Wrapper Overview

Goal

Improve ease of use and functionality for ASP.NET users of the MailChimp API by providing a class library wrapper and Testing UI.

Approach

- Use .NET classes to wrap functionality of the API, pass input parameters, and output results.
- Provide for both Xml-Rpc and Serial (GET/POST) access to API information via strongly type classes for input that format and create the appropriate API calls for you. Multiple class signatures provided.
- Provide a consistent output type, using .Net generic types in place of arrays and structs; e.g., List<string> instead of string[], Dictionary<string, string> for associative arrays, etc.
- Provide choice of unformatted output types; XML, JSON, and PHP.
- Provide custom default values that can override MailChimp api defaults; e.g., limit=200 vs. 1000.
- Provide validation (selectable, on/off) for parameter completeness and type checking.
- Return wrapper validation messages and returned API errors messages as generic lists for ease of checking and reporting.
- Results, other than simple integer, string or Boolean values, are defined as custom classes making them easier to use, especially in data bound controls.

Specifications & Requirements

- MailChimp API functionality through the v1.2.7 announcement dated 16-Jul-2010.
- .NET 3.5 required
- Cook Computing xml-rpc.net v2.5.0 library is used & included

Sample Wrapper API Method Calls

Lists

C#

```
      // A real simple request (using all default settings)

      public void lists_method ()
      {
         // input parameters, using default apikey
         listsInput input = new listsInput( MCAPISettings.default_apikey );

         // execution
         lists cmd = new lists( input );
         listsOutput output = cmd.Execute();

         // format output (Assuming a User control named show_lists)
         show_lists1.Display( output );
      }
```

VB

```
   ' A real simple request (using all default settings)

   Public Sub lists_method()

      ' input parameters, using default apikey
      Dim input As listsInput = New listsInput(MCAPISettings.default_apikey)

      ' execution
      Dim cmd As lists = New lists(input)
      Dim output As listsOutput = cmd.Execute()

      ' format output (Assuming a User control named show_lists)
      show_lists1.Display(output)

   End Sub
```

listBatchSubscribe

C#

```
// ---------------------------------------------------------------------
// A little more intense request (listBatchSubscribe using Serial/JSON)
// assumes a collection was created elsewhere from database info
// with email, email_type, first name, last name, and last_payment

public void listBatchSubscribe_method ( List<Subscriber> SubscriberList )
{
   listBatchSubscribeInput input = new listBatchSubscribeInput();
   // any directive overrides
   input.api_Validate = true;
   input.api_AccessType = EnumValues.AccessType.Serial;
   input.api_OutputType = EnumValues.OutputType.JSON;
   // method parameters
   input.parms.apikey = MCAPISettings.default_apikey;
   input.parms.id = "YourListId";
   input.parms.double_optin = false;
   input.parms.replace_interests = true;
   input.parms.update_existing = true;
   //
   List<Dictionary<string, object>> batch =
      new List<Dictionary<string, object>>();
   foreach ( Subscriber sub_rec in SubscriberList )
   {
      Dictionary<string, object> entry = new Dictionary<string, object>();
      entry.Add( "EMAIL", sub_rec.email );
      entry.Add( "EMAIL_TYPE", sub_rec.email_type );
      entry.Add( "FNAME", sub_rec.first_name );
      entry.Add( "LNAME", sub_rec.last_name );
      DateTime next_payment = sub_rec.last_payment.AddMonths( 1 );
      entry.Add( "NEXTPAY", next_payment );
      batch.Add( entry );
   }
   input.parms.batch = batch;

   // execution
   listBatchSubscribe cmd = new listBatchSubscribe( input );
   listBatchSubscribeOutput output = cmd.Execute();

   // output, format with user control
   if ( output.api_ErrorMessages.Count > 0 )
   {
      showResults( output.api_Request, output.api_Response,      // raw data
      output.api_ErrorMessages, output.api_ValidatorMessages ); // & errors
   }
   else
   {
      show_listBatch1.Display( output );
   }
}
```

VB

```
' A little more intense request (listBatchSubscribe using Serial/JSON)
' assumes a collection was created elsewhere from database info
' with email, email_type, first name, last name, and last_payment

Public Sub listBatchSubscribe_method(ByVal SubscriberList As List(Of
Subscriber))

   Dim input As listBatchSubscribeInput = New listBatchSubscribeInput()

   ' any directive overrides
   input.api_Validate = True
   input.api_AccessType = EnumValues.AccessType.Serial
   input.api_OutputType = EnumValues.OutputType.JSON
   '  method parameters
  input.parms.apikey = MCAPISettings.default_apikey;
  input.parms.id = "YourListId";
  input.parms.double_optin = false;
  input.parms.replace_interests = true;
  input.parms.update_existing = true;
   '
   Dim batch As List(Of Dictionary(Of String, Object)) = _
         New List(Of Dictionary(Of String, Object))

   For Each sub_rec As Subscriber In SubscriberList

      Dim entry As Dictionary(Of String, Object) = _
         New Dictionary(Of String, Object)

      entry.Add("EMAIL", sub_rec.email)
      entry.Add("EMAIL_TYPE", sub_rec.email_type)
      entry.Add("FNAME", sub_rec.first_name)
      entry.Add("LNAME", sub_rec.last_name)
      Dim next_payment As DateTime = sub_rec.last_payment.AddMonths(1)
      entry.Add("NEXTPAY", next_payment)
      batch.Add(entry)

   Next
   input.parms.batch = batch

   ' execution
   Dim cmd As listBatchSubscribe = New listBatchSubscribe(input)
   Dim output As listBatchSubscribeOutput = cmd.Execute()

   ' output, format with user control
   If output.api_ErrorMessages.Count > 0 Then
      ' raw data & errors
      showresults(output.api_Request, output.api_Response, _
         output.api_ErrorMessages, output.api_ValidatorMessages)
   Else
      show_listBatch1.Display(output)
   End If

End Sub
```

Installation

The project is open source and hosted at http://perceptivemcapi.codeplex.com/

In addition to this documentation, the two other files in the recommended download .zip file are the PerceptiveMCAPI.dll – the wrapper; and the second is the CookComputing.XmlRpcV2.dll. These are the only files you need to use the PerceptiveMCAPI .Net wrapper.

If you use a website, copy the 2 DLLs to your `bin` folder.
If you use web applications, copy the DLLs to any folder, right-click on the `Reference` folder, select `Add Reference…`, browse to the folder where you put the files, select `PerceptiveMCAPI.dll` and click `OK`. You do not need to reference the CookComputing DLL.

Settings & such

The following declarations are required in programs using the wrapper:

C#

```
      using PerceptiveMCAPI;
      using PerceptiveMCAPI.Types;
      using PerceptiveMCAPI.Methods;
```

VB

```
      Imports PerceptiveMCAPI
      Imports PerceptiveMCAPI.Methods
      Imports PerceptiveMCAPI.Types
```

A configuration class `(MCAPISettings)` contains the MailChimp API default values for optional method parameters, as well as the API Control and Wrapper directive default values. If you wish to change these defaults for your application, you can include a custom configuration section in the web.config file.
You need specify only those values for which you wish to change the defaults.

Example:

```
<configuration>
   <configSections>
      <section name="MailChimpAPIconfig"
      type="PerceptiveMCAPI.MCAPISettings, PerceptiveMCAPI"/>
…… other entries…
</configSections>

<MailChimpAPIconfig>
      <MCAPI
      SecureAccess="False"
      Validate="False"
      DataCenter="auto"
      apikey="YourApiKey-us1"
      delete_member="True"
      send_goodbye="False"
      double_optin="False"
      send_welcome="True" />
</MailChimpAPIconfig>
```

A list of current Api setting values can be obtained by executing `MCAPISettings.ListAPISettings();`

Test Install

A quick and easy way to test the install is to create a new page, add the using/Imports statements, drop a gridview control on the page, and put the following in the code-behind to list the Api settings …

C#

```
    protected void Page_Load( object sender, EventArgs e )
    {
       string apikey = "YourApikey";   // or default to config

       GridView1.DataSource = MCAPISettings.ListAPISettings().ToList();
       GridView1.DataBind();
    }
```

VB

```
 Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load

    Dim apikey As String = "YourApikey"   ' or default to config

    GridView1.DataSource = MCAPISettings.ListAPISettings().ToList()
    GridView1.DataBind()

 End Sub
```

Solution Setup for Source Code

Obviously there are a number of ways to set this up, but here is one..

1) Download the latest Change Set, and if you don't have the CookComputing.XmlRpcV2.dll yet, you can download the 'Recommended' release which has a copy in it.
2) Create a new class library named PerceptiveMCAPI, copy the source folders under the PerceptiveMCAPI\trunk directory into your project folder.
3) 'Include' the folders into your project.
4) Add the XmlRpcV2.dll into a folder (such as 'bin') and add a reference to it.
5) Make sure that System.Configuration, System.Web, and System.Web.Extensions are listed as References – or add them.
6) Do a build of the class library – all should be good at this point.

If you want to use the Testing UI Source as a project in your solution…

1) Create a new EMPTY web application in your solution.
2) Copy the source folders under the PerceptiveMCAPI_UI\trunk\PerceptiveMCAPI_UI directory into your project folder ; omit the web.config and 'project type' items. (project type items should not have been svn'd and will be removed in a future release – sorry about that)
3) Include what you added in step #2 into the project. Include any setting overrides you need in the web.config section `MailChimpAPIconfig`
4) Add a reference to the PerceptiveMCAPI class library project.
5) Do a Build -- and it should be successful.

Types

Each MailChimp API method has multiple entries associated with it that are named;
*method*Input, *method*Output, *method*Parms, and optionally *method*Results, etc.

"Input" has the input parameters for the method to execute plus the API Wrapper directives.
"Output" is the result of the execution; it contains an 'inputParms' variable (of the "Parms" class), and a 'result' variable -- either a simple type, or a "Results" class type.
"Parms" has the method input parameters and are contained in the output class for convenience.
"Results" class is used when the output is not a simple value such as a string, integer, etc.
"_ValidateInput" is a partial class entry to the `ValidateInput` class.
"_Validator" class is used when method specific validation is required.

Each "Input" type class will have multiple constructors defined; an empty constructor, a 'complete' constructor, a "Parms" object constructor, and others as a matter of convenience.
Example: `apikeysInput input = new apikeysInput( username, password, apikey );`

```
Wrapper settings can be specified after the input instance is created if needed;
      // -----------------------------------------------------------------
      // **** access, output, method, encode, and validate
      input.api_AccessType = EnumValues.AccessType.Serial;
      input.api_OutputType = EnumValues.OutputType.JSON;
      input.api_Validate = false;
      // -----------------------------------------------------------------
```

Most wrapper and api settings that are a selection of values or defaults are defined as enumerated values for ease of selection, consistency, and maintainability.
These and other constant values are found in the `EnumValues` class.

Base Types

Each MailChimp API 'Input' and 'Output' type inherits from a base type; api_BaseInput and api_BaseOutput.

The 'Input' base class has the wrapper directives and method to call. The 'Output' base class has a copy of the wrapper directives used for input, separate collections for Validation Messages and Api Errors, and the request and response strings for serial requests.

Intermediate Result Types

When results returned from the MailChimp Api are not simple types such as bool, string, etc., an 'intermediate' result class is defined and returned as: x_*methodName* or x_*methodName* []. These intermediate classes are contained in the PerceptiveMCAPI.Types.Internal namespace.

These intermediate result classes are converted (within the method "Output" class) to the 'external' class definition that is returned as the `methodResults` or `List< methodResults >` value in the Output.result field.

There are a number of apiMethods that return complex fields within the intermediate class. These fields may be defined as `object` or `XmlRpcStruct` types, or may also have a 'sub-class' defined for convenience.

Methods

Each MailChimp API method is defined as a separate class (named the same as the MailChimp api method) that performs access (Xml-Rpc or Serial) , gets the response -- including unformatted values for XML, JSON, and PHP -- and formats the output into .NET friendly classes (except PHP, which only returns the unformatted response).

Each api method exposes a public function 'Execute()' which returns the Output class and may be invoked in a number of ways as there are overloads for both the constructor and Execute() method;

```
listsInput input = new listsInput( apikey );
lists cmd = new lists( input );
listsOutput output = cmd.Execute();
-- or –
listsInput input = new listsInput( apikey );
listsOutput output = new lists( input ).Execute();
-- or --
apikeysOutput output = new lists().Execute(new listsInput( apikey ));
```

Validation

Validation is optional. For the most part, validation only checks for the existence and type of parameters before submitting the request to the MailChimp API. Limited validation of parameter key/values is performed.

Validation is optional, can be specified as a default wrapper directive, or specified on an individual request basis. Validation is NOT recommended for production use where a large volume of requests may affect performance.

Each method has a `ValidateInput` partial class segment that issues validation requests for the input parameters. The requests are made to the `Validator` class and optionally to a method specific validation class using the naming convention, *method*_Validator. The `ValidateInput` partial segment can call the *method*_Validator class for method specific validation and custom business-rule validation if required.

The 'base' `ValidateInput` segment calls the `Validator` class for the api_BaseInput parameters.

The `Validator` class contains methods that validate common parameters such as the apikey, or common features such as a required value check, or type checking.

If any validation results in an 'Error' level message, no call to the MailChimp api is made.
Validation messages are returned in the Output class in the `api_ValidatorMessages` collection.

Limitations and Known Issues

Limitations

- Method specific limitations are listed in the method notes.
- Unless otherwise mentioned, all Api methods and features are supported through the current announcements and updates according to the MailChimp Api Changelog.

Known Issues

- System-wide and method specific issues are documented in the CodePlex Project Issue Tracker section.
- CodePlex Issue Tracker doesn't allow for indicating items are 'external' issue items – so those items that are an issue and that have been reported to MailChimp for investigation will indicate it in the comments. Issue status will be ‘pending’ until resolution. Issues being investigated and worked on within the project team will be 'assigned' and have a status of 'active'.

Field Type Notes

DateTime

Input parameters: When defined as `DateTime.MinValue`, date paramenters are treated as ‘empty’ and not passed to the api request.
Output results: Rather than use nullable `DateTime` definitions, when a date value returned from MailChimp is null or empty `DateTime.MinValue` is output to the result value.

Boolean

Input parameters: Boolean values are always passed. Default settings are used unless overridden.
Output results: Raw results from the MailChimp Api are typically returned as 1 and 0. These are converted to Boolean, True and False.

Enumerated Values

Input parameters: Enumerated Values are always passed. They are defined in the partial class `PerceptiveMCAPI.EnumValues`. Default settings are used unless overridden. There is often a `NotSpecified` or similar value defined, along with the values specified in the MailChimp Api.
Output results: Raw results from the MailChimp Api are converted to their `EnumValues` equivalent.

Values are generally converted for the MC Api using the ToString() method and string values can be converted to the enumeration using a generic method in the apiHelper class method, `GetEnumFromString`.

Method Notes

Overview

In general, the PerceptiveMCAPI .NET wrapper works as indicated in MailChimp's API documentation on their website. Temporary limitations, known issues, and feature requests are tracked on the CodePlex Issue Tracker. When there is a deviation from the documentation for an Api method, a limitation on functionality, or clarification is required for most efficient use, then a 'Method Note' will be included in this section.

Method Notes -- Security & Helper

apikeyExpire

This method was changed to require an additional parameter for input; apikeyToExpire. This allows for *not* using the 'request' apikey as the one that will be expired, although there is nothing preventing the two values from being the same. This provides for edge cases where apikeys are assigned to individuals or clients and usage is logged; it provides a way to keep the usage log 'correct' and not lose who 'expired' whom.

ecommAddOrder

ecommAddOrder, campaignEcommAddOrder, and campaignEcommOrders all use the same abstract class to inherit their definition of the 'Order' and all use the same item class definition. Whilst this creates consistency between these methods, it causes some input parameters and output results to be different from the MailChimp Api documentation.

getAccountDetails

Result field 'date' in the orders array has been renamed 'order_date'

generateText

Input parm 'content' is defined as a generic object ('mixed' in the documentation). The two expected object types are `String` for all 'type' entries except 'template', which expects a `Dictionary<string, string>` object type – where the key is the template section name and the value is the template section content.

Note: GET Serial requests – large/complex html entries break, small ones are ok. Error … [414] Request-Uri Too Large … is returned. Only POST serial requests are recommended for this method.

inlineCss

Note: GET Serial requests – large/complex html entries break, small ones are ok. Error … [414] Request-Uri Too Large … is returned. Only POST Serial requests are recommended for this method.

Method Notes -- Campaign Related

campaignCreate

Limitation: Content type 'archive' is not currently supported – a future build will.

campaignEcommAddOrder

See note above for ecommAddOrder.

campaignSendTest

M.C. documentation indicates that a 'null' value will send 'both formats'; this means that it sends a single multi-part mime version as opposed to a text-only or html-only version. To select the multi-part mime option, set the send_type to the 'NotSpecified' enumeration.

campaignShareReport

The opts parameter is defined as `Dictionary<string, object>`. All 'object' values are `string` with the exception of 'theme' which expects a type of `Dictionary<string, string>`.

campaignUpdate

The campaignUpdate method while straightforward in execution, is a little difficult to explain, but here it goes. The `value` parameter field is defind as an `object` type, with a number of overloads; the simple types of int, string, and bool, as well as `Dictionary<string, string>`, and two custom classes `campaignTracking` and `campaignSegmentOptions`.

By referencing the campaignCreate method documentation, notice the left-most columns with the names; options, content, segment_opts, and type_opts. They provide the clue on how to use the 'name' & 'value' fields.

options – The 'name' for entries in this group is the field name listed; subject, title, etc. The 'value' fields are all simple types except for 'tracking' which will use the `campaignTracking` class and 'analytics' which will use the `Dictionary<string, string>` just like in the campaign create method.

content – The `'name'` for all entries in this group is `'content'` – then the entries all use the `Dictionary<string, string>` option where the key is the parameter name (html, text, url, etc.) and the value is the content value you want to update.

segment_opts – Use the `campaignSegmentOptions` just as was done for the campaignCreate option. The segment_opts can be completely cleared by selecting this option, and passing no segment definitions. No clear for individual segment line item definitions is available.

type_opts – very much like the 'content' group, except that the 'name' for the entries reflect the campaign type. Use 'rss' for RSS campaigns, 'absplit' for A/B Split campaigns, and 'auto' for AutoResponder campaigns. Then use the `Dictionary<string, string>` the same as in the 'content' group section.

campaigns

Filters are defined as `Dictionary<string, object>` types. The string key is the filter name; the object value is the string, int, date, or bool type value as defined for the specific filter.

Method Notes -- Campaign Stats & AIM

campaignClickStats

Result set is 'flattened' compared with the documented layout.

campaignEcommOrders

See note above for ecommAddOrder.

campaignEmailDomainPerformance

Percentage result fields are defined as float instead of integer.

Method Notes -- List Related

listBatchSubscribe

'batch' is defined as `List<Dictionary<string, object>>` -- Each entry in the List is a Dictionary containing the MergeVar key/value pairs for the subscriber: FNAME, LNAME, etc. – as well as the special values of EMAIL and EMAIL_TYPE. Merge_vars are defined as described in listSubscribe.

listGrowthHistory

The 'month' result field has been separated into two integer fields; 'year' and 'month'

listInterestGroupingAdd

groups parameter is defined as `List<string>`.

listInterestGroupings

groups result property is defined as `List<listInterestGroupingsGroups>`.

listMemberInfo

I found it cumbersome to work with the groupings information 'buried' within the merges collection, so I split this out into a `List<interestGroupings> groupings` property.

listMergeVarAdd

The 'req' field is renamed 'options' and is of the type `mergevar_options`.

listMergeVarUpdate

The 'options' field is of the type `mergevar_options`.
There is no ability to change the 'name' field in the api.

listMergeVars

Results include more fields than indicated in the documentation. Two of the 'extra' result fields were named with the .NET reserved words, 'public' and 'default'. (bool) public is renamed to 'isPublic'; (string) default is renamed to 'defaultValue'

listSubscribe

Merge_vars are defined as `Dictionary<string, object>` where:

- If the string name is `Groupings`, the object value expected is of type: `List<interestGroupings>`
- If the object type is `DateTime`, the value is converted to an acceptable api string equivalent
- if the object type is `Dictionary<string, object>` (used for address array), the Key-Value pairs are formated as required
- Any other object type is passed to the api **as-is.**

listUpdateMember

For field 'email_type' – The enumerated value 'NotSpecified' will leave the current email_type value intact. Input constructors without an email_type parameter default to 'NotSpecified', or it can be explicitly passed.
Merge_vars are defined as described in listSubscribe.

Testing User Interface

TestUI.aspx & TestCode.aspx

Overview

The TestUI page provides a convenient means to enter, validate, and display output of the MailChimp API methods.
The TestCode page is thrown in as a lighter sample code page, without the user controls of the UI, although it's easy to put in output user controls as a few are in the TestCode sample.

Features listed below are for the TestUI page.

Features

- Selectable access method (Xml-Rpc or Serial) and output types for Serial access (XML, JSON, or PHP).
- Display of request, formatted output, and unformatted output for serial requests.
    - Xml-Rpc access will contain ** Not Supplied ** for api_Request & api_Response field values. A static class `apiXmlRpcRequestDisplayFormat` is used to format Xml-Rpc input parameters for display in the UI.
    - Serial access requests will contain the constructed url in the api_Request field as it was sent to the `HttpWebRequest` class.
    - Serial access requests will contain the unformatted response in the api_Response field as returned from the `HttpWebResponse` class.
- User controls allow easy input for testing all Api methods.
- Formatted responses are shown in user controls, with complex results bound to listview, gridview, dataview, or detailsview controls.
- Display of Api Settings used for method execution is available.

Limitations

The limitations and notes listed here pertain to the Testing UI only, and DO NOT apply to the .NET wrapper itself.

Input Limitations

- Groupings entry is not supported in listBatchSubscribe.

Output Limitations

- None of note

UI Entry Notes –

listBatchSubscribe

merge_var entries are entered like request string values, e.g.;
FNAME=David&account=123&billing=monthly

ListSubscribe & listUpdateMember

merge_var entries are entered like request string values, e.g.;
FNAME=David&account=123&billing=monthly

EXCEPT

when the merge tag is GROUPINGS (case insensitive) AND there are multiple groupings entered, the multiple groupings must be separated by any one of { `"+", "|", ";"` }, e.g.,
id=69&name=MCAPI &groups= beta tester+id=33&groups=dogs,birds

RoadMap - History

v1.2.5 – Stable

- See change log at end of this document
- Wrapper – Restructure source code for Execute & Output classes (see change log)
- Wrapper – Internal improvements & bug fixes
- UI – allow display of result ToString() values

v1.2.4 – Deprecated

- *Support for new MailChimp functionality announcements as available (v1.2.6, v1.2.7).*
- *Wrapper – Restructure input & parms classes (see change log)*
- *Wrapper – Allow input parms to be passed as an object, support custom input objects.*

v1.2.3 – Deprecated

- *MailChimp Api v1.2.5 – All release changes supported*
- *Complete Codeplex issues 2271, 2355, 3397, 3610, & 3892.*
- *Wrapper – Changes to MergeVar handling in various Methods & Types*
- *Wrapper – Serial requests as POST form data.*
- *UI – remove limitation on listBatchSubscribe() entry*

v1.2.2 – Deprecated

- *MailChimp Api v1.2.4 – All release changes supported*
- *MailChimp Api v1.2.3 – All release changes supported*
- *Geo-location Segmentation options per MailChimp Api 2009-Oct-02 notices supported.*
- *Wrapper -- Cleanup of PerceptiveMCAPI.Types.Internal namespace and more consistent approach to complex return values from the Api (phase 1)*
- *UI -- allow selection of specific apikey to use for requests*
- *UI -- user control enhancements & clean-up*

RoadMap – Future

Notes on .NET & VS versions

- v1.2.x – all versions developed using VS2008 targeting .NET 3.5
- v1.3.0 – development using vs2010, targeting .NET 3.5
- v1.3.x – future development using vs2010, will target the .NET 4 framework

The *Solution Setup for Source Code* section has been revised to be used without any project/solution files as those will be removed from the repository at CodePlex due to VS compatability issues.

v1.2.x – Maintenance only.

- With MC's release of their v1.3.0 api, the 1.2.x will be branched and only critical fixes applied.

v1.3.0 – In Progress

- Support for MailChimp Api v 1.3.0 functionality
- This is targeted for rollout as follows:
    - First Beta – prior to Christmas. Details forthcoming.
    - Release – Target Jan 10.

Export Api v1.0 – Planning

- Will be a separate source project and .dll, but is expected to have some dependency on wrapper functions initially.
- Will be supported for either 1.2.x or 1.3.x of the wrapper.
- Target – End of January.

v1.3.1 – Planning

- Full + 'Light' versions.
- Wrapper – Allow external (non-web.config) configuration handling.
- Wrapper – JSON input support, and additional output type, JSON-FMT for .NET formatted result class, as opposed to default 'raw' JSON result from MailChimp api – valid for all input types.
- Wrapper -- `ToString()`overrides for more types, improve existing.
- Wrapper -- Validation improvements for complex types (campaignCreate, etc.)

v1.3.x – Future Potential

- Wrapper -- Allow custom error messages to override default MC messages
- Wrapper -- Allow Api activity logging
- Wrapper -- Provide ability to restrict specific methods by Apikey

Appendex A – Wrapper Directives

All current settings can be retrieved by calling the MCAPISettings.ListAPISettings() method.
Wrapper defaults can be over-ridden on a per request basis.

Api Control

api_Url – this value is the MailChimp api endpoint. The value cannot be changed directly.
It is composed of the following…plus run-time values for DataCenter,

- The api 'base' domain -- `api.mailchimp.com` (hard-coded)
- The value of the api_Version directive (hard-coded)

api_Version – the MailChimp api version number

- 1.2 – value is hard-coded

api_UserAgent – the User Agent value specified by MailChimp to be used

- MailChimp.Net / 1.2 – value is hard-coded

api_DataCenter – as specified in your account by MailChimp

- defaults to auto, or must match the last 3 digits of your apikey

api_xmlrpc_UseStringTag, api_xmlrpc_Indentation, and api_xmlrpc_EnableCompression

- affects xmlrpc requests only –provides values to reduce the request size
- defaults are; UseStringTag=false, Indentation =0, EnableCompression =true

Wrapper Defaults

api_AccessType – XmlRpc or Serial

- XmlRpc – uses the CookComputing xml-rpc library to make the calls
- Serial – uses http GET or POST depending on the MethodType directive

api_OutputType – XmlRpc, JSON, XML, PHP

- XmlRpc – only for XmlRpc requests. Only returns the formatted result.
- JSON & XML – only for Serial requests. Returns the formatted result AND the raw JSON or XML string.
- PHP – only returns the raw PHP string.

api_MethodType – GET or POST for Serial Requests

- GET – Creates an Http GET request string. Complex requests could result in 'Request too large' errors.
- POST – Creates a post request with ContentType = `application/x-www-form-urlencoded`. All parameters other than 'output' and 'method' are passed in the message body.

api_EncodeRequest – true or false

- This directive is ignored for other than Serial-GET requests. The default is true. The MailChimp Api expects encoded values, but it is sometimes helpful for testing to see the un-encoded values.

api_Validate – true or false

- Whether to do validation prior to method call. Validation errors will not allow execution to continue.

api_Timeout – timeout for api requests

- in milliseconds, default is 90,000

api_SecureAccess – whether to use SSL calls

- true or false – defaults to false (determines use of http:// or https://)

Appendex B – Wrapper Source

Directory Structure

_config [future use]

- Eventually to be used to store one or more configuration files to be used as a replacement for the web.config section currently used.
- Will also contain other files such as custom error definitions

_internal

- BaseInput, BaseOutput, Error, ValidatorMessage, and common-use structures

Application

- BaseExecute, Exception and Validation, config settings, EnumValues, apiHelper, and xml-rpc calls

bin

- contains the CookComputing.XmlRpcV2.dll

Campaign

- Folder for each method defined in MailChimp documentation section "Campaign Related"

CampaignAIM

- Folder for each method defined in MailChimp documentation section "Campaign AIM"

CampaignStats

- Folder for each method defined in MailChimp documentation section "Campaign Stats"

Helper

- Folder for each method defined in MailChimp documentation section "Helper"

List

- Folder for each method defined in MailChimp documentation section "List Related"

Security

- Folder for each method defined in MailChimp documentation section "Security Related"

Method Entries

method.cd – Class diagram
method.cs – The 'execute' class for method processing.
*method*_ValidateInput.cs – Partial class that acts as a controller for pre-execute validation .
*method*_Validator.cs – method specific or custom validation routines, if required.
*method*Input.cs – contains both the api directives and method parameters
*method*Output.cs –api directives, original parameters, errors & messages, and the raw & formatted response
*method*Parms.cs – method parameters
*method*Results.cs – formatted response definition, if required
x_*method*.cs -- intermediate result definition, if required

PerceptiveMCAPI – v 1.2.5: Change Log

This is a 'quick-hit release that does not include any MailChimp Api changes – only internal improvements and source code restructuring.

NOTE: **CookComputing xml-rpc version v2.5.0.0 is now being used for xml-rpc calls.**

Source Code Restructuring

None of these changes result in breaking the existing behavior of applications using the .NET wrapper.

- Output format functions moved from the 'execute' class to the Output class – go figure
- Parameter conflict checking routine moved from Output class to Api_BaseExecute
- Moved non-helper type functions from apiHelepr to apiSerial (new class, see below) or apiXmlRpc
- Made apiHelper public, to allow use of common conversion routines in applications, if desired.
- Changed internal class property names; input->Input, output->Output
- Performed an 'Organized Usings->Remove and Sort' on all source; and other cleanup

apiSerial – New Class

- This class contains the static methods specific to serial access processing that used to be in the apiHelper class; the main being what used to be called '`ProcessSerialRequest`' and is now `apiSerial.ProcessRequest`.

Functional Enhancements

None of these changes result in breaking the existing behavior of applications using the .NET wrapper, but provide additional flexibility and features.

Additional Constructor and Execute Signatures for All Api Methods

- Constructor has two overrides; with & without Input class
- Execute() has two overrides; with & without Input Class
- Output class is now instantiated in the Execute() function
- Why? Multi-Account & long-running tasks, personal preference.

For multiple accounts, makes certain tasks easier/faster by not having to do a re-init of the class; for example if you wanted to get all your lists for mutiple accounts, you could take a list of apikeys and loop through and execute.

```
List<listsResults> lists_Results = new List<listsResults>();

lists cmd = new lists();
foreach ( string apikey in key_collection )
  {
    listsOutput output =
      cmd.Execute( new listsInput( new listsParms { apikey = apikey } ) );
     lists_Results.InsertRange( lists_Results.Count, output.result );
  }
```

For long-running tasks like listBatchSubscribe() – it helps to break the data into smaller chunks to reduce the chance of timeouts, makes recovery easier & faster, and implement a 'pipe-lining' of data type approach. You could already do this, but this just adds a bit more flexibility. A sample of the restructured 'Execute' class is shown below.

```
public class listAbuseReports : Api_BaseExecute
{
   private new listAbuseReportsInput Input { get; set; }
   private listAbuseReportsOutput Output { get; set; }
   // =============================================== constructors
   public listAbuseReports() { }
   // ---------
   public listAbuseReports( listAbuseReportsInput input )
        { Input = input; }
   // =============================================== Execute
   public listAbuseReportsOutput Execute()
   {
      // empty constructor & Execute() is bad
      if ( Input == null )
         throw new ArgumentException( "Input class not provided" );
      // do the deed
      Output = Execute( Input );
      return Output;
   }
   // ---------------------------------------------
   public listAbuseReportsOutput Execute( listAbuseReportsInput input )
   {
      Input = input;
      Output = new listAbuseReportsOutput( Input );
      ValidationIsOK = true;
      //------------------------- validate it, if requested
      if ( Input.api_Validate )
      {
         Output.api_ValidatorMessages = ValidateInput.Validate( Input );
         ValidationIsOK =
             ValidateInput.IsOK( Output.api_ValidatorMessages );
      }
      //------------------------- do it
      if ( ValidationIsOK )
      {
         // execute API & format output
         ApiRun( Input, input.parms.apikey );
      }
      // ---
      return Output;
   }
```

……….. Api calls go here – omitted for example.

MCAPISettings – Configuration, Api_BaseInput, Api_BaseOutput

- Timeout is now a 'Wrapper Control' value allowing it to be set on individual api calls.
- SecureAccess is now a 'Wrapper Control' value allowing it to be set on individual api calls.
- DataCenter now uses a value of 'auto' as the default configuration setting. See note below.
- The api_Url `MCAPISettings` value no longer returns the 'http://' or 'https://' or DataCenter portion of the api request URL – these are now applied at execution time.

apiXmlRpc changes

Because the xml-rpc proxy is defined as static, the Timeout & SecureAccess values need to be applied slightly differently, but they can be set for individual calls – sort of.

- If you don't need to change the xml-rpc on a call-by-call basis, you don't need to do anything – all works as before regarding getting the default values from the config settings.
- The apiXmlRpc class is now public, with two pubic methods available.
  - CreateProxy – requires 0 or 3 parameters; data center, secure access and timeout values. This re-creates the proxy with the new values (and reuses the other config values)
  - DeleteProxy – this sets the proxy to null, so the next time an xml-rpc api call is requested the proxy gets recreated with the default config values.
- Please remember that since the proxy is a static property, changes will stay in effect and be applied to ALL xml-rpc calls until the proxy is recreated.

Change Notes

DataCenter

The DataCenter relates to the MailChimp Api endpoints, and are directly related to the api key values that since September 2009 include the datacenter as part of the api key; e.g. -- apikeyvalue-us1.

The 'auto' configuration setting will take those last 3 positions of the apikey, and use them in the Submit URL for the datacenter portion, providing flexibility for multi-account scenarios.

If you have a single account or all your accounts are in a single data center, you may still assign the specific DataCenter in the configuration settings. Existing, specific configuration settings for the DataCenter value will override the default 'auto' setting, and must be removed to take advantage of the new option.

UI Enhancements

- Provide ability to show the ToString() results from the Output class

CodePlex Issues & Features Summary

Wrapper

- **Fixed:** CodePlex #5193 – Xml-Rpc access only, campaignCreate() would throw an error when some properties of the campaignCreateOptions class were missing.
- **Fixed:** CodePlex #5242 – Serial –XML ouput only, listInterestGroupingDel and listInterestGroupingUpdate api methods, always returned false for result, regardless of the success or failure of the action for XML output only – other output types were OK.

Testing UI

- **Fixed:** CodePlex #5212 – sub_MergeVars.ascx.cs was picking up the wrong value for entries 2 -10, only the first entry was correct. The key was being returned for the value due to a naming problem.
- **Feature:** CodePlex #5238 – UI now has option to show the ToString() Output class value.

v1.2.5.1; Beta 1 – Fixes and Changes from Alpha source

- Moved FormatCampaignOptions() method from campaignCreate.cs to apiXmlRpc.cs
- FIX: CodePlex # 5357, campaignCreate() & campaignUpdate(), Serial only – The 'extra' segment option not applied when specified.
- FIX: campaignCreate(), Xml-Rpc only – changed 'to_email' so that a blank value is passed if not specified. This makes the default behavior on create be NO personalization, instead of the *|FNAME|* for the email; if that value is desired, the parameter value must be specified.
- FIX: campaigns(), Xml-Rpc only – if status & type filter values were specified, a program error occurred.
- NEW: campaignEmailStatsAIM() & campaignEmailStatsAIMAll() – a new field was added to the results of these methods – ip – the ip address from the open/click action.
- NEW: campaignEepUrlStats() – Two completely new sections of data were added; 'Click" and 'Referrers', when MC went 'social'. Significant changes to the result set were made, but no documentation was available at this time. Currently only the 'click' information has been added based on 'best guess' from a client statistics dump, that did not include any 'referrers' information. Expect changes when more documentation is available from MailChimp.
- FIX: ecommAddOrder(), Serial only – the result was not properly returned, and so would always indicate false, event though the order was added properly. This was only in Alpha code, not in the previous production release.
- FIX: getAccountDetails(), XML output – boolean values were always returning false.
- FIX: getAccountDetails(), JSON output – formatting errors occurred when the account was on the monthly plan option. Pay-as-you-go, formatted correctly.
- CHANGE: listInterestGroupings() – the 'bit' property was changed from type int to long, as values could exceed the int value capacity.
- FIX: listInterestGroupUpdate() – result returned was always false. This was only in Alpha code, not in the previous production release.

v1.2.5.2; Release Candidate – Fixes and Changes from Beta 1 source

- FIX: CodePlex # 5381, Interest group parsing breaks when comma is part of group name
- FIX: CodePlex # 5430, ListMergeVarAdd exception using Xml-Rpc
- CHANGE: Completed campaignEepUrlStats() output formatting based on new documentation.
- CHANGE: Based on #5430, I went through input code and applied liberal (perhaps excessive) use of the null-coalescing operator (x = property ?? "") to avoid program exception errors, particularly in xml-rpc.