, details);
```

If your request was handled successfully the returned integer will match the API_RESPONSE_OK. Otherwise, it will match a FAILURE code.

Obtaining List Details

To obtain details about a particular list you can use one of the *listGetData* methods.

Results are returned in a *WhatCountsAPIList* object (described earlier) if successful and null if not successful. As an example, to retrieve the settings for the list named "marketing list" with ID 187, you could perform the following:

```
String name = "marketing list";
WhatCountsAPIList details = api.listGetDataByName (session_code, name);
```

To retrieve the data for the list with ID number 187, you could use the following:

```
int listID = 187;
WhatCountsAPIList details = api.listGetDataById (session_code, listID);
```

After this call is made the object *details* should be filled with your list settings.

Updating Lists

If you would like to update a list you should first make sure you've obtained current information about the list through a call to *listGetData*. After you've done that, you can modify the fields within the returned *WhatCountsAPIList* object and resubmit it to a method named *listUpdate.*

The *listUpdate* method takes three arguments: session code, list number and your *WhatCountsAPIList* object. Example:

```
int rc = api.listUpdate (session_code, 5, details);
```

You can change any field except the list_id field – which will be ignored in any case. If your request was handled successfully the returned integer will match the API_RESPONSE_OK. Otherwise, it will match a FAILURE code.

Deleting Lists

To delete a list, call the method *listDelete.* It takes two arguments, your session code and a list ID. Example:

```
int rc = api.listDelete (session_code, 5);
```

If your request was handled successfully the returned integer will match the API_RESPONSE_OK. Otherwise, it will match a FAILURE code.

Copying Lists

To copy a list, call the method *listCopy*. It takes three arguments: your session code, the list ID of the source list, and a name for the new list you'd like to create.

```
int rc = api.listCopy (session_code, 100, "new list");
```

If your request was handled successfully the returned integer will match the API_RESPONSE_OK. Otherwise, it will match a FAILURE code.

Note: copying a list copies only the settings for that list, not the subscription records associated with it.

Testing a List

To test a list, call the method *listTest*. It takes many arguments. Specifically:

- Session code
- Email address of the sender
- List ID
- Template ID
- List of recipients (comma-separated email addresses in a String)
- Boolean flag to indicate whether a multipart MIME version of the message should be generated
- Boolean flag to indicate whether a plain text version of the message should be generated
- Boolean flag to indicate whether an HTML version of the message should be generated
- Boolean flag to indicate whether to prepend "Testing" to the Subject line found in the template

For example:

```
int list_id = 10;
int template_id = 250;
int rc = api.listTest (session_code, "tester@whatcounts.com",
       list_id, template_id, "recipient1@test.com, recipient2@test.com",
       false, false, true, true);
```

If your request was handled successfully the returned integer will match the API_RESPONSE_OK. Otherwise, it will match a FAILURE code. After calling this method check to see that the test messages were properly generated and delivered.

7. Campaign Deployment

To run a list as a campaign, use one of the listRun methods. The original listRun method allows you to set basic rules to send the campaign. The newer listRun method uses an XML parameter to set additional campaign rules, such as throttling and campaign_alias.

Sending a Campaign using listRun

The basic listRun method takes several arguments:

- Session code/cookie
- Email address of the sender
- List of one or more email addresses (in a String) to notify on completion
- List ID
- Template ID
- Segmentation ID (0 = all subscribers)
- Format of the message
  - 0 for Plain-text
  - 2 for HTML
  - 99 for Multipart MIME
  - -1 for Subscriber specified (based upon their subscription record)
- Boolean value to indicate whether a test should be performed; if set to "true" no actual email will be delivered

The listRun method is defined as follows:

```
public int listRun(String cookie, String creator_email, String notify_email,
int list_id, int template_id, int segmentation_id, int forced_format,
boolean test_mode)
```

For example:

```
int list_id = 10;

int template_id = 250;

int segmentation_id = 0; // all subscribers

int format = 2; // HTML

int rc = api.listRun (session_code, "sender@mydomain.com",
      "notify1@mydomain.com, notify2@mydomain.com",
      list_id, template_id, segmentation_id, format, false);
```

After calling this method check to see that the test messages were properly generated and delivered.

On success, the method returns the code APIResponseCodes.API_RESPONSE_OK. This method will return very quickly, often well before the messages begin getting published and sent through the SMTP servers.

Sending a Campaign using listRun with XML

To send a campaign to a list of subscribers, use *listRun*. This method requires two parameters including a set of arguments in the form of an XML-formatted string, including:

- Session code/cookie
- List ID

While the xml_flags use XML-formatted data, it is not processed as true XML. Therefore, it does not require an XML header, and there is no need to encode special characters or use <![CDATA[ ]]>. Instead, just define strings as variables or regular text between the XML tags. For example:

```
String xmlFlags = "<list_id>mymailbox@mydomain.com</list_id>" +
"<template_id>reply@mydomain.com</template_id>" +
"<format>2</format>";
```

The listRun method is defined as follows:

```
public String listRun(String cookie, String xml_flags)
```

If the call is successful, instead of a simple response code, the method returns an XML response string. The response always contains at least a <result> and a <message>. If successful, it will also contain a <task_id> and <count>. The count is the number of subscribers found BEFORE segmentation or suppression is applied.

For example:

```
<response>
<result>SUCCESS</result>
<message>A task was successfully created to run list 722</message>
<task_id>34887</task_id>
<count>357113</count>
</response>
```

Method parameters are as follows:

Parameter	Type	Required	Description
cookie	String	required	Session code
xml_flags	String	required	See the XML table below

The XML arguments are as follows:

Argument	Type	Required	Definition
list_id	int	required	The id number of the list properties to use. If no other parameters override the list properties, they are used to generate the message.
template_id	int	optional	The id number of the template content to send. Overrides the template identified in the List properties. Template ID is required if no template is defined in the list.
subject	String	optional	Subject for the email message. Overrides the subject in the template, if specified, or if retrieved from the list's template.
format	int	optional	To specify the format of the email messages your subscriber will receive, set the Format to the corresponding value. 1 = Plain Text 2 = HTML 99 = Multipart MIME
campaign_throttle _ rate	int	optional	If throttling is enabled for a realm, this is used to limit the number of messages sent per hour.
creator_email	String	optional	Email address of the person creating the mailing task.
notify_email	String	optional	Email address of the person to send a success or failure email to upon task completion.
seed_list_id	int	optional	ID of the special seed list of subscribers to add to the campaign. This is a specially defined list of 25 or fewer subscribers addresses, depending on the number configured in the system.
segmentation_id	int	optional	ID of the segmentation rule to use to limit the subscribers who will receive the message. If not present, the campaign will deploy to all subscribers on the list.
set_data_macro_i d	int	optional	If specified, runs the specified set data macro during publishing.
suppression_list	String	optional	The name of the suppression list to use during publishing.
target_rss	String	optional	If true, then publishes to RSS subscribers. Valid values include: true, yes, 1 OR false, no, 0
vmta	String	optional	Name of the virtual MTA IP address through which to send the messages. If list is defined, then the VMTA defined in the list properties will automatically be used.
campaign_ alias	String	optional	Friendly name to identify the campaign in reports.

For example:

```
String xmlFlags = "<list_id>1234</list_id>" +
    "<template_id>4321</template_id>" +
    "<format>mime</format>";

String responseCode = api.listRun(cookie, xmlFlags);
```

8. Transactional Message Sending

While several methods support one-off, or transactional, message sending, a single method supports the most recent features and improvements and should be used in place of all previous *sendMessage* methods.

Sending a Transactional (One-off) Message

To send a transactional email message to a single email address, use *sendMessage*. This method requires several parameters and a set of arguments in the form of an XML-formatted string, including:

- Session code/cookie
- Email address of the sender (From Address)
- Email address to send the message (To Address)
- Template ID or Body

While the xml_flags use XML-formatted data, it is not processed as true XML. Therefore, it does not require an XML header, and there is no need to encode special characters or use <![CDATA[ ]]>. Instead, just define strings as variables or regular text between the XML tags. For example:

```
String xmlFlags = "<from>mymailbox@mydomain.com</from>" +
"<reply_to>reply@mydomain.com</reply_to>" +
"<to>" + recipientEmail + "</to>" +
"<format>" + format + "</format>";
```

The sendMessage method is defined as follows:

```
public int sendMessage(String cookie, String xml_flags, String subject, String
html_body, String plain_text_body, MapEntry[] predata)
```

On success, the method returns the code APIResponseCodes.API_RESPONSE_OK.

Method parameters are as follows:

Parameter	Type	Required	Description
cookie	String	required	Session code
xml_flags	String	required	See the XML table below
subject	String	conditional	Subject for the email message. Overrides the subject in the template, if specified, or if retrieved from the list's template. The subject can contain custom data , for example: Welcome, to %%$newsletter%%, %%$first_name%%!
html_body	String	conditional	Body of the message in HTML format. Overrides HTML content if a template is specified. Required if no template is specified or defined in the list and the message format is set to HTML or MIME.
plain_text_ body	String	conditional	Body of the message in Plain Text format. Overrides Plain Text content if a template is specified. Required if no template is specified or defined in the list and the message format is set to PLAIN or MIME.
predata	MapEntry[]	optional	Custom data to use in the subject or body of your email message.

The XML arguments are as follows:

Argument	Type	Required	Definition
template_id	int	optional	The id number of the template content to send. Overrides the template identified in the List properties. Template ID is required if no template is defined in the list or no Body is defined in the parameters.
list_id	int	optional	The id number of the list properties to use. If no other parameters override the list properties, they are used to generate the message. If the list has the sticky campaign flag set, open and click-through tracking will be enabled.
format		optional	To specify the format of the email messages your subscriber will receive, set the Format to the corresponding value. The format of the message can be defined as a number or text: 1 or plain = Plain Text 2 or html = HTML 99 or mime = Multipart MIME Example: <format>99</format> <format>html</format>

from	String	conditional	From email address. Overrides the from address specified in the list. This can be formatted using a decorative portion, such as: "Decorative Portion" <mailbox@domain.com>. Required as an argument if no List is defined, or if the defined List has no From Address.
to	String	required	Recipient email address.
first_name	String	optional	First name of the recipient. Use only when the command is defined for one recipient.
sender	String	optional	Sender address. The Sender appears in Outlook and Hotmail as "Sender on behalf of Friendly From (From Address)." Use in situations where you want to use an agent's from address to indicate to the recipient who the email is from but cannot use the from address for delivery reasons.
cc	String	optional	CC email address, a maximum of 5 are allowed. Use only when the command is defined for one recipient. This feature is enabled in the realm settings. Please contact support for more information about using this feature.
reply_to	String	optional	Reply-to email address. Overrides the reply-to address specified in the list. This can be formatted using a decorative portion, such as: "Decorative Portion" <mailbox@domain.com>.
mail_from	String	optional	Bounce mailbox email address used to forward bounced messages. Overrides the Bounce address in the list. NOTE: While the Bounce address can be set to any valid email address, if not set to the bounce address for your realm, it will not use the system Bounce Handler and no bounce tracking will be recorded.
charset	String	optional	Sets the character set to use for the message content, such as UTF-8.
encoding	String	optional	Sets the message content encoding to one of the following: base64 - for BASE64 quoted - for Quoted-Printable default - for system default
vmta	String	optional	Name of the virtual MTA IP address through which to send the messages. If list is defined, then the VMTA defined in the list properties will automatically be used.

campaign_name	String	optional	To track one-off message events, such as opens and click-throughs, the List properties must be set to Sticky Campaigns. If you then define a campaign name in the Campaign tag, you can view the results as a single campaign. NOTE: any campaign sent with the same Campaign Name will be aggregated into a single campaign using the initial List, even if using different lists.
ignore_optout	String	optional	Indicates that the message must be sent to the subscriber email address even if an optout record exists. Only valid value for this argument is true. USE WITH CAUTION. This is designed for situations where you must be able to get the email through, such as password reset emails or confirmation-type emails. Should not be used for most transactional emails.

Sending Simple Messages

To send a message without using a template, you can define the Plain Text or HTML body in your code. For example:

```
String fromAddress = "mymail@mydomain.com";
String replytoAddress = "errors@mydomain.com";
String recipientEmail = "info@mydomain.com";
String subject = "This is a test message";
String format = "plain";

String xmlFlags = "<from>" + fromAddress +"</from>" +
    "<reply_to>" + replytoAddress + "</reply_to>" +
    "<to>" + recipientEmail + "</to>" +
    "<format>" + format + "</format>";

String plainTextBody = "Hello World!";
String htmlBody = "";

MapEntry[] preData = null;

try {
    int rc = api.sendMessage(cookie, xmlFlags, subject, htmlBody,
        plainTextBody, preData);
    if (rc == 0) {
        // success
    }
    else if (rc == 1) {
        // handle general failure
    }
    else if (rc == 18) {
        // recipient has previously opted out
    }
}
catch (Exception e) {
    // handle exception . . . some error messages will show up here
}
```

If the method is successful, it will return the response API_RESPONSE_OK. Otherwise, it will return a Failure code.

To send a message as an HTML message, use the same method but set the format to HTML, or 2. Then define the HTML body:

```
String htmlBody = "<p>Hello World!"</p>";
```

Similarly, to send a Multipart Mime message, define both the HTML and Plain Text body, and set the format to mime, or 99.

```
String plainTextBody = "Hello World!";
String htmlBody = "<p>Hello World!"</p>";
```

Sending Messages Using a Template

To send a message using content from a template, define the template_id in the XML string. Leave the subject, html_body and plain_text_body parameters blank. The exception is if you want to override the subject defined in the template. Then, send in a subject parameter with the content you want for your subject.

This example specifies both a list_id and template_id. There are some nuances here. If you have a list that has a default template, the default template could be used. Simply don't specify a template_id and the system will use the default template set up with the list. To send an email with a template, you do not have to specify a list_id. If you do not, however, your email would not be tracked.

```
String recipientEmail = "janedoe@mydomain.com";

String xmlFlags = "<list_id>1234</list_id>" +
    "<template_id>4321</template_id>" +
    "<to>" + recipientEmail + "</to>" +
    "<format>mime</format>";

MapEntry[] preData = null;

try {
    int rc = api.sendMessage(cookie, xmlFlags, "", "", "", preData);

    if (rc == 0) {
        // success
    }
    else if (rc == 1) {
        // handle general failure
    }
    else if (rc == 18) {
    // recipient has previously opted out
    }
}
catch (Exception e) {
    // handle exception . . . some error messages will show up here
}
```

In the following example, the template is not specifically defined in the XML string, but instead is pulled from the specified list.

```
String recipientEmail = "janedoe@mydomain.com";

String xmlFlags = "<list_id>1234</list_id>" +
"<to>" + recipientEmail + "</to>" +
"<format>mime</format>";

MapEntry[] preData = null;

try {
    int rc = api.sendMessage(cookie, xmlFlags, "", "", "", preData);

    if (rc == 0) {
        // success
    }
    else if (rc == 1) {
        // handle general failure
    }
    else if (rc == 18) {
        // recipient has previously opted out
    }
}
catch (Exception e) {
     // handle exception . . . some error messages will show up here
}
```

Sending a Message with Custom Data

To send a message with custom data using template variables, use the MapEntry to set an array of custom values. For example:

```
MapEntry[] preData = new MapEntry[3];
preData[0] = new MapEntry();
preData[0].setKey("first");
preData[0].setValue("Jane");
preData[1] = new MapEntry();
preData[1].setKey("city");
preData[1].setValue("Seattle");
preData[2] = new MapEntry();
preData[2].setKey("age");
preData[2].setValue("39");
```

To take advantage of these replacement variables in your template (in any format) simply use the template tags similar `%%$first%%`, `%%$city%%`, and `%%$age%%`. Replacement variables are applied to both the subject and body of your message. Note that these variables do not have to be defined as Custom Fields in order to use them in Templates.

Sending a Message with Transactional Data

Transactional messages can be personalized using a combination of Articles and Logic statements in your template. This robust model allows you to define various data and does not constrain the amount or type of information that can be contained within an article. To customize the content per Subscriber, you must:

1. Create an Article to pull data from data defined in the sendMessage method call.
2. Create a Template that calls the Article through a logical loop.
3. Define the custom data using a MapEntry array.

As an example, the following message displays a simple shipping statement:

```
Dear Jane Doe:

Thank you for your recent purchase.  Please find a summary of your recent
purchase below, along with shipment tracking information.
```

Quantity	SKU	Description	Price
2	Z1 2345 128282	Widget	$24.95
1	Z1 2727 182883	Sprocket	$19.95
1	Z1 2345 189764	Doohickey	$0.88

```
Thank you for shopping with us.
```

In this example, the Subscriber's first and last names are grabbed from the Subscriber record in the database. The shipping information is merged into the Template by the inclusion of several versions of an Article that grab information from the XML Data File.

Article Example

The first step is to define the Article content. Use names of the Item columns you will be defining in the MapEntry definition. In this case, the column names are product, price, and shipping_number:

```
<tr>
    <td>%%$qty%%</td>
    <td>%%$sku%%</td>
    <td>%%$description%%</td>
    <td>%%$price%%</td>
</tr>
```

Since the resulting message will display each shipped item in a separate row, this Article is formatted using HTML, however plain text content is also an option.

Template Example

To merge the defined items into a message, next define the template. The following is an HTML example, but Plain-Text can also be used:

```
<P>Dear %%$first%% %%$last%%: </p>
<P>Thank you for your recent purchase.  Please find a summary of your recent
purchase below, along with shipment tracking information.</p>
<table>
  <tr>
    <td>Quantity</td>
    <td>Sku</td>
    <td>Description</td>
    <td>Price</td>
  </tr>
  %%foreach "order" article "order_rows"%%
</table>
<P>Thank you for shopping with us.</p>
```

If you are familiar with the system Templates and Tags, you will recognize the Subscriber Data tags %%$first%% and %%$last%% which call the Subscriber's name. The third tag in this example is a logic tag foreach:

```
%%foreach "order" article "order_rows"%%
```

The foreach tag is a logical looping mechanism to display rows of information through an article. It checks the identified Item block and prints the corresponding article content for each row of defined data. The loop does the following:

1. Look at the defined items and find the Items block called "order" for this subscriber.
2. If there is a <row> tag, then call the article "order_rows".
3. Print the article "order_rows" for every <column> tag, matching it to a tag defined in the article, such as %%$sku%%, corresponding to the column named product.
4. Continue until there are no new rows found.

Items

Information in an Item block defined by a MapEntry array follows a row/column data model. Rows can contain any number of columns (as defined by your article template) and columns can contain any type of data. Data specified within rows is not stored in the Subscriber record, and is only available through the MapEntry array to create and send one-off email messages.

The MapEntry for the Subscriber includes at least one Item block identified by a name "order", which is used in the Template to find the correct Item block. Each column is also identified with a name so the Article can grab the correct content.

In this XML example the rows of information are the products the Subscriber purchased. Three rows defined the products purchased, and each row includes four columns of data: product name, price, and shipment tracking number.

```
<items name="order">
  <row>
    <column name="qty">2</column>
    <column name="sku">Z1 234512 8282</column>
    <column name="description">Widget</column>
    <column name="price">$24.95</column>
  </row>
  <row>
    <column name="qty">2</column>
    <column name="sku">Z1 2727 182883</column>
    <column name="description">Sprocket</column>
    <column name="price">$19.95</column>
  </row>
  <row>
    <column name="qty">2</column>
    <column name="sku">Z1 2345 189764</column>
    <column name="description">Doohickey</column>
    <column name="price">$0.88</column>
  </row>
</items>
```

Code Example

The following example code shows how to define the MapEntry array, converting the XML to store in the array:

```
List<MapEntry> mapEntry = new List<MapEntry>();
```

```
MapEntry me = new MapEntry();
me.key = "Order";
me.value = "<items name=\"Order\"><row><column name=\"qty\">1</column><column
name=\"sku\">S-1234</column><column name=\"description\">Widget -
Black</column><column name=\"price\">14.99</column></row><row><column
name=\"qty\">4</column><column name=\"sku\">S-2345</column><column
name=\"description\">Gadget</column><column
name=\"price\">38.16</column></row><row><column name=\"qty\">1</column><column
name=\"sku\">S-3456</column><column
name=\"description\">Sprocket</column><column
name=\"price\">4.99</column></row></items>";
mapEntry.add(me);
int listId = 1234;
int templateId = 123;
String toAddress = "mymailbox@mydomain.com";
String xmlFlags = "<list_id>" + listId + "</list_id" > +
    "<template_id>" + templateId + "</template_id" > +
    "<to>" + toAddress + "</to>" +
    "<format>99</format>";
try
{
    int rc = api.sendMessage(cookie, xmlFlags, "", "", "", mapEntry.toArray());

    if (rc == 0) {
        // success
    }
    else if (rc == 1) {
        // handle general failure
    }
    else if (rc == 18) {
        // recipient has previously opted out
    }
}
catch (Exception e) {
    // handle exception . . . some error messages will show up here
}
```

9. Blog API XML-RPC

WhatCounts implements both the Blogger XML-RPC API and the metaWebLog XML-RPC API.

Note: The Blogger API's input parameter appkey is ignored and the blogger.getTemplate and blogger.setTemplate are not implemented because Blogger templates are not compatible with WhatCounts templates.

The following is a list of supported methods, parameters, and return value(s):

wcBlogAPI.getVersion

Get the Blog API version in the major.minor form: "[0-9]+\.[0-9]+".

Parameters: String appkey, String blogid, String username, String password, String content, boolean publish

Return: String version, otherwise fault.

blogger.newPost

Create a new post and optionally publish it.

Parameters: String appkey, String blogid, String username, String password, String content, boolean publish

Return: String postid of new post on success, otherwise fault

blogger.editPost

Update the information in an existing post.

Parameters: String appkey, String postid, String username, String password, String content, boolean publish

Return: on success, boolean true value, otherwise fault

blogger.deletePost

Delete a post.

Parameters: String appkey, String postid, String username, String password, boolean publish

Return: boolean `true` on success, otherwise fault

blogger.getRecentPosts

Get the most recent posts in the system.

Parameters: String appkey, String blogid, String username, String password, int numberOfPosts

Return: array of structs containing ISO.8601 dateCreated, String userid, String postid, String content; or a fault on failure

blogger.getUsersBlogs

Get the weblogs to which an author has posting privileges.

Parameters: String appkey, String username, String password

Return: on success, array of structs containing String url, String blogid, String blogName; on failure, fault

blogger.getUserInfo

Get information about an author in the system.

Parameters: String appkey, String username, String password

Return: on success, struct containing String userid, String firstname, String lastname, String nickname, String email, String url; on failure, fault

metaWeblog.newPost

Create a new post and optionally publish it.

Parameters: String blogid, String username, String password, struct content, boolean publish

Notes: the struct content can contain the following standard keys: title, for the title of the entry; description, for the body of the entry; and dateCreated in ISO.8601 format, to set the created-on date of the entry.

Return: on success, String postid of new post; on failure, fault

metaWeblog.editPost

Update information about an existing post.

Parameters: String postid, String username, String password, struct content, boolean publish

Return: on success, boolean true value; on failure, fault

Notes: the struct content can contain the following standard keys: title, for the title of the entry; description, for the body of the entry; and dateCreated, to set the created-on date of the entry.

metaWeblog.getPost

Get information about a specific post.

Parameters: String postid, String username, String password

Return: on success, struct containing String userid, ISO.8601 dateCreated, String postid, String description, String title, String link, String permaLink; on failure, fault

Notes: link and permaLink are both the URL pointing to the archived post.

metaWeblog.getRecentPosts

Get the most recent posts in the system.

Parameters: String blogid, String username, String password, int numberOfPosts

Return: on success, array of structs containing ISO.8601 dateCreated, String userid, String postid, String description, String title, String link, String permaLink; on failure, fault

Notes: dateCreated is in the timezone of the weblog blogid; link and permaLink are the URL pointing to the archived post

metaWeblog.newMediaObject

Upload a file to your webserver.

Parameters: String blogid, String username, String password, struct file

Return: URL to the uploaded file.

Notes: the struct file should contain two keys: base64 bits (the base64-encoded contents of the file) and String name (the name of the file). The type key (media type of the file) is currently ignored.

Example

The following sample code is an example in Java using the apache XML-RPC client library (many other XML-RPC client implementations work as well). The program will list the first three blog entries of every blog the user owns.

```
import java.util.*;
import java.io.IOException;
import java.text.BreakIterator;


import org.apache.xmlrpc.*;


/**
 * XMLRPCBlogAPIExample
 *
 * Simple Java example showing basic usage of the WhatCounts BlogAPI using XML-
RPC.
 * For every blog the user has permissions to post, prints the first three blog
entries.
 * <br>
 * It uses the Apache XML-RPC client package (http://ws.apache.org/xmlrpc/).
 * <p>
 * Usage: java XMLRPCBlogAPIExample <userid>:<realm> <password> [<endpoint-url>]
 */
public class XMLRPCBlogAPIExample
{
    public static void main( String args[] ) throws IOException, XmlRpcException
    {
        if ( args.length < 2 )
        {
            System.err.println( "Usage: java XMLRPCBlogAPIExample
<userid>:<realm> <password> [<endpoint-url>]" );
            System.exit( -1 );
        }
        String userid = args[0];
        String passwd = args[1];
        String endpointUrl = "http://www.whatcounts.com/webservices/XMLRPC";
```

```
        if ( args.length == 3 )
            endpointUrl = args[2];

        // Create an endpoint to the XML-RPC WhatCountsAPI
        XmlRpcClient xmlrpc = new XmlRpcClient( endpointUrl );

        // Method parameters are put in this vector
        Vector params = new Vector();

        // Call wcBlog.getVersion() to get the current API version (for example)
        String version = (String) xmlrpc.execute(
"WhatCountsBlogAPI.getVersion",
                                                 params );
        System.out.println("WhatCounts Blog API version = " + version );

        // Using a Blogger 1.0 method, get a list of blogs that
        // this user has permission to post to.
        // This returns an array of structs (as hashtables)
        // containing the keys blogid, blogName, and url
        params.clear();
        // parameter <appkey> is ignored, so pass an empty string
        params.addElement( "" );
        params.addElement( userid );    // the user's WhatCounts login id
        params.addElement( passwd );    // the user's password
        Vector blogs = (Vector) xmlrpc.execute( "blogger.getUsersBlogs", params
);
        Enumeration enum = blogs.elements();
        while ( enum.hasMoreElements() )
        {
            Hashtable h = (Hashtable) enum.nextElement();
            String blogid = (String) h.get( "blogid" );
            String blogName = (String) h.get( "blogName" );
            String url = (String) h.get( "url" );
            System.out.println( "\n\nBlog: " + blogName + ", blogid=" + blogid +
                                ", url=" + url );

            // For every blog found get the last three posts
            // Use the MetaWebLog method metaWebLog.getRecentPosts
            // instead of the Blogger
            // version to get more information about each post.
            params.clear();
```

```
            // parameter <appkey> is ignored, so pass an empty string
            //params.addElement( "" );
            params.addElement( blogid );            // the blog id
            // the user's WhatCounts login id
            params.addElement( userid );
            params.addElement( passwd );            // the user's password
            // Get at most three posts at a time
            params.addElement( new Integer(3) );
            Vector posts = (Vector) xmlrpc.execute( "metaWeblog.getRecentPosts",
                                                    params );
            Enumeration postEnum = posts.elements();
            while ( postEnum.hasMoreElements() )
            {
                // Print each post
                h = (Hashtable) postEnum.nextElement();
                String title = (String) h.get( "title" );
                String description = (String) h.get( "description" );
                String link = (String) h.get( "link" );
                String postid = (String) h.get( "postid" );
                Date dateCreated = (Date) h.get( "dateCreated" );
                System.out.println( "\n    " + title + " [" + postid + "]\n" );
                printLineWrap( description, 50, "    " );
                System.out.println( "\n    date created: " + dateCreated + ",
                                    permalink: " + link );
            }
        }
        System.out.println( "\nok." );
    }

    static void printLineWrap( String text, int maxLength, String linePrefix )
    {
        //System.out.println( linePrefix + text );
        if ( text != null && text.length() > 0 )
        {
            BreakIterator boundary = BreakIterator.getLineInstance();
            boundary.setText( text );
            int start = boundary.first();
            int end = boundary.next();
            int lineLength = 0;
```

```
                System.out.print( linePrefix );
                while ( end != BreakIterator.DONE )
                {
                    String word = text.substring( start, end );
                    lineLength = lineLength + word.length();
                    if ( lineLength >= maxLength )
                    {
                        System.out.println();
                        System.out.print( linePrefix );
                        lineLength = word.length();
                    }
                    System.out.print( word );
                    start = end;
                    end = boundary.next();
                }
            }
        }
    }
```

10. Frequently Asked Questions

Q: What kind of communications are performed by the API? How are messages sent?

A: The API utilizes HTTP-like communications over port 80. This allows it to operate in most corporate environments.

Q: How fast is the API?

A: Performance depends somewhat on network bandwidth and server load on our end. In general it should be faster than the web based UI. Try some of the test programs to see for yourself.

Q: We need some additional methods not described by this document. How often is the API updated?

A: The API is updated regularly and we're always looking to support new applications and uses of the interface. If you have a method or feature that you would like to see included, email to your Technical Account Manager or support@whatcounts.com describing your request.

Q: How can the latest version of Microsoft's .NET framework be obtained?

A: Microsoft makes downloads of the .NET framework available at:
http://msdn.microsoft.com/netframework/technologyinfo/howtoget/default.aspx

You can also obtain it via your system's update facility.

11. Return Codes and Constants

The following is a list of integer values that may be returned the API methods. Also included are constant values that can be used as parameters in certain methods.

Return codes

```
int API_RESPONSE_OK = 0;
int API_RESPONSE_FAILURE = 1;
int API_RESPONSE_APIUSERMANAGER_INIT = 2;
int API_RESPONSE_APIUSER_NOTFOUND = 3;
int API_RESPONSE_INVALID_VERSION = 4;
int API_RESPONSE_INVALID_KEY = 5;
int API_RESPONSE_INVALID_DATA = 6;
int API_RESPONSE_APIURL_NOTFOUND = 7;
int API_RESPONSE_INVALID_REALM = 8;
int API_RESPONSE_INVALID_ENVELOPE = 9;
int API_RESPONSE_UNKNOWN_COMMAND = 10;
int API_RESPONSE_SESSION_MANAGER_INIT = 11;
int API_RESPONSE_SESSION_NOTFOUND = 12;
int API_RESPONSE_INVALID_IP = 13;
int API_RESPONSE_EXISTS = 14;
int API_RESPONSE_NOT_FOUND = 15;
int API_RESPONSE_NOT_IMPLEMENTED = 16;
int API_RESPONSE_ACCESS_DENIED = 17;
```

Constants

```
int ENUM_TYPE_NUMBERS = 0;
int ENUM_TYPE_NAMES = 1;
int ENUM_TYPE_XML = 2;

int TEMPLATE_PART_SUBJECT = 0;
int TEMPLATE_PART_DESCRIPTION = 1;
int TEMPLATE_PART_NOTES = 2;
int TEMPLATE_PART_PLAINTEXT = 3;
int TEMPLATE_PART_HTML = 4;
```

```
int TEMPLATE_PART_AOL = 5;
int TEMPLATE_PART_WAP = 6;

int ARTICLE_PART_NAME = 0;
int ARTICLE_PART_TITLE = 1;
int ARTICLE_PART_DESCRIPTION = 2;
int ARTICLE_PART_DECK = 3;
int ARTICLE_PART_CALLOUT = 4;
int ARTICLE_PART_BODY = 5;
int ARTICLE_PART_AUTHOR_NAME = 6;
int ARTICLE_PART_AUTHOR_EMAIL = 7;
int ARTICLE_PART_AUTHOR_BIO = 8;

int DATA_TYPE_INT = 0;
int DATA_TYPE_STRING = 1;
int DATA_TYPE_DATE = 2;
int DATA_TYPE_BIGSTRING = 3;
int DATA_TYPE_NTEXT = 6;
int DATA_TYPE_FLOAT = 7;


int LIST_ATTRIBUTE_ADV_CUSTOM_FORM = 100;
int LIST_ATTRIBUTE_ADV_ENVELOPE = 101;
int LIST_ATTRIBUTE_ADV_LANDINGPAGE = 102;
int LIST_ATTRIBUTE_ADV_ERROR_LANDING_PAGE = 103;
int LIST_ATTRIBUTE_ADV_PUBLISH_TO_BLOG = 104;
int LIST_ATTRIBUTE_ADV_PUBLISH_TO_BLOG_ID = 105;
int LIST_ATTRIBUTE_ADV_PUBLISH_TO_BLOG_TEMPLATE_ID = 106;
int LIST_ATTRIBUTE_ADV_RSS_DEFAULT_TITLE = 107;
int LIST_ATTRIBUTE_ADV_RSS_DEFAULT_DESCRIPTION = 108;
int LIST_ATTRIBUTE_ADV_RSS_NO_CONTENT_MESSAGE = 109;
int LIST_ATTRIBUTE_ADV_DEDUPE_ON_SEND = 110;
int LIST_ATTRIBUTE_ADV_WORKFLOW_APPROVAL_REQUIRED = 111;
int LIST_ATTRIBUTE_ADV_WORKFLOW_APPROVAL_LIST = 112;
int LIST_ATTRIBUTE_ADV_WORKFLOW_APPROVAL_FORMAT = 113;
int LIST_ATTRIBUTE_ADV_HABEAS_ENABLED = 114;
int LIST_ATTRIBUTE_ADV_STICKY_CAMPAIGNS_ENABLED = 115;
int LIST_ATTRIBUTE_ADV_SET_DATA_MACRO_ID = 116;

int LIST_ATTRIBUTE_SYNC_ENABLED = 200;
```

```
int LIST_ATTRIBUTE_SYNC_SEND_EVENT_MACRO_ID = 201;
int LIST_ATTRIBUTE_SYNC_OPEN_EVENT_MACRO_ID = 202;
int LIST_ATTRIBUTE_SYNC_CLICK_EVENT_MACRO_ID = 203;
int LIST_ATTRIBUTE_SYNC_OPTIN_EVENT_MACRO_ID = 204;
int LIST_ATTRIBUTE_SYNC_OPTOUT_EVENT_MACRO_ID = 205;
int LIST_ATTRIBUTE_SYNC_PREDEPLOY_EVENT_MACRO_ID = 206;
int LIST_ATTRIBUTE_SYNC_POSTDEPLY_EVENT_MACRO_ID = 207;
int LIST_ATTRIBUTE_SYNC_SOFTBOUNCE_EVENT_MACRO_ID = 208;
int LIST_ATTRIBUTE_SYNC_HARDBOUNCE_EVENT_MACRO_ID = 209;
```

12. Support Information

Support issues should be directed by email to your Technical Account Manager or support@whatcounts.com.