Getting Started with the API

The YourMembership.com API is an XML-based web service that allows for easy communication between your community database and other external systems.

The Basics | back to top

The YourMembership.com API is grouped into a series of methods designed to fetch data from, and update data in, your online community database. In its simplest form, an API implementation consists of one or more calls to these methods chained together in a sequence to perform a specific task or set of tasks.
A typical API implementation may make anywhere from a few simple calls on a periodic basis to keep data synchronized between applications, to several thousand calls (one or more for each Member for example) depending on the goal of the implementation.

At its core, the API relies on XML-formatted messages being sent back and forth between the API and the calling application using standard HTTP requests and responses. The calling application is responsible for composing the XML message and HTTP request, sending the request to the API and awaiting a response. Response messages contain error code information and, if successful, may also contain response data in XML format for consumption.

Sending invalid or improperly formatted XML to the API will result in an error. Additionally, all XML messages sent to the API should use the UTF-8 character set.

XML message data must be sent to the API in the body of the request via secure HTTP (HTTPS) protocol using the POST method and standard form data encoding (application/xwww-form-urlencoded). The only exception is the MediaGallery.Upload method, which requires the multipart/form-data format, and requires the XML data to be passed along in a named parameter called XmlMessage.

Example API Request in XML format

```
<?xml version="1.0" encoding="utf-8" ?>

<YourMembership>
    <Version>1.61</Version>
    <ApiKey>3D638C5F-CCE2-4638-A2C1-355FA7BBC917</ApiKey>
    <CallID>003</CallID>
    <SessionID>A07C3BCC-0B39-4977-9E64-C00E918D572E</SessionID>
    <Call Method="Feeds.Get">
    </Call>
</YourMembership>
```

The example above illustrates a simple API request and method call in XML format. Note the parameters are packaged as individual elements and the Call element includes the name of the method to call (Feeds.Get). This method takes no parameters. The response to this call is displayed below.

Example API Request in XML format

```
<?xml version="1.0" encoding="utf-8" ?>

<YourMembership_Response>
    <ErrCode>0</ErrCode>
    <Feeds.Get>
        <Feed>
            <FeedID>NOW</FeedID>
            <Name>NOW!</Name>
            <LastUpdated>9/3/2009 4:56:38 PM</LastUpdated>
        </Feed>
        <Feed>
            <FeedID>Mobile</FeedID>
            <Name>Mobile NOW!</Name>
            <LastUpdated>9/8/2009 8:49:22 AM</LastUpdated>
        </Feed>
    </Feeds.Get>
</YourMembership_Response>
```

This response indicates a successful call and includes information on the two data items being returned from the original request (Feeds.Get). All API methods are called in a similar fashion, with an XML-formatted request resulting in an XML-formatted response which includes the ErrCode element and one root 'method results' element (in this case the Feeds. Get element), which may or may not contain child elements.

Language and Technology Requirements | back to top

Solutions built to interact with the API can be implemented in any modern programming or scripting language with available libraries for HTTP communication and XML parsing. Below are some of the more common languages that can be used.

- .NET Framework – (Recommended)
- Classic ASP / VBScript / Visual Basic
- Javascript / jQuery
- PHP, C/C++, Java and more

Regardless of what language and platform are used, programmers attempting to build solutions which interact with the API foundation should be familiar with the following concepts:

- HTTP communication basics
- XML formatting and parsing
- Date/Time Localization and UTC/GMT date formats
- .NET Framework version 3.5 or greater (when using the YMSDK)

Available Functionality as of Version 1.62* | back to top

The API exposes methods that allow applications to retrieve, modify and create entities in your community database. Note however, not all entities are accessible through the API. The list below outlines what functionality is available for which entities, as of the time of this document was written. *Please consult the online documentation for an up-to-date list of available methods.

Community-related functionality

- Get community RSS Feeds
- Get recent activity (including: newest member, latest post, latest comment, and latest photo)

Member and Non-Member specific functionality

- Search for member/non-member profiles
- Get member/non-member profile details
- Update member/non-member profile details\
- Create new member/non-member profiles

Member Communications

- Get inbox messages for a member
- Read a message for a member
- Get sent messages for a member
- Send a message for a member

Member Connections and Referrals

- Get a list of connection categories for a member
- Get a list of connections for a member
- Approve a connection request for a member

Member Content and Wall Management

- Get a list of media galleries
- Get a list of items in a media gallery for a member
- Get a single media gallery item
- Upload a new item to a member's media gallery
- Post to a member's wall
- Get information from a member's wall

Export Data

- Career Openings
- Financials
  - Donation details and invoice items
  - Dues details and invoices
  - Store orders and invoices

Member Communications

- Get inbox messages for a member
- Read a message for a member
- Get sent messages for a member
- Send a message for a member

Member Connections and Referrals

- Get a list of connection categories for a member
- Get a list of connections for a member
- Approve a connection request for a member

Method Call Details | back to top

API methods are called by submitting messages in XML format to the API endpoint URL (https://api.yourmembership.com). The XML message must include details about which method to call as well as the parameters and arguments required for that method. The XML message must be sent in the body of the request using the POST method.

Example API Request in XML format

```
POST https://api.yourmembership.com/ HTTP/1.1
Host: api.yourmembership.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 226

<YourMembership>
    <Version>1.62</Version>
    <ApiKey>3D638C5F-CCE2-4638-A2C1-355FA7BBC917</ApiKey>
    <CallID>001</CallID>
    <SaPasscode>**********</SaPasscode>
    <Call Method="Sa.People.Profile.Get">
        <ID>8D88D43A-B15B-4041-BEA0-89B05B2D9540</ID>
    </Call>
</YourMembership>
```

This example shows the format of a message calling the Sa.People.Profile.Get method. This particular method requires one argument, the ID of the profile to retrieve. The response is listed below.

Example API Request in XML format

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/xml; charset=utf-8
Content-Length: 218
Connection: Keep-alive

<?xml version="1.0" encoding="utf-8" ?>

<People.Profile.Get>
    <MemberTypeName>Alumnus</MemberTypeName>
    <LastUpdated>2010-08-31 17:01:16</LastUpdated>
    <EmailAddr>test@yourmembership.com</EmailAddr>
    <Name>Elizabeth Allen</Name>
    <Birthdate>1980-09-15</Birthdate>
    <AnniversaryDate>1999-01-01</AnniversaryDate>
    <Employer>University of South Carolina</Employer>
    <Title>Graduate Student</Title>
    <Profession>Banking</Profession>
    <PrimaryAddressType>Primary</PrimaryAddressType>
    <HomeAddrLine1>1910 Anyplace Ave</HomeAddrLine1>
    <HomeAddrLine2>Suite 9999</HomeAddrLine2>
    <HomeCity>Sweetwater</HomeCity>
    <HomeLocation>Alabama</HomeLocation>
    <HomePostalCode>29208</HomePostalCode>
    ...
</People.Profile.Get>
```

The response of the method call includes error code details and method results, if any. The method results in this example contain the profile information for the selected member.

API Sessions | back to top

The API includes Session management features which can be helpful for implementations that need to act on behalf of, and impersonate, authenticated users. **Some API implementations may not need Sessions at all**, but for other implementations, Authenticated Sessions in particular, are crucial.

Methods that Don't Need an Active Session

Most Sa.* methods don't require an active session to be called. These methods are typically administrative in nature and don't require the context of a logged in user to be run.

Session Lifetime

API sessions are not permanent and will timeout after a period of inactivity (no method calls using a given SessionID). To ensure that an API session doesn't time out, developers may wish to implement periodic calls to Session.Ping, at a frequency no less than every 20 minutes.

Anonymous Sessions

A standard, anonymous API Session is initialized by calling Session.Create. This method returns a SessionID token which is then used for making subsequent method calls. This type of Session is useful and perfectly valid for most API method calls.

However, in some situations it may be important for the API to know which Member context is making the request. These methods require an Authenticated Session.

Authenticated Sessions

Sessions can be authenticated as well. To authenticate an existing Session, call the **Auth.Authenticate** method. Upon successful authentication, this method will bind the current Session with the authenticated user. Successive calls using the same authenticated SessionID token can be made and the API will always know which user is making the request.

Required Parameters | back to top

Sessions can be authenticated as well. To authenticate an existing Session, call the **Auth.Authenticate** method. Upon successful authentication, this method will bind the current Session with the authenticated user. Successive calls using the same authenticated SessionID token can be made and the API will always know which user is making the request.

- For standard methods, use your Public API Key. This key is used to identify your organization and tells the API which constituent database to use.
- For all Sa.* methods, use your Private API Key. Remember that you will also need to provide your SaPasscode when making calls to Sa.* methods.

Version

All method calls require the Version parameter to help identify the features that are available for use. In most cases, developers should configure their application to use the latest version of the API, but may also want to code in configuration options that will allow additional features to be implemented when new versions of the API are released. The value for the Version field must match either the current version of the API (e.g. 1.62) or a previously released version of the API. Setting this field to something like 1.* is not allowed.

CallID

All method calls required the CallID parameter. This required parameter is used internally to identify individual API method calls. Each call in a given Session must have a unique CallID. A common pattern is to use an auto-incrementing numeric value for each call made, although any combination of letters and numbers is allowed, up to 25 characters.

SessionID

A valid SessionID is required for nearly all calls to the API. A SessionID is obtained by calling Session.Create. This method initializes a new Session and returns the unique ID for that Session. API Sessions have a limited lifetime and will expire after 20 minutes of inactivity. To keep an active Session alive, make a call using the SessionID before the timeout period has elapsed or call the Session.Ping utility method.

SaPasscode

The SaPasscode parameter is required for all Sa.* methods (Sa.People.Profile.Get for example). Each client has a separate, unique SaPasscode. Your SaPasscode can be obtained from Customer Service.

YMSDK – The YourMembership.com Software Development Kit | back to top

The YMSDK was developed as a companion library for use in ASP.NET web sites and other .NET framework applications. Developers building API-driven solutions using the .NET Framework may want to consider utilizing this feature-rich library as it may significantly reduce development efforts.

The YMSDK consists of helper utilities and wrapper functions that makes interacting with the YM API as easy as making normal .NET method calls. Each API method has been exposed and support for future API versions and methods has been included as well.

When using the YMSDK, all the developer needs to worry about is the internal logic regarding which calls to make and when; and how data is handled between calls. The library handles just about all of the other details.

Frequently Asked Questions | back to top

Q: How do I make an API call?
A: By submitting an XML-formatted message in the body of an HTTP POST request to the main API endpoint URL (https://api.yourmembership.com/).

Q: What protocol and request method should I use?
A: API calls must be submitted via secure HTTP (HTTPS) protocol using the POST method.

Q: Should I use a specific content type and character set when making method calls?
A: Yes. For most method calls, the content type (Content-Type) header must be set to application/x-www-form-urlencoded and the recommended character set is UTF-8. However, for specific methods that require a file or files to be sent along with the request, the content type must be multipart/form-data to enable transmission of the uploaded file. Also note that in this situation the XML data should be sent over in a form field named XmlMessage.

Q: How do I compose XML messages to send to the API and parse the XML responses?
A: Libraries which assist the developer in creating XML documents are available in most common programming languages. It is recommended to use a robust library for XML composition and parsing, as most feature built-in validation, encoding and escaping of content which can help reduce common XML-related errors. An alternative is to construct "XML strings" manually. In this scenario, however, escaping, encoding and validation of the XML are left up to the developer.

Q: What is the difference between an empty tag and a tag that is not included with the request?
A: When calling an update method, the system will ignore any field that was not included in the request, thus leaving the existing database value as is. A tag that is included, but contains no value, instructs the system to update the corresponding database field with an empty value.

Q: What fields are included in the exports when using the Sa.Export namespace?
A: The Sa.Export.Examples (.zip) file contains sample exports created using each of the methods in the Sa.Export namespace.

Q: What is the difference between Sa.* methods and normal methods?
A: Sa.* methods are methods that are reserved for System Administrators (SA). These methods typically have the capability of operating on records that are not specific to any one user. As a result and for security purposes, these methods require the SaPasscode parameter. They do not however, require an active session or a unique CallID.

Q: I am receiving the error 201 - [SessionID] invalid
A: The SessionID parameter, which is required for most API calls, is either missing or invalid. Make sure you have called Session.Create and use the returned SessionID value for subsequent calls.

Q: I am receiving the error 202 - [SessionID] has expired
A: The SessionID parameter given is for an API Session that has expired (timed out). An API Session begins when Session.Create is first called and expires after 20 minutes of inactivity. To keep an active Session alive, you must make a call to the API (any call that requires a SessionID parameter) at least once every twenty (20) minutes. See the Session. Ping method for more information.

Q: I am receiving the error 401 – Invalid Method Call
A: The API cannot identify the method to call. The name of the method to call is case-sensitive. Also verify that you are passing in the correct version parameter and that the method you are trying to call is available for that version. Some methods are retired and are no longer available with later versions of the API.

Q: I am receiving the error 402 - Method requires an active session
A: Make sure you are passing a valid SessionID to the method. Obtain a new SessionID by calling Session.Create.

Q: I am receiving the error 404 - Method call failed. One or more arguments is missing or invalid
A: Verify that you are passing in the correct method arguments and that the values are valid. This error can happen when the method is expecting an integer value for the ID argument for example, but encounters a non-numeric string value instead. This error may also be generated when an unexpected internal error is encountered.

Q: I am receiving the error 406 - Method could not uniquely identify a record on which to operate
A: The value given for the ID argument (or similar) could not be located in the system. Double-check the value and try again.

Q: I am receiving the error 901 - Invalid HTTP request
A: You have submitted a method call to the API system using a method other than HTTP POST. Verify that you are posting the XML payload data properly and have all required HTTP headers set properly and try again.

Q: I am receiving the error 902 - XML packet is malformed or otherwise un-readable
A: The XML message you are sending is either missing or invalid. This could mean that invalid characters are present in the XML document. Be sure to properly escape and encode XML entities and only send valid XML messages to the API.

Appendix I: Error Codes | back to top

CODE	DESCRIPTION
0	No error
101	[Version] is missing, invalid or unrecognized
102	[ApiKey] is missing or invalid
103	[SaPassCode] is invalid
201	[SessionID] invalid
202	[SessionID] has expired
301	[CallID] is missing or invalid
302	[CallID] has already been used during this session
401	Invalid method call
402	Method requires an active session
403	Method requires authentication
404	Method call failed. One or more arguments is missing or invalid
405	Method requires SA authority. Proper credentials must be supplied
406	Method could not uniquely identify a record on which to operate
901	Invalid HTTP request
902	XML packet is malformed or otherwise unreadable
998	The API service is currently unavailable
999	An unknown error has occured

Appendix II: Time Zone Considerations | back to top

The YourMembership.com API servers are localized to the U.S. Eastern time zone (EST/ EDT). All dates and times are localized to this time zone.

Customers outside of the Eastern time zone will need to take the differences into account when dealing with date and time values in the API. To convert a DateTime variable from your locality to Eastern Time, use the Convert.ToEasternTime API Method.

Pass in the current date and time for your local time zone (in YYYY-MM-DD hh:mm:ss format), along with your current GMT offset and the method will return the current date/time where the server resides, along with the current GMT offset of the server (GMT-5 by default, or GMT-4 when daylight savings is in effect).

Consider the following example:

Customer A has offices in California, where their offset is GMT-8. They want to retrieve a list of any members who registered or updated their profiles in the last 24 hours.

1. The current date/time value in their local time zone: 2010-01-21 14:30:01
2. Subtract 24 hours (1 day) from that date: 2010-01-20 14:30:01
3. Send that value to Convert.ToEasternTime, which returns an updated date/time value of: 2010-01-20 17:30:01
4. This new value is then sent over as a parameter to Sa.People.All.GetIDs which re turns a list of IDs for profiles that were created or updated AFTER the date/time value given, and also returns the current date/time on the server.
5. The returned date/time value can be saved and sent over directly the next time they want to get fresh data
6. This process of sending over a starting date in Eastern Time, and storing the date that is returned can be repeated at will, with only one call to Convert.ToEasternTime being necessary (for the initial conversion).