Yahoo! Mail Development Platform Documentation

Last updated 2/23/09

YAHOO! MAIL

Table of Contents

YAHOO! MAIL

What is the Yahoo! Mail Development Platform?

The vision for Yahoo! Mail is a Smarter Inbox experience that allows users to communicate better and get more things done.

The Yahoo! Mail team has built an application platform that will allow you as a developer to create applications that can run in the All New Yahoo! Mail. Using the Yahoo! Mail Development Platform (YMDP), you will be able to add your own rich functionality to Y!Mail and distribute your services to our huge user base through your applications.

Yahoo! Mail is already a top starting point for consumers on the Web, and the Mail development platform brings new possibilities and a more productive experience to hundreds of millions of e-mail users around the world, helping unite them with their favorite services from the top online brands.

How it works

As a developer, you will have access to our Developer Tool where you will be able to create and manage your application configuration, code, images, assets and testers. Yahoo! also hosts your application for you.

Applications are written in HTML and will have access to our Javascript APIs. With these APIs, your application will be able to do the following (and more):

- trigger and prepopulate a new compose email
- access a plethora of Yahoo! data via YQL
- act on a message that is dropped on your application
- authenticate a user
- create a Yahoo! Calendar event

All published applications appear in the Application Gallery which is accessible via the “Add” link in the left pane of the Mail UI next to “Applications”. A user can add or remove your application via the Application Gallery. Once a user has added your application, it will appear in the left pane under “Applications”.

You can control how your application is rendered: in a dialog box or a tab. A user can trigger your application by clicking on your application in the left pane or dragging and dropping an email message onto your application. If a user drags and drops an email onto your application, you will have access to the contents of that email message.

YAHOO! MAIL

Sounds great. What are the restrictions?

In an effort to protect our users' data and keep the application quality levels high, we have Application Technical Requirements (see below section). The main restrictions are as follows:

- We do not allow you to publish apps. The Mail team publishes apps to be seen by users
- All Javascript. Flash and CSS must either be inlined or uploaded via our Asset Uploader
- You may not embed iframes pointing to non-Yahoo! servers

Please refer to the Application Technical Requirements section below for further details.

YAHOO! MAIL

Becoming a YMDP developer

Now that you know the background on our platform, here's how you can get started building applications:

1. Send the following information to openmail-devsupport@yahoo-inc.com:
    a. A tester Yahoo! ID - This is the YID that you will be using to test your app. It must have an activated email address associated with it. You may assign other tester IDs via the Developer Tool, but those YIDs will only be able to see your app, not edit. We will also add this Yahoo! ID to the Yahoo! Group ymdp_developers.
    b. Bouncer information – To have access to the Developer Tool, you will need a Bouncer account. We will need the following information to set up your account:
        i. First and Last Name
        ii. Company email address
        iii. Company name

2. Once you have received confirmation that your Yahoo! ID and Bouncer account have been set up, go to http://om1.mail.vip.mud.yahoo.com/openmail/dev_activate.php to activate your developer account. You may be prompted to change your password for security purposes.

3. Please read through the latest version of our documentation (all included below unless otherwise stated):

    a. Our API references: http://developer.yahoo.com/mailapplications/guide/)
    b. Dropped Message Object Description – information available when the user drags and drops and email message onto an app
    c. Application Configuration Description
    d. Sample Application Code
    e. Application Design Guidelines (separate from this doc)
    f. App Technical Requirements - includes any technical restrictions for your app
    g. Developer FAQs

4. To access the Developer Tool, go to: http://om1.mail.vip.mud.yahoo.com/openmail/devtool.php
Please note that Yahoo! hosts the application content, so it is not necessary to configure anything outside of the developer tool, such as other servers, etc.

The primary components of an application are:

a. **Application configuration** – This is an XML specification of various meta data about the application, including what sorts of data it is interested in subscribing to (e.g. mail messages), what to do when such data is received (e.g. open a view into a tab inside the Y! Mail UI), etc.
b. **Views** - Views are simply HTML. They are rendered inside the Mail UI as specified by the application configuration, and they have access to our JavaScript apis (see the API Reference doc for more API information)
c. **Authorization** – This is where you can set up your profile for authenticating a user. We currently support OAuth (2 and 3 legged) and BBAuth. (please see the AuthService section of the API Reference doc)
d. **Images** – You can upload images that are used inside the Y! Mail UI, such as to display an icon image next to the application name, or to show larger images inside the application gallery.
e. **Assets** – You upload and store up to 5MB of files. This is a great place to put your JavaScript and CSS files (since we don't allow you to externally reference them). We provide an asset url so that you may access your assets in your app code.

5. To test your app, simply go to http://mail.yahoo.com (make sure you are using the All New Yahoo! Mail), click on the Add link next to Applications in the left pane and add your application from the Gallery.

6. When you are ready to publish your app to Yahoo! Mail users, please email openmail-devsupport@yahoo-inc.com and we will publish your app.

7. For further questions, please contact us via:
   a. Yahoo! Group ymdp_developers (http://tech.groups.yahoo.com/group/ymdp_developers/). Other YMDP developers are in this group, but our support team is also monitoring and answering questions on this Y!Group.
   b. Email openmail-devsupport@yahoo-inc.com
   c. Email Stephanie Shum at sshum@yahoo-inc.com or Dan Avery at davery@yahoo-inc.com

YAHOO! MAIL

Dropped Message Object Description

Dropped message object is a JavaScript object passed along with openmail.getParameters() callback function. It contains information of the email message dropped into an Openmail application.

Types of Dropped Message Objects

Depending on what you specify in your mail_message application config, you may get different types of dropped message objects. See application config documentation.

Format 'display'

If you don't specify this parameter or set it to 'display', then you get the displayable parts of a message, with inline shortcuts annotation into the message text, if you have shortcuts turned on in your preference.

Format 'full'

If you specify 'full' you get the displayable as well as nondisplayable parts of the message in a multipart mime format (json). Any annotation appears under the part that represents the text/plain part of the message as a child node. This part is unescaped and unannotated. If the message was formed in richtext mode, then there is another part of type text/html (displayable part of the message). Attachments appear as urls in other parts with type information indicating what type of attachment they are.

Viewing 'display' format Dropped Message Object

The easiest way to learn about the object is to use "View Message JSON" application. The object can be browsed in similar fashion to using firebug object viewer. Just install View Message JSON application from the Gallery and drop a message onto it.

Field Description

Top-level

Property	Type	Description
from	Address	An Address object containing the address of the sender
replyto	Address[]	Array of Address object corresponding to reply-to, to and cc email fields. Empty array if the value is unspecified

to	Address[]	
cc	Address[]	
subject	string	Email subject field
date	number	Received date in local time zone
body	Message Body	Message body information
attachment	Attachment[]	Array of attachments, empty array if there's no attachment

Address

Property	Type	Description
name	string	Name associated with the email address
email	string	Email address

Message Body

Property	Type	Description
content	string	Message body in HTML

Attachment - This object contains information about each attachment

Property	Type	Description
type	string	MIME media type
subtype	string	MIME media subtype
url	string	attachment URL

Dropped Message Object Sample

```
{
    from:
        {
            name: "Openmail Announcement",
            email: "foo@yahoo.com"
        },
    replyto: [],            // No reply-to address
    to: [                      // Multiple recipient
        {
            name: "Openmail Developers",
            email: "openmail-list@yahoo.com"
        },
        {
            name: "",               // Name is not specified
            email: "someone@yahoo.com"
        }
    ]
    cc: [],                    // No cc address
    subject: "Welcome!",
    date: 1219366011420,
    body: {
        content: "html message body"
    },
    attachment: [
        {
            type: "image",
            subtype: "png",
            url: "http://somedomain.com/file-1.png"
        },
```

```
        {
            type: "image",
            subtype: "png",
            url: "http://somedomain.com/file-2.png"
        }
    ]
}
```

Viewing 'full' format Dropped Message Object

The easiest way to learn about the object is to use "View getMessage JSON" application. The object can be browsed in similar fashion to using firebug object viewer. Just install View Message JSON application from the Gallery and drop a message onto it.

Field Description

Top-level

Property	Type	Description
from	Address	An Address object containing the address of the sender
replyto	Address[]	Array of Address object corresponding to reply-to, to and cc email fields. Empty array if the value is unspecified
to	Address[]	
cc	Address[]	
subject	string	Email subject field
date	number	Received date in local time zone
part	Mime part Array	Message content information including header, body,annotation, attachments

Address

Property	Type	Description
name	string	Name associated with the email address
email	string	Email address
Part		
Property	Type	Description
partId	string	Part ID
type	string	type e.g 'text','application','multipart'
subtype	string	specific type e.g 'plain','html','vnd.ms-powerpoint','alernative'
disposition	string	'attachment' if attachment part
text	string	for parts that hold message content
encoding	string	encoding of part e.g '7bit'
typeParams	string	
size	number	size of the part text

annotation	object	associated with the part of type 'text/plain'

Annotation

Property	Type	Description
hasSensitiveText	boolean	if content has sensitive words
entity	array of annotation entities	array containing the various annotation items

Annotation entity

Property	Type	Description
text	string	The text of the annotation item itself
type	array of annotation types	array of the various shortcuts types associated with this item
startChar	number	The offset position in the document where the entity starts
endChar	number	The offset position in the document where the entity ends
weight	float	A normalized weight score between 0 and 1 that represents the overall importance of an entity

Annotation type

Property	Type	Description
text	string	the original type string returned from shortcuts. This is an internal string and can change without notice. Recommended not to use
category	string	indicates item category - 'instance'(particular item in class e.g diabetes),'class'(e.g disease), 'concept','abstract'(something of which an instance is probably not a child),'tag'(Subjective classifications)
type	string	the type of entity the annotation is . Valid values are URL, phone_number, email_address, date_only, date_time, day_of_week, day_of_week_time, month_only, month_year, time_only, place, person, organization, event, animal/fictional, other, technology, news, celebrity, finance/glossary_mail
subtype	string	any extra information associated with the type

Attachment - This object contains information about each attachment

Property	Type	Description
type	string	MIME media type
subtype	string	MIME media subtype
url	string	attachment URL

YAHOO! MAIL

Dropped Message Object Sample

```
{
    from:
        {
            name: "Openmail Announcement",
            email: "foo@yahoo.com"
        },
    replyto: [],            // No reply-to address
    to: [                      // Multiple recipient
        {
            name: "Openmail Developers",
            email: "openmail-list@yahoo.com"
        },
        {
            name: "",               // Name is not specified
            email: "someone@yahoo.com"
        }
    ]
    cc: [],                    // No cc address
    subject: "Welcome!",
    date: 1219366011420,
    part
        [0] : {<some data>},
        [1] : {<some data>},
        [2] : {<some data>},
        [3] : {
                partId : "1.1",
                subtype : "plain",
                type : "text",
                typeParams : "charset=us-ascii",
                disposition : "",
                encoding: "7bit",
                filename : "",
                size : 48,
                text : "fyi ... Here's some info on the lowfade release",
                annotation : {"hasSensitiveText":true,
                                    entity
                                        [0] : {
                                                text: "Thursday Nov 11"
                                                type
                                                   [0] : {
                                                                text :
"shortcuts:/us/instance/identifier/date_only",
                                                                category :
"instance",
                                                                type :
"date_only",
                                                                 subtype : "",
                                                           }
                                                startChar :359,
                                                endChar :373,
                                                weight:1
                                            }
                                        [1] : {
                                                text: "http://www.flickr.com"
                                                type
                                                   [0] : {
                                                                text :
"shortcuts:/us/instance/identifier/URL",
                                                                category :
"instance",
                                                                type : "URL",
                                                                 subtype : "",
                                                           }
                                                startChar :241,
                                                endChar :261,
                                                weight:1
                                            }
            }
      [4] :
```

```
            {
              partId : "2",
              subtype : "vnd.ms-poerpoint",
              type : "application",
              disposition : "attachment",
              encoding: "base64",
              filename : "low Fade Planning.ppt",
              size : 92134,
             }


      }
}
```

YAHOO! MAIL

Application Configuration Description

The application configuration is an XML specification of various meta data about the application, including what sorts of data it is interested in subscribing to (e.g. mail messages), what to do when such data is received (e.g. open a view into a tab inside the Y! Mail UI), etc.

Here's what's available to put in your app config, following the structure below.

- The app config is contained in a root <openmail_app_config> element.

- The current minimum valid application config is
  - <?xml version="1.0"?>
  - <openmail_app_config version="0"/>

- The <openmail_app_config> must contain an integer "version" attribute, developer-defined.

- That element may contain 0 or more of three types of elements:
  - <data> This element specifies what data inputs the application can react to. Currently there is only one available data source:
    - <mail_message> An individual mail message, when dragged-and-dropped on the application icon
  - <events> More general events the application can react to.
    - <load> Initial Candygram launch and application reload (which happens for all installed applications when any application is installed or removed from the gallery).
    - <click> override the default click behavior, which is to launch a full view in a tab.
    - <uninstall> Receive notification when the user has chosen to uninstall the app.

- The <mail_message> element may contain the <format> element to specify the format of the mail message provided. <format> is one of:
  - "display" -- mime part of the message that contains the displayable message, this is the default
  - "full" -- the displayable as well as non-display parts of the message and attachments as urls except for plain text attachments that are embedded as a part

- Each data or event element should contain one <action> element. It should contain one of the following elements specifying the action to perform.
  - <launch> Renders an iframe for your app. Used in virtually all apps.
    - <view> string -- name of the view to display

        - <context> string -- made available to you via getParameters() call
        - <target_zone> string -- where to render your frame. One of "tab", "dialog", or "hidden".
            - For <uninstall> events, "uninstall" or "hidden" is allowed here. "uninstall" will open a dialog with Candygram-owned OK & Cancel buttons.
        - <ttl> integer -- automatically close after this many seconds. This works only for targetzone "hidden".
        - <width> integer -- width in pixels (only for dialog and hidden iframes)
        - <height> integer -- height in pixels (only for dialog and hidden iframes)
        - <title> string -- displayed in titlebar
    - <mailto> compose a mail message. Rarely used because parameters currently have to be hard-coded in the child elements:
        - <to>
        - <cc>
        - <bcc>
        - <subject>
        - <body>
    - <mail_search> perform a vespa search. Rarely used because the query is hard coded in the child element:
        - <query_string> a Vespa query string

Sample Applications

Here are some sample applications that make use of the different configurations and APIs that are available to you:

Hello World

This app configuration is about as simple as it gets. It's XML tells Yahoo! Mail to launch the app's "main" view in a tab once the app's icon is clicked.

```
<openmail_app_config version="2">
   <data/>
   <events>
      <click>
         <action>
            <launch>
               <view>main</view>
               <target_zone>tab</target_zone>
            </launch>
         </action>
      </click>
   </events>
</openmail_app_config>
```

The app has one view and it's called main, to match the value in the config xml above. It demonstrates how to use one of the API calls, Mail.compose. When the "Compose" button is clicked, the form invokes Mail.compose, which will open the mail composer with some fields prepopulated from the form.

Like many of the OM API calls, Mail.compose takes a first argument that is a javascript object literal. This is handy because it allows the programmer to pass arguments by keyword instead of worrying too much about ordering. In this case, it has three fields that map to parts of an email message. See the API doc for information on what other fields can be used with this call.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<!-- Required scripts for using the Yahoo Mail Developer Platform API -->
<script src="http://yui.yahooapis.com/2.6.0/build/yahoo-dom-event/yahoo-dom-event.js"></script>
<script src="http://yui.yahooapis.com/2.6.0/build/json/json-min.js"></script>
<script src="http://mail.yimg.com/a/lib/om/crossframe/1.0.14/crossframe-min.js"></script>
<script src="http://mail.yimg.com/a/lib/om/om_api_public/1.0.1/om_api_public.js"></script>
</head>

<body>
<textarea id="body" cols="80" rows="20">Enter some text and press the compose button.</textarea>
<p/>
<button onclick="demoCompose()">compose</button>
<script>
function demoCompose(){
   var bodyTxt = document.getElementById('body').value;
   openmail.Mail.compose({ to: 'my friend',
                     subject: 'hello world',
                     body: bodyTxt });
}
```

```
</script>
</body>
</html>
```

Memories

Much of the OpenMail API utilizes callbacks. This is to allow operations to take place in the background and asynchronously notify the caller upon completion. Application.getData and Application.setData are typical examples of such functions.

These function allow the application to save and load key/value pairs specific to the logged-on mail user. When you call them, you pass in a callback, also known as a handler function. The handler will be called back with a response containing the requested data or error information.

This application demonstrates using the setData and getData API to remember a user's name and the date when it first 'met' the user.

The config is nothing terribly interesting. It says to launch the "main" view in a dialog when the user clicks on the application icon.

```
<openmail_app_config version="2">
   <data/>
   <events>
      <click>
         <action>
            <launch>
               <view>main</view>
               <target_zone>dialog</target_zone>
               <title>Memories Sample App</title>
               <height>400</height>
            </launch>
         </action>
      </click>
   </events>
</openmail_app_config>
```

This apps one and only view is called "main" to match the value in the config xml above. The main view runs code on launch to query by key for the user's name and time they first met. The keys and values are persisted in per-user-per-app storage, which means that the data is specific to this user and this application. This query is initiated with the getData call.

The callback passed to getData is onGetData, so it will be invoked once the server has responded with the query results. Assuming a non-error response, it will contain the requested keys and their values. If they haven't been set, main.html prompts the user to enter their name and uses the setData call to save the user's name and the current time.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
<head>
<title>Memories</title>
<!-- Required scripts for using the Yahoo Mail Developer Platform API -->
<script src="http://yui.yahooapis.com/2.6.0/build/yahoo-dom-event/yahoo-dom-
event.js"></script>
<script src="http://yui.yahooapis.com/2.6.0/build/json/json-min.js"></script>
<script src="http://mail.yimg.com/a/lib/om/crossframe/1.0.15/crossframe-min.js"></script>
<script
src="http://mail.yimg.com/a/lib/om/om_api_public/1.0.1/om_api_public.js"></script>
</head>

<body>
<!-- We'll talk to the user by appending text to this div -->
<div id="message">I think I might have met you before, let me see. . .<p/></div>

<!-- By default, this div isn't displayed.  If we've never met the user, display it -->
<!-- so that they can submit their name -->
<div id="meet" style="display:none">
<p/>What's your name?
<input id="name"><button onclick="onJustMet()">Submit</button>
</div>

<!--Another div that isn't displayed by default.  We'll display it if we've met the -->
<!--user before, to give them the option of being forgotten. -->
<p/>
<button id="forget" style="display:none" onclick="onForget()">Forget About Me</button>

<script>
var messageDiv = document.getElementById('message');
openmail.Application.getData({keys:['name','timeWeFirstMet']}, onGotData);

//Async getData callback invoked once it has finished our query.
function onGotData(response){
    if(response.error){
        messageDiv.innerHTML += 'I\'m having technical difficulties with my memory.<p/>'
+
                                'Try again later.<p/>';
        return;
    }

    var timeWeFirstMet = response.data.timeWeFirstMet;
    if(!timeWeFirstMet){
        messageDiv.innerHTML += 'Either I\'ve never met you before or I\'ve forgotten.  '
+
                                'about you.  You seem pretty cool, though.';
        document.getElementById('meet').style.display = "";
        return;
    }


    messageDiv.innerHTML += '<p/> I do remember you, ' + response.data.name + '!';

    document.getElementById('forget').style.display = "";

    var now = new Date().getTime();
    var MS_PER_DAY = 3600000 * 24;
    var daysKnown = Math.floor((now - timeWeFirstMet) / MS_PER_DAY);

    if(daysKnown < 1){
        messageDiv.innerHTML += "<p/>We met pretty recently.";

        return;
    }
    messageDiv.innerHTML += '<p/>I met you ' + daysKnown + ' days ago.'
}

//Handler for when the user submits their name.
function onJustMet(){
   var who = document.getElementById('name').value;
   var now = new Date().getTime();

   openmail.Application.setData({keys : {name: who, timeWeFirstMet : now}});
```

```
   messageDiv.innerHTML += '<p\>I\'ll try to remember you next time we meet, '
                           + who + '.';
   //Hide the meet-and-greet dialog
   document.getElementById('meet').style.display="none";
}

//Handler for when the user clicks the "Forget About Me" button.
function onForget(){
   openmail.Application.setData({keys : {timeWeFirstMet : ""}});
   //Hide the forget button
   document.getElementById('forget').style.display="none";
   messageDiv.innerHTML += '<p\>It was nice knowing you.';
}

</script>
</body>
</html>
```

Note that getData does not consider requested keys that have never been set to be an error. It will respond to such requests with the requested keys' values set to the empty string.

Drag and Drop

This app demonstrates how a view can use the getParameters call to:

- distinguish between different reasons for its launch
- access an email message after a drag-and-drop launch.

The config says:

- When an email is drag-and-dropped onto the app, launch the 'full' view in a dialog box.
- Use non-default dimensions and a title for the dialog.
- When the app is launched by a user clicking on the app icon, launch the 'full' view in a tab.

Note that there is nothing special about the name of the view. Other samples with only one view call it 'main'. In this app, we launch the same view for both launch reasons, but we could have easily added a second view and configured a different view for each launch reason. We choose to have the same view programatically deal with both situations instead.

Note also that an bug currently requires you to have the context fields here, even though they are optional and not truly used in this example.

```
<?xml version="1.0"?>
<openmail_app_config version="2">
  <data>
    <mail_message>
      <action>
        <launch>
          <view>full</view>
          <target_zone>dialog</target_zone>
          <title>Drag and Drop Sample App</title>
          <height>200</height>
```

```
          <width>600</width>
          <context/>
        </launch>
      </action>
    </mail_message>
  </data>
  <events>
    <click>
      <action>
        <launch>
          <view>full</view>
          <target_zone>tab</target_zone>
          <context/>
        </launch>
      </action>
    </click>
  </events>
</openmail_app_config>
```

The "full" view immediately calls getParameters so that it can determine the launch reason. It specifies an anonymous inline function for the callback to getParameters instead of using a previously-defined function. When that handler is invoked with the response, it checks to see why the app was launched. If the app was launched due to an email drag-and-drop ('data/message'), it probes the email and dumps out some basic information.

More complex walking of the email's data is covered in the View JSON app, not covered in this sample guide.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script src="http://yui.yahooapis.com/2.6.0/build/yahoo-dom-event/yahoo-dom-
event.js"></script>
<script src="http://yui.yahooapis.com/2.6.0/build/json/json-min.js"></script>
<script src="http://mail.yimg.com/a/lib/om/crossframe/1.0.14/crossframe-min.js"></script>
<script
src="http://mail.yimg.com/a/lib/om/om_api_public/1.0.1/om_api_public.js"></script>
</head>

<body>
<div id="status">Checking launch reason. . .</div>

<script>
openmail.Application.getParameters(function(response){
    var statusDiv = document.getElementById('status');

    if(response.error){
        statusDiv.innerHTML = 'getParameters failed: ' + response.error;
        return;
    }

    if(response.reason != 'data/message'){
        statusDiv.innerHTML = 'This app is more interesting if you launch it by ' +
                              'drag and dropping an email message onto it.';
        return;
    }

    //Assuming there is only one author listed in the email's "From:" field.
    statusDiv.innerHTML = 'The messages you dropped was from:<br> ' +
response.data.from[0].email
                          + '<p/>The subject was:<br>' + response.data.subject;
});

</script>
```

```
</body>
</html>
```

Saved Searches

Builds an app that lets you save searches. Uses the getParameters, setData, getData, search, and setFolderConfig calls.

Here's the config:

```
<openmail_app_config version="2">
   <data/>
   <events>
      <load>
         <action>
            <launch>
               <view>full</view>
               <target_zone>hidden</target_zone>
               <context>load</context>
               <ttl>0</ttl>
            </launch>
         </action>
      </load>
   </events>
</openmail_app_config>
```

Here's the "full" view:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script src="http://yui.yahooapis.com/2.5.1/build/yahoo-dom-event/yahoo-dom-
event.js"></script>
<script src="http://yui.yahooapis.com/2.5.1/build/json/json-min.js"></script>
<script src="http://mail.yimg.com/a/lib/om/crossframe/1.0.13/crossframe-min.js"></script>
<script
src="http://mail.yimg.com/a/lib/om/om_api_public/1.0.1/om_api_public.js"></script>
<style>
body, html, iframe {
  margin:0px;
  padding:0px;
  border:0px;
}
body {
  overflow:hidden;
}
#searchList .item {
  background: #ddf;
  border: 1px solid #aaa;
  margin:1em;
  padding:0.3em;
}
</style>
</head>
<body>
<div id="searchList">
Loading...
</div>
<script>

var state = [
    {label:"Sample 1", query:"test"},
    {label:"Sample 2", query:"openmail"}
```

```
];

openmail.Application.getParameters(function(args){
  openmail.Application.getData( { keys: ["state"] }, function(result) {
    var stateKey = result.data["state"];
    state = stateKey||state;
    if (typeof state === "string") {
        state = YAHOO.lang.JSON.parse(state);
    }
    if (args.context == "load") {
      loadHandler();
    } else {
      render();
    }
  });
});

function updateFolders(){
    var obj = {
        subFolders:[]
    }
    for (var i=0;i<state.length;i++) {
        var s = state[i];
        s.tooltip = s.query;
        obj.subFolders.push(s);
    }
    obj.count = state.length;
    openmail.Application.setFolderConfig(obj);
}

function doSearch(data)
{
   var stateKey = data.data["state"];
   state = stateKey||state;
   if (typeof state === "string") {
        state = YAHOO.lang.JSON.parse(state);
    }
   openmail.Mail.search( {query: state[subfolderIndex].query} );
}

function omSubfolderClick(args)
{
    subfolderIndex = args.index;
    openmail.Application.getData({keys:["state"]}, doSearch);
}

function loadHandler() {
  updateFolders();
  openmail.Application.setListener( {event: "subfolder.click"}, omSubfolderClick);
}

function render() {
  var html = "<h1>Saved Searches</h1>";
  for (var i=0;i<state.length;i++) {
    html += generateRow(state[i], i);
  }
  html += "<button id='create'>New Saved Search</button>";
  document.getElementById("searchList").innerHTML=html;
  var btns = document.getElementsByTagName("button");
  for (var i=0;i<btns.length;i++) {
    btns[i].onclick = function(btn){ return function(){buttonClick(btn)}; }(btns[i]);
  }
}

function htmlencode(s) {
  return
s.split("&").join("&").split("<").join("<").split(">").join(">").split('"').join("\"");
}

function generateRow(search, index) {
  var a = [
    "<div class='item'>",
```

```
    "<div id='display_",index,"'><span
id='label_",index,"'><b>",htmlencode(search.label), "</b></span><br />",
    "<span id='query_'",index,"'>Query: <tt>", htmlencode(search.query), "</tt></span><br
/>",
    "<button id='run_",index,"'>Run</button> - ",
    "<button id='edit_",index,"'>Edit</button> - ",
    "<button id='delete_",index,"'>Delete</button>",
    "</div>",
    "<div style='display:none' id='input_",index,"'>",
    "<p>Name: <input type=text class='labelinput' id='labelinput_",index,"' /></p>",
    "<p>Query: <input type=text class='queryinput' id='queryinput_",index,"' /></p>",
    "<button id='save_",index,"'>Save</button> - ",
    "<button id='cancel_",index,"'>Cancel</button>",
    "</div>",
    "</div>" ];
  return a.join('');
}

var $ = function(e) { return document.getElementById(e); };

function buttonClick(btn) {
  var a = btn.id.split("_");
  var action=a[0], index=a[1];
  var stateStr=null;
  switch (action) {
    case "run":
      openmail.Mail.search( {query:state[index].query} );
      break;
    case "edit":
      $("display_"+index).style.display="none";
      $("input_"+index).style.display="";
      $("labelinput_"+index).value = state[index].label;
      $("queryinput_"+index).value = state[index].query;
      break;
    case "save":
      state[index].label = $("labelinput_"+index).value;
      state[index].query = $("queryinput_"+index).value;
      openmail.Application.setData({ keys: { state: state} });
      render();
      updateFolders();
    case "cancel":
      $("display_"+index).style.display="";
      $("input_"+index).style.display="none";
      break;
    case "delete":
      if(confirm("Are you sure you want to delete this saved search?")) {
        state.splice(index,1);
        openmail.Application.setData({ keys: { state: state} });
        render();
        updateFolders();
      }
      break;
    case "create":
         state.push({label:"New Search", query:"from:bob"});
         openmail.Application.setData({ keys: { state: state} });
         render();
         updateFolders();
         break;
    }
}
</script>
</body>
</html>
```

Kitchen Sink

YAHOO! MAIL

As in "everything but the kitchen sink," and that's probably in here somewhere too. This app used to be called "Hello World," which is a cruel and misleading name to give an app that sounds like it's meant for noobs, but is really just a circus of API calls in no pedantically-useful arrangement.

The config:

```
<openmail_app_config version="2">
    <data>
        <mail_message>
            <action>
                <launch>
                    <view>full</view>
                    <target_zone>tab</target_zone>
                    <context>dragdrop</context>
                </launch>
            </action>
        </mail_message>
    </data>
    <events>
        <load>
            <action>
                <launch>
                    <view>full</view>
                    <target_zone>hidden</target_zone>
                    <context>load</context>
                    <ttl>30</ttl>
                </launch>
            </action>
        </load>
    </events>
</openmail_app_config>
```

The "full" view:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script src="http://yui.yahooapis.com/2.5.1/build/yahoo-dom-event/yahoo-dom-
event.js"></script>
<script src="http://yui.yahooapis.com/2.5.1/build/json/json-min.js"></script>
<script src="http://mail.yimg.com/a/lib/om/crossframe/1.0.13/crossframe-min.js"></script>
<script
src="http://mail.yimg.com/a/lib/om/om_api_public/1.0.1/om_api_public.js"></script>
<script src="http://twiki.corp.yahoo.com/pub/TWiki/ChecklistPlugin/itemstatechange.js"
language="javascript" type="text/javascript"></script><script
src="http://twiki.corp.yahoo.com/pub/TWiki/TimeTablePlugin/timetabletooltips.js"
language="javascript" type="text/javascript"></script></head>
<p />
<body>
Hello World!<br>
<!-- commenting out till we have a proper doc location for 318-->
<!--a href='http://twiki.corp.yahoo.com/view/Mail/YmailOpen'>Openmail </a-->
<hr>
<div id="params"></div>
<hr>
Search: <input id="query">
<button onclick="openmail.Mail.search({ query: document.getElementById('query').value
})">Go</button>
<hr>
Compose an email to joe:
<input id="body"><button onclick="openmail.Mail.compose({ to: 'joe@joe.joe', subject: 'hi
joe', body: document.getElementById('body').value })">Compose</button>
<hr>
Close self:<button onclick="openmail.Application.closeView()">Close</button>
<hr>
```

```
Open another tab with different content: <button onclick='openmail.Application.openView({
id: "otherTab", view: "full", target: "tab", title:"Y! Groups", context:"tab",
parameters: {text:"data for the tab"} })'>Open Tab</button>
<br>
Open a dialog: <button onclick='openmail.Application.openView({ id: "otherDialog", view:
"full", target: "dialog", height: 500, parameters: {textData:"some data for the dialog"},
title:"Bunny!", context:"dialog"})'>Open Dialog</button>
<br>
Open a dialog and close it 10s later:
<button onclick='openmail.Application.openView({ id:"otherDialog", view: "full", target:
"dialog", parameters: {textdata:"this dialog will close itself"}, title:"Self Closing",
context: "dialog self-closed"} );setTimeout("openmail.Application.closeView({ id:
\"otherDialog\" })",10000)'>Open Dialog</button><br>
<hr>
<button onclick='openmail.Application.closeView({ id: "otherTab" })'>Close that
tab</button> - <button onclick='openmail.Application.closeView({ id: "otherDialog"
})'>Close Dialog</button><br>
<hr>
<button onclick='loadHandler()'>refresh</button><br>
<hr>
<button onclick='loadDiggStuff()'>load some digg API stuff</button><br>
<hr>
Set Example Data (mykey1=myval1, mykey2=myval2, mykey3=myval3): <button
onclick='testSetData()'>set test data</button><br>
Get Example Data : <button onclick='testGetData()'>get test data</button><br>
<p />
<script>
function loadHandler() {
    openmail.Application.setFolderConfig({
        label:" + more",
        tooltip:"click me",
        subFolders:[{
            label:"eden_people",
            tooltip:"sub1 tt"
        },{
                label:"flexcoders",
                tooltip:"sub2 tt"
        },{
                    label:"shoop da woop",
                    tooltip:"blah"
        }
        ]
    });
}

var diggURL = "http://services.digg.com/stories/diggs";
function loadDiggStuff() {
    openmail.Application.callWebService( { url: diggURL, method: "GET", parameters:
{appkey:"http://apidoc.digg.com"} }, function(args) {
        alert(["got data: ", args.data].join(''));
    });
}

function testSetData() {
     openmail.Application.setData( {
       keys: {
         mykey1: "myval1",
         mykey2: "myval2",
         mykey3: "myval3"
       }
     } );
}

function testGetData() {
    openmail.Application.getData(
        { keys: ["mykey1", "mykey2", "mykey3" ] },
        function(result) {
          alert( "mykey1=" + result.data["mykey1"] + "\n" +
                 "mykey2=" + result.data["mykey2"] + "\n" +
                 "mykey3=" + result.data["mykey3"] );
        }
    );
}
```

```
function refresh() {
    openmail.Application.getParameters(function(args){
        var ta = document.createElement("textarea");
        ta.value = YAHOO.lang.JSON.stringify(args);
        ta.style.width="100%";
        ta.style.height="50px";
        document.getElementById("params").appendChild(ta);
        if (args.context == "load") {
            loadHandler();
        }
    });
}

YAHOO.util.Event.onDOMReady(refresh);
</script>
</body>
</html>
```

YQL Sample App

This app demonstrates: use of the Yahoo Mail Application Development Platform AuthService API to call the YQL web service.

Yahoo! Query Language (YQL) lets developers query, filter, and combine data across Yahoo! properties without having to talk to each property's web service seperately. Developers get the simplicity of only having one web service to contact and the power of manipulating disparate web services with powerful, SQL-like queries.

When using YQL, you may want to access data on Yahoo! properties (or beyond) that requires user authorization. OAuth facilitates such authorization. It allows users to share private resources stored on one site with another site without having to hand out their user name and password.

Before working with this demo, you should read up on YQL and study the basics of OAuth.

App Config

The config says to launch our "full" view in a tab when the application icon is clicked.

```
<?xml version="1.0"?>
<openmail_app_config version="2">
  <events>
    <click>
      <action>
        <launch>
          <view>full</view>
          <target_zone>tab</target_zone>
          <context>load</context>
          <title>YQL</title>
        </launch>
      </action>
    </click>
  </events>
</openmail_app_config>
```

<?xml version="1.0"?> <openmail_app_config version="2"> <events> <click> <action>
<launch> <view>full</view> <target_zone>tab</target_zone> <context>load</context>
<title>YQL</title> </launch> </action> </click> </events> </openmail_app_config>

Since we'll be using the AuthService API, we need an authorization configuration, which is explained here.

Auth Config
Our auth configuration says that:

- The app will access to one web service, nicknamed 'yahoo'.
- The app proves its identity to OAuth using the data from the Consumer and Signature tags. This is all the app needs for "2-legged" auth. If you are going to build your own app, acquire a key and secret for your app at https://developer.yahoo.com/dashboard/.
- To access protected data that the user has granted the app privileges to read (and possibly write), some additional tags are needed: Request, Access, and Authorization. These are used in the "3-legged token dance."

```
<?xml version="1.0"?>
<Application>
  <Service name="yahoo" type="OAuth">
    <Consumer
key="dj0yJmk9emJIbk1YRDJ4ejd1JmQ9WVdrOVkzUmlaMFI0TXpRbWNHbzlNVEUxTlRFeE1UUXpPUS0tJnM9Y29u
c3VtZXJzZWNyZXQmeD0yNQ--"
              secret="861cf734c7cc47ded49020ab1e58a726a7104340"/>
    <Request url="https://api.login.yahoo.com/oauth/v2/get_request_token" method="GET"/>
    <Access url="https://api.login.yahoo.com/oauth/v2/get_token" method="GET"/>
    <Authorization url="https://api.login.yahoo.com/oauth/v2/request_auth"/>
    <Signature method="HMAC-SHA1"/>
  </Service>
</Application>
```

Sample View
The "full" view defines the bulk of our application. When it loads, the auth.init function is invoked. It calls Application.getAuthService to obtain the auth service object associated with the service nicknamed 'yahoo' in the Auth Config above. It then calls AuthService.getStatus to check whether the user has authenticated themselves. Provided the user hasn't already logged on, they're given an option of doing so.

Trace through the login() flow to see how authentication involves use of the AuthService.getUserAuthorizationURL and AuthService.notifyAuthorizationReceived calls.

Even without logging on, they can do YQL queries to services that don't require user authentication.

It provides a text box where the user may enter YQL queries. When a query is submitted, it passes those to the YQL service using AuthService.callWebService, requesting results in JSON format. Those results are rendered in a YUI Tree View.

Some YQL queries worth trying that don't require authentication:

- show tables
- desc social.profile
- select * from search.web where query="peanut butter cookies"

Some YQL queries worth trying that do require authentication:

- select * from social.profile where guid=me
- select * from social.updates where guid=me
- select * from social.contacts where guid=me
- select * from social.presence where guid=me

```
<html>
    <head>
        <title>YQL Sample Application</title>
        <link rel="stylesheet" type="text/css"
href="http://yui.yahooapis.com/combo?2.6.0/build/button/assets/skins/sam/button.css&2.6.0
/build/treeview/assets/skins/sam/treeview.css">
        <style>
            body, table {
                font-family: Arial, Helvetica, sans-serif;
                font-size: 12px;
                } #jsontree {
                    width: 600px;
                } .key {
                    float: left;
                    color: #111;
                    padding-left: 5px;
                } .value {
                    float: right;
                    width: 400px;
                } .str {
                    color: #c00;
                } .num {
                    color: #007;
                } .obj {
                    color: #070;
                } .ygtvitem table {
                    width: 100%;
                }
            #login,
            #logout,
            #complete {
                display: none;
                float: right;
            }
            #result_container {
                display: none;
            }
            .auth_wizard {
                display:none;
                float:right;
                padding: 5px;
                width:220px;
                background-color: #EEE;
            }
        </style>
        <script type="text/javascript"
src="http://yui.yahooapis.com/combo?2.6.0/build/yahoo-dom-event/yahoo-dom-
event.js&2.6.0/build/element/element-beta-min.js&2.6.0/build/button/button-
min.js&2.6.0/build/json/json-min.js&2.6.0/build/treeview/treeview-min.js"></script>
        <script src="http://mail.yimg.com/a/lib/om/crossframe/1.0.14/crossframe-
min.js"></script>
        <script
src="http://mail.yimg.com/a/lib/om/om_api_public/1.0.1/om_api_public.js"></script>
```

```
        <script>
            YAHOO.namespace("yql.auth");
            YAHOO.namespace("yql.auth.wizard");
            YAHOO.namespace("yql.util");
            YAHOO.yql.auth.guid = "";
            YAHOO.yql.auth.profile_url =
"http://social.yahooapis.com/v1/user/{guid}/profile";
            YAHOO.yql.auth.url = "http://query.yahooapis.com/v1/yql";

            YAHOO.yql.auth.query = function() {
                var ta = YAHOO.util.Dom.get('queryArea');
                YAHOO.yql.util.display(['result_container'],'none');

                YAHOO.yql.auth.service.callWebService(
                {
                    url:YAHOO.yql.auth.url,
                    method:"GET",
                    parameters: {
                        format:'json',
                        q:ta.value
                    }
                },
                function(response) {
                    var data;
                    if (response.error) {
                        data = {Result:"Call failed."};
                    } else {
                        data = YAHOO.lang.JSON.parse(response.data);
                    }
                    YAHOO.yql.util.draw(data);
                    YAHOO.yql.util.display(['result_container'],'block');
                });
            };

            YAHOO.yql.auth.load_profile = function() {
                YAHOO.yql.auth.service.callWebService(
                {
                    url:YAHOO.yql.auth.profile_url.replace('{guid}',YAHOO.yql.auth.guid),
                    method:"GET",
                    parameters: { format:'json' }
                },
                function(response){
                    if (response.error) {
                        return;
                    }
                    var data = YAHOO.lang.JSON.parse(response.data);
                    YAHOO.util.Dom.get("profile_id").innerHTML = data.profile.givenName +
" " + data.profile.familyName;
                    YAHOO.yql.util.display(['logout'],'block');
                });
            };

            YAHOO.yql.auth.init = function() {
                YAHOO.openmail.Application.getParameters(function(response) {
                    YAHOO.yql.auth.guid = response.user.guid;
                    YAHOO.yql.auth.service =
YAHOO.openmail.Application.getAuthService({profile:'yahoo'});
                    YAHOO.yql.auth.wizard.init_launch();
                    YAHOO.yql.auth.wizard.init_redirect();
                    YAHOO.yql.auth.wizard.init_complete();
                    YAHOO.yql.auth.service.getStatus(YAHOO.yql.auth.handle_status);
                });
            };

            YAHOO.yql.auth.wizard.init_launch = function() {

YAHOO.util.Event.addListener('signout_link','click',YAHOO.yql.auth.log_out);
                YAHOO.util.Event.addListener('signin_link','click',
                    function() {
                        YAHOO.yql.util.display(['login'],'none');
                        YAHOO.yql.auth.service.getUserAuthorizationURL(
                            {},
                            YAHOO.yql.auth.handle_url
```

```
                    );
                }
            );
        }

        YAHOO.yql.auth.wizard.init_redirect = function() {
            YAHOO.yql.auth.cancel_redirect = new
YAHOO.widget.Button('cancel_redirect');

YAHOO.util.Event.addListener('cancel_redirect','click',YAHOO.yql.auth.log_out);

            YAHOO.util.Event.addListener('redirect_link','click',
                function() {
                    YAHOO.yql.util.display(['redirect'],'none');
                    YAHOO.yql.util.display(['complete'],'block');
                }
            );
        }

        YAHOO.yql.auth.wizard.init_complete = function() {
            YAHOO.yql.auth.complete = new YAHOO.widget.Button('complete_link');
            YAHOO.util.Event.addListener('complete_link','click',
                function() {

YAHOO.yql.auth.service.notifyAuthorizationReceived(YAHOO.yql.auth.login_ok);
                }
            );

            YAHOO.yql.auth.cancel_complete = new
YAHOO.widget.Button('cancel_complete');

YAHOO.util.Event.addListener('cancel_complete','click',YAHOO.yql.auth.log_out);
        }

        YAHOO.yql.auth.handle_status = function(response) {
            if(response.error) {
                // dont show any login links
                return;
            }
            if(response.data.status=='authorization required') {
                YAHOO.yql.util.display(['login'],'block');
            }
            else if (response.data.status=='ok') {
                YAHOO.yql.auth.login_ok(response);
            }
        };

        YAHOO.yql.auth.handle_url = function(response) {
            if (response.error) {
                // handle error
            }
            YAHOO.util.Dom.get('redirect_link').href = response.data.url;

            YAHOO.yql.util.display(['login'],'none');
            YAHOO.yql.util.display(['redirect'],'block');
        };

        YAHOO.yql.auth.login_ok = function(response) {
            if (response.error) {
                // handle error
            }
            YAHOO.yql.auth.load_profile();
            YAHOO.yql.util.display(['complete'],'none');
        };

        YAHOO.yql.auth.log_out = function() {
            YAHOO.yql.util.display(['logout','login','complete','redirect'],'none');
            YAHOO.yql.auth.service.removeAuthorization(function(response) {
                if (response.error) {
                    // handle error
                }
                YAHOO.yql.util.display(['login'],'block');
            });
```

```
            };

            // Util functions from here on out:

            YAHOO.yql.util.display = function(array, display) {
                var i;
                for (i = 0; i < array.length; i += 1) {
                    YAHOO.util.Dom.get(array[i]).style.display = display;
                }
            };

            YAHOO.yql.util.draw = function(data) {
                var tv = new YAHOO.widget.TreeView('results'), root = tv.getRoot();
                YAHOO.yql.util.DFNodeDump('Result', data, root);

                tv.draw();

                if (root.children[0]) {
                    root.children[0].expand();
                    if (root.children[0].children[0]) {
                        root.children[0].children[0].expand();
                    }
                }
            }

            YAHOO.yql.util.DFNodeDump = function(k, v, parent) {
                switch (typeof v) {
                    case 'string':
                        new YAHOO.widget.TextNode(YAHOO.yql.util.getHTMLString(k, '"' + v
+ '"', 'str'), parent, false);
                    break;

                    case 'number':
                    case 'function':
                    case 'boolean':
                    case 'undefined':
                        new YAHOO.widget.TextNode(YAHOO.yql.util.getHTMLString(k, v,
'num'), parent, false);
                    break;

                    case 'object':
                        if (v === null) {
                            new YAHOO.widget.TextNode(YAHOO.yql.util.getHTMLString(k, v,
'str'), parent, false);
                        }
                        else if (v instanceof Array) {
                            var node = new
YAHOO.widget.TextNode(YAHOO.yql.util.getHTMLString(k, '(Array)', 'obj'), parent, false);
                            for (var i = 0; i < v.length; i++) {
                                YAHOO.yql.util.DFNodeDump(i + '', v[i], node);
                            }
                        }
                        else if (v instanceof Object) {
                            var node = new
YAHOO.widget.TextNode(YAHOO.yql.util.getHTMLString(k, '(Object)', 'obj'), parent, false);
                            for (var prop in v) {
                                if (v.hasOwnProperty(prop)) {
                                    YAHOO.yql.util.DFNodeDump(prop + '', v[prop], node);
                                }
                            }
                        }
                    break;
                }
            };

            YAHOO.yql.util.getHTMLString = function(k, v, cls) {
                return ['<span class="key">', k + '', '</span><span class="value ' + cls
+ '">', YAHOO.yql.util.escapeHTML(v + ''), '</span>'].join('');
            };

            YAHOO.yql.util.escapeHTML = function(v) {
                return v.replace(/[<>'"]/g, function(s) {
                    var sym = {
```

```
                    '<': '<',
                    '>': '>',
                    "'": ''',
                    '"': '"'
                }

                return sym[s];
            });
        };

        YAHOO.util.Event.addListener(window, 'load',  YAHOO.yql.auth.init);
    </script>
</head>
<body class="yui-skin-sam">
    <div id="logout">
        <p>Signed in as <span id="profile_id"></span> |
            <a href="#" onClick="return false;" id="signout_link">Sign out</a>
        </p>
    </div>
    <div id="login">
        <p><a href="#" onClick="return false;" id="signin_link">Sign in</a></p>
    </div>
    <div id="redirect" class="auth_wizard">
        <p>The following button will take you to a new window where you can authorize
Yahoo! Mail to access to your profile information. After that just come back to
continue.</p>
        <p><span class="yui-button yui-link-button"><span class="first-child"><a
href="#" id="redirect_link" target="_blank">Yes, sign me in!</a></span></span></p>
        <p>Or, you can continue what you were doing (and sign in later).</p>
        <button id="cancel_redirect">No, thanks</button>
    </div>
    <div id="complete" class="auth_wizard">
        <p>Once you've completed the authorization process, please click here:</p>
        <p><button id="complete_link">All done!</button></p>
        <p>Or, you can cancel the process:</p>
        <button id="cancel_complete">Cancel</button>
    </div>
    <br/>
    <p>Type your YQL query in the box below. Before signing in you do things
like:</p>
    <p>
    <code>show tables</code><br/>
    <code>desc <table name></code><br/>
    <code>select * from search.web where query="Yahoo! Mail"</code>
    </p>
    <p>After signing in you can also access your personal data:</p>
    <p>
        <code>select * from social.profile where guid = me</code><br/>
        <code>select * from social.updates where guid = me</code><br/>
    </p>
    <b>Query:</b>
    <br/>
    <textarea id='queryArea' cols=80 rows=5></textarea>
    <br/>
    <button onclick='YAHOO.yql.auth.query()'>Run Query</button>
    <div style="clear:both"></div>
    <br/>
    <div id="result_container">
        <b>Results: </b>
        <br/>
        <div  id="results"></div>
    </div>

</body>
</html>
```

YAHOO! MAIL

Developer FAQs

We get lots of questions from our app developers. Here are some of our most frequently asked questions:

callWebService

My callWebService call isn't returning any data, and is returning a cryptic "Error #2048" in the error object.

All callWebService external requests are relayed through a Flash object. In the Flash security model, the target site for a web service call needs to have a crossdomain.xml file available (typically at the server root) with the appropriate permissions granted (for details, see http://livedocs.adobe.com/flash/8/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00001621.html. Part of the security model also seems to include the production of cryptic error messages.) The external site needs to allow Flash code from "*.yimg.com" to make web service calls. The following file, saved as crossdomain.xml at the root level of the server you're trying to access, will accomplish this:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*.yimg.com" />
</cross-domain-policy>
```

There's a valid crossdomain.xml file on my target server, but HTTPS callWebService calls are failing.

If you're trying to make a web service call over SSL, you need to add the attribute 'secure = "false"' to the "allow-access-from" element, as in: <allow-access-from domain="*.yimg.com" secure="false"/>)

This tells Flash that it's OK to request secure data using code loaded from a non-secure site.

My GET/POST variables aren't getting passed through to my target server intact.

If you're making a signed web service call, it's best if none of your variable names contain dots. The signing request is processed by PHP on our servers, which will silently alter those variable names. PHP automatically converts all dots in GET/POST variables to underscores. We'll correct this at some point, but in the meantime, don't use dots in your variable names.

callWebService just stopped working entirely, but only in some browsers, specifically IE6/7. FF3 works fine. In fact, I'm not getting responses from any API calls.

This will happen if you have HTML that looks like this:

```
<a href="#" onlick="doStuff()">click me</a>
```

In many browsers (IE6, IE7, and FF2 included), our cross-frame communication mechanism requires a constant fragment identifier (everything after the "#") in the application iframe URL. A click on an element with href="#" removes the fragment identifier from the URL for the active iframe's src attribute. In the affected browsers, after that URL is modified, no communication can happen between your app and the web service Flash object. In fact, at that point your communication with the entire OpenMail API effectively ends--almost all of the API methods require the transmission of cross-frame data.

While we work to fix this on our end, there are a couple of things you can do to fix this in your apps for now. One easy thing to do is just to add "return false;" to your onclick handlers for existing tags with fragment identifiers, like so:

```
<a href="#" onclick="doStuff(); return false;">click me</a>
```

...which leaves the iframe's fragment identifier intact.

I can't use an arbitrary string for my POST data--all parameters have to be in key-value pairs.

This is correct.

No, really, can't I use an arbitrary string for my POST data?

Unfortunately, no. It's a Flash limitation. You'll need to configure your remote server to accept your POST data in a key-value pair, or create a proxy that replaces "q=arbitrary_string_data" with "arbitrary_string_data" and re-POSTs to your server.

I've got a cool REST-based API, and I'd like to use HTTP requests that aren't GET or POST. Can I?

Unfortunately, no. This is another Flash limitation. One possible workaround is to configure your server (or a proxy) to interpret an added GET/POST parameter like "method=PUT" or "method=DELETE" and deal with the rest of the supplied data as if it originated from an actual PUT or DELETE request.

IE6-Specific Issues

IE6 is always displaying a horizontal scrollbar in my app when my content isn't wide enough to require one.

In IE6, when rendering an Iframe in Standards Mode, a horizontal scroll bar will appear whenever a vertical scroll bar is present. Adding html { _overflow-x: hidden; } to your CSS will prevent it from appearing. (Note that this will also prevent the horizontal scrollbar from *ever* appearing, even when it might be useful.)

My signed calls occasionally fail with "Error #2032" on IE6 but they work fine with every other browser.

IE6 appears to have caching issues with the web service Flash object. Use one of the following methods to work around the problem by not caching your data:

- Add a random query parameter to the callWebService() parameters object to cache bust your request.
- Add "max-age=0,must-revalidate" to your server's "Cache-Control" header.

Caching

I can't see changes I've just made to my application.

Application data is sent out with a "Cache-Control: max-age=300" HTTP header. To immediately see changes in an application, try disabling caching in your browser. Alternatively, wait five minutes.

I still can't see changes I've just made to my application.

You need to close any tabs your application has opened. Currently opened tabs/dialogs will not be updated in place. You need to close those views and re-invoke your application.

Views

I'm getting the "Oops! Your application is currently unavailable. Please try again in a couple minutes." dialog whenever I try to run my app. Should I wait?

Likely, no. That dialog will appear if an unknown view is requested by the launch handler or an openView call. Make sure that your view names refer to actual views stored in your application.

I'm using numbers as view names, and strange things are happening in the development tool.

Don't use numbers as view names.

Authentication/Authorization

I'm not able to use an auth profile that I know I've created.

Auth profiles are contained within an application, and are only available from within that individual application. If you want to use an auth profile in more than one app, you'll need to copy and paste the auth XML into each app individually.

YAHOO! MAIL

Application Technical Requirements

As an application developer for YMDP, you will need to create apps that meet our Technical Requirements. If you have any questions, please contact us for clarifications.

A. Dimensions, Functionality, and Editorial Requirements For Applications:
   a. To optimize for any variations in page assembly, as well as to maximize future scalability for any environmental changes, Yahoo! encourages that Applications be developed using a fluid presentation that can accommodate subtle width and height variations. Further, usage of semantic markup (to separate content from its presentation) is highly recommended.
   b. Content displayed through any Application must remain within the dimensions of the Application's placement on the applicable Yahoo! Mail pages and cannot "float" or "break through" to other parts of the End User's Yahoo! Mail page.
   c. Animation, video, audio and any other interactive Application content (collectively, "Interactive Content") on the user profile page must be initiated only upon user click, unless the user has explicitly authorized the automatic initiation of such Interactive Content.
   d. With the sole exception of third-party user generated content, You must have intellectual property rights to any content, code or other materials displayed and be permitted to display such content in the Application.
   e. You may display third-party user generated content in its Application only if You implement a compliant Digital Millennium Copyright Act (DMCA) notice and takedown policy and process.
   f. Applications may not re-direct the IFrame or top/parent.
   g. Applications may not embed additional inner IFrames pointing to non-Yahoo! servers.
   h. Applications may not include external references to javascript or CSS; prior to deployment, all such dependencies must be inlined with the main application code or hosted by Yahoo!.
   i. Applications may not include object/embed tags pointing to non-Yahoo! servers.
   j. Applications must use the provided APIs for access to external data.
   k. Applications must not contain busy loops or perform browser denial of YMDP attacks.
   l. Applications should successfully run through JSLint check (www.jslint.com/lint.html).
   m. Detected obfuscation is disallowed.

n. Applications may not use /Yahoo/i in variable or function names.
o. Applications may not use global variables.
p. Applications may not use YAHOO.* namespace.
q. Applications may not use eval() or similar functions such as SetTimeout(), SetInterval(), or function() on strings fetched from un-trusted servers.
r. Applications may not contain cross-site scripting vulnerabilities.
s. Applications may not get or set document.cookie from javascript code.
t. Remote web service apis that return personalized JSON data must be protected with a secret, user-specific token.
u. Remote servers may not retrieve, access, or store the referrer http header.
v. All code is subject to review by Yahoo!
w. Yahoo! reserves the right to modify or require You to modify any content (e.g., to "flatten" any animated .gifs to remove the animation) displayed on any Application to conform to these YMDP Guidelines.
x. Yahoo! suggests that You develop your Application using industry standard internationalization techniques to enable adoption by the largest number of End Users.
y. You may not require End Users to enter any username or password within Your Application.
z. Third party resource requests should not be triggered as a result of the Load event.