Yahoo! Messenger Plug-in SDK User Guide and Reference

June 19, 2006

Contents

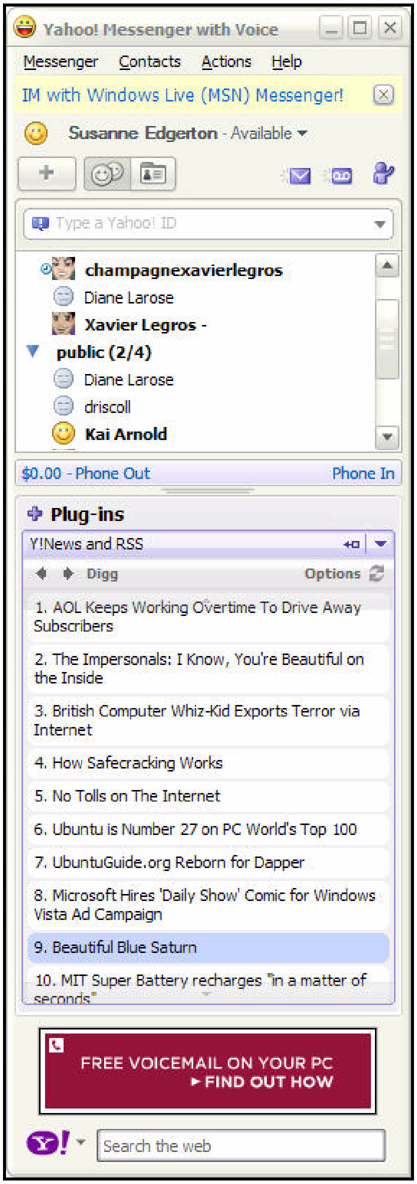
User Guide

Overview

The Messenger Plug-in SDK is a JavaScript and C++ API that you use to create add-ons with collaborative features that can run inside Yahoo! Messenger. Plug-ins run in the main Messenger window, or as part of an IM conversation. Yahoo! users can download Plug-ins from the Yahoo! Messenger Plug-in Gallery. Create plug-ins that are as simple as an HTML web page, or as sophisticated as a map collaboration tool that uses ActiveX, Flash, DHTML, AJAX or any other browser-based technology.

What is a Messenger Plug-in?

A Yahoo! Messenger Plug-in is an HTML page that runs inside Yahoo! Messenger. Developers create plug-ins using the Yahoo Messenger Plug-in SDK —a toolkit of JavaScript and C++ APIs. With the Plug-in SDK APIs, you can create exciting and helpful collaborative plug-ins and make them available to anyone through the Messenger Network, offering you a chance to show your next great idea to over 60 million users.



champagnexavierlegros
Diane Larose
Xavier Legros -
public (2/4)
Diane Larose
driscoll
Kai Arnold

1. AOL Keeps Working Overtime To Drive Away Subscribers
2. The Impersonals: I Know, You're Beautiful on the Inside
3. British Computer Whiz-Kid Exports Terror via Internet
4. How Safecracking Works
5. No Tolls on The Internet
6. Ubuntu is Number 27 on PC World's Top 100
7. UbuntuGuide.org Reborn for Dapper
8. Microsoft Hires 'Daily Show' Comic for Windows Vista Ad Campaign
9. Beautiful Blue Saturn
10. MIT Super Battery recharges "in a matter of seconds"

What Can Plug-ins Do?

As HTML pages embedded inside Yahoo! Messenger, Plug-ins can transfer files, access the buddy list, send instant messages, display graphics and text, access web sites, display your calendar, and more.

Plug-ins are launched from either the content tab window, or from the conversation window. Where your plug-in is launched from will depend on the type of plug-in you decide to create. Certain functionality is available to both types of plug-ins, but some functionality is specific to each type. For example, Tab plug-ins can launch a secondary window and communicate with it; Conversation plug-ins can intercept and send instant messages, and write text to the history window.

What Types of Plug-ins Can I Create?

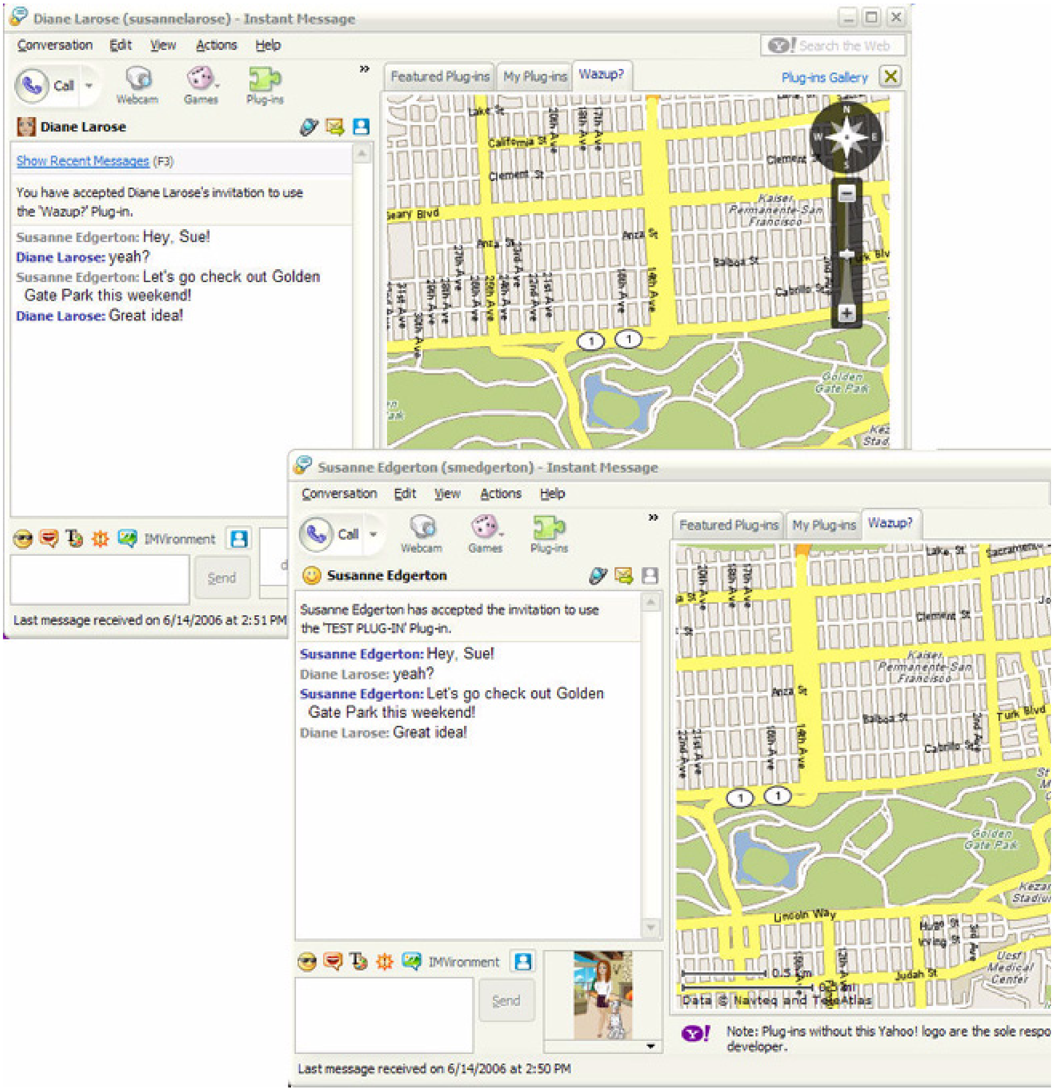
There are two types of plug-ins you can create, defined by what part of Messenger launches them: the conversation window or the content tab. While each type of plug-in is capable of specific tasks, much of the functionality in the API is available to both.

Tab Plug-ins

Tab plug-ins are launched by the user from the main messenger (content tab) window and can interact with the buddy list to receive notifications, such as when a user selects a friend, or when a friend’s status has changed. You can also open a sliding side window, to display additional content, and message between the two windows.

Conversation Plug-ins

Conversation plug-ins are launched from the conversation window and provide collaborative functionality, such as intercepting text as it is typed, and communication between the plug-ins of each user. For example, a map plug-in can allow users to collaborate a geographic location by allowing each user's modifications to the map to be immediately shown in the other user's map view.



Show Recent Messages (F3)

You have accepted Diane Larose's invitation to use the 'Wazup?' Plug-in.

Susanne Edgerton: Hey, Sue!
Diane Larose: yeah?
Susanne Edgerton: Let's go check out Golden Gate Park this weekend!
Diane Larose: Great idea!

Last message received on 6/14/2006 at 2:51 PM

Susanne Edgerton has accepted the invitation to use the 'TEST PLUG-IN' Plug-in.

Susanne Edgerton: Hey, Sue!
Diane Larose: yeah?
Susanne Edgerton: Let's go check out Golden Gate Park this weekend!
Diane Larose: Great idea!

Last message received on 6/14/2006 at 2:50 PM

Note: Plug-ins without this Yahoo! logo are the sole respo developer.

Components

A plug-in itself is an HTML file with JavaScript code that calls the Plug-in APIs. A manifest text file specifies configuration settings for the plug-in, such as where Messenger can find the plug-in at run-time. Each plug-in that a user selects, will download an HTML file and a manifest file to their machine.

HTML File

The HTML file contains a combination of HTML tags and JavaScript API code that implements the plug-in. When a user selects a plug-in from the Yahoo! Messenger Network, their download will contain an HTML file and a corresponding manifest file.

Manifest File

The manifest file is a text file that contains configuration settings for each plug-in. Each plug-in has a corresponding manifest file, located in the manifest directory. Messenger uses this file to know what functionality is available to the plug-in, including where the plug-in is located, and how many user's are allowed. For a complete list of manifest file settings, see "The Manifest File," in the Getting Started section.

Terminology

The terminology associated with Messenger plug-ins are as follows:

History Window

A pane in the conversation window that displays the messages that have been sent and received.

Conversation Window

The window that appears when the user wishes to send an instant message to another user.

Content Tab

The main Messenger window.

Toast

A small alert window that pops up in the tray when a friend’s status changes, or a instant message has been received.

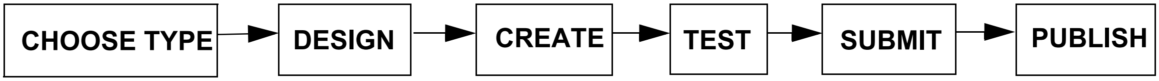
Input Window

A pane in the conversation window where the user types their instant message before it is sent.

Work Flow

The steps needed to create a plug-in are:

1. Decide which type of plug-in you will to create: Conversation or Tab
2. Design the plug-in
3. Create the plug-in using HTML and JavaScript
4. Test the plug-in
5. Submit the plug-in to Yahoo! for approval
6. Publish the plug-in on the Yahoo! Messenger Network



Publishing Your Plug-in

Publishing your plug-in to the Yahoo! Messenger Plug-in Gallery, located at http://gallery.yahoo.com/messenger, makes it available to all users on the Messenger Network for download. To be published, Yahoo! must approve it based on security and appropriateness guidelines. Publishing your plug-in is an easy process.

For security reasons, ActiveX plug-ins and plug-ins that access the Address Book may take longer to approve. Plug-ins designed for commercial usage will not be accepted through this process; if you have created a commercial plug-in that you wish to publish on Yahoo!, send an email to messenger-plugins@yahoo-inc.com.

Plug-in UI Guidelines

Since plug-ins are hosted by Yahoo! Messenger, it's important that they fit well aesthetically and functionally with Yahoo! Messenger. To following recommended UI guidelines will help you to create plug-ins that work well with Messenger. For the most up-to-date information, go to http://gallery.yahoo.com/messenger.

Compact the Plug-in Size

The size of a Tab plug-in in the main window should be relatively compact: less than 100 pixels high. Some plug-ins may be higher, such as a calendar plug-in, but most plug-ins should have a vertical size of less than 100 pixels.

Keep the Content Fresh

Keep the content of the main plug-in window "fresh", using such features as buttons and active scrolling to keep content up to date. For example, the Amazon plug-in has an auto scroll list of top rated books that the user can switch to top sellers for electronics, and other Amazon categories.

Main Window Content

For Tab plug-ins, the main window content should offer instant access to data that is immediately relevant. For instance, if a user is looking at email or working on an excel spreadsheet, a click on Messenger should show data related to the user at that instant: the number of emails they have in Yahoo! Mail, the number of local traffic accidents, bid on eBay items, questions answered on Y! Answers, etc. If the user wants more info at that point, the developer should use a secondary window to present that data and offer the user the ability to drill down on the data.

Secondary Window Content

User or descriptive content in a Tab plug-in—content that needs user attention for more than a couple of seconds—should be displayed in the secondary (sliding) window. For example, in a news plug-in, if a user clicks on a news item, the news summary with a picture could display in the secondary window. The main window should drive access to new content through a set of buttons or explicit action from the user.

Use Collaborative Features

Plug-in developers are encouraged to leverage the buddy list. The ability for user plug-ins to communicate with each other is one of the key benefits of developing a Yahoo! Messenger plug-in. For instance, in the Amazon plug-in, you can see a friend's wish list by clicking on the friend in the buddy list.

Submission Guidelines

To be accepted, submissions should adhere to the following:

- The plug-in must perform the function described in the plug-in description.
- The debug window must not appear when the plug-in is run.
- The plug-in must not contain copyrighted material that you are not the sole owner of, or have not received permission to use.
- The plug-in must not contain any imagery, text or sounds that may be offensive.

In general, Plug-ins should to have the following attributes:

- Look attractive
- Serve a useful purpose
- Have some unique features to differentiate it from other Plug-ins in the Gallery

Submissions may be rejected if they fall exceptionally short of these standards.

Getting Started

This page will get you quickly going on creating your own Messenger plug-ins.

Installing the SDK

The Messenger Plug-in SDK contains the SDK Documentation and sample plug-in files. You must have version 8.x or later of Yahoo! Messenger installed on your machine.

To install the Messenger Plug-in SDK

1. In the Yahoo Messenger folder, unzip the Test.zip file into the subdirectory "plugin", located where Yahoo! Messenger was installed, typically "C:\Program Files\Yahoo!\Messenger\plugin".

   You should end up with the following folder structure:

```
<messenger installation directory>/Plugin/Test/ContentTab/test.yplugin
<messenger installation directory>/Plugin/Test/SidePanel/test.yplugin
```

   In the test.yplugin folder you will find a MANIFEST folder that contains the manifest file ("plugin.property"). This file is the heart of the plug-in and specifies the features of the plug-in.

2. Double click the file "PluginTest.reg" to update your registry. This adds two keys to HKEY_CURRENT_USER\Software\Yahoo\Pager:
   - a DWORD value named "Use XML Format", value of 1,
   - a DWORD value named "Use SendDataThru", value of 1.

3. To run multiple versions of Messenger on one machine, create a DWORD registry value named “Plural” at **`HKEY_CURRENT_USER\Software\Yahoo\Pager\Test`** and set to something other than 0 (zero).

   You can now run multiple versions of Yahoo! Messenger for testing Conversation plug-ins. Be sure that the “Sign in automatically” box is not checked on the Messenger login page. You may have to logout to uncheck this box.

Please note that the Messenger wasn’t designed and implemented to run multiple instances on the same machine and they can collide and interfere with each other.

To Run the Test Plug-ins

Now that you have installed the Messenger Plug-in SDK, you can try out the test plug-ins that come with the SDK.

1. Run and log in to Messenger.
2. To invoke the Tab test plug-in, from the Messenger menu, select Actions|Load Test Plug-in.
3. To invoke the Conversation test plug-in, start a conversation with a friend in the buddy list, and from the conversation window, select Actions|Load Test Plug-in.

Yahoo! and third-party plug-ins approved by Yahoo! can be launched from the Plug-in Picker accessed by selecting Actions|Choose a Plug-in from the main Messenger tab, or the conversation window.

The Manifest File

The manifest file contains information that Messenger uses to access and configure your plug-in. Each plug-in that a user downloads to their machine will have a manifest file that tells messenger information about the plug-in, such as where the plug-in module can be found, whether it's a Tab or Conversation plug-in, whether it can transfer files, etc.

The manifest file for test plug-ins is called plug-in.properties. For Tab plug-ins this file is located in the following directory.

```
<messenger
folder>\Plugin\Test\ContentTab\test.yplugin\MANIFEST\plugin.properties
```

For Conversation plug-ins, the file is located at:

```
<messenger
folder>\Plugin\Test\SidePanel\test.yplugin\MANIFEST\plugin.properties
```

The manifest file defines the attributes of the plug-in in key/value pairs. The key name must start with an upper case letter. For example, the key name “Name” is allowed; while "NAME" and "name" are invalid.

The most important entry in the manifest file is the **`Location`** entry. This setting tells messenger where to find the main module of your plug-in.

For example, the following section from the sample manifest file, declares the location and locale for the plug-in:

```
Locale=en-us
Location=../maps.html
```

The following table contains all possible manifest file entries:

Property Value	**Value**
Activex	Specifies whether the plug-in can run an active-x control. By default, this is False.
Addressbook	Specifies whether the plug-in can access the address book exposed by Messenger. By default, this is False.
Author	Optional. The author’s name.
Category	Roughly defines the category of the plug-in. Default value is 4. Can be one of the following: 1: games 2: fun 3: info-share 4: generic
Company	Optional. The company name of the plug-in author.
Contracts	Semicolon separated list of contracts supported by the plug-in. For yahoo messenger plug-in, it should only support one contract. And there only two contracts in total supported by Yahoo! Messenger: com.yahoo.messenger.sidepanel and com.yahoo.messenger.contenttab. For example, Contracts=com.yahoo.messenger.contenttab
Description	A short description about the plug-in. Should be short and clear.
Email	Optional. The email address of plug-in author.
Homepage	Optional. The homepage for the plug-in, where more information about the plug-in could be found.
Icon	Optional. A 24x24 pixel icon meaningful to the plug-in. The recommendation is a 24bit PNG file with transparency.
Id	Unique ID assigned by Yahoo to the plug-in. Don't change the ID.
Locale	The locale for the plug-in. Defaults to en-us. For example, Locale=en-us.

Property Value	**Value**
Location	The directory location of the main module of the plug-in. This can be a public URL or local file path. If it's a local file path, it MUST be relative to this file. For example, if the file is located in the parent directory, then the value should be ../<file name>. For example, Location=../maps.html.
Min-height	Minimum height of the plug-in in pixels. Maximum value is 500, minimum value is 0. Default value is 0. For example, Min-height=100
Min-width	Minimum width of the plug-in in pixels. Maximum value is 500, minimum value is 0. Default value is 0. For example, Min-width=100.
Module-type	Defines the main module type of the plug-in. For now, Messenger only supports one: HTML: 1
Name	Human readable plug-in name. Appears in title bar of a Tab plug-in, and in the invitation window of Conversation plug-ins.
Plugin-type	Describes the plug-in type. Currently, two types of plug-ins are supported: 1: Conversation 4: Tab For example, Plugin-type=4 specifies that this is a Tab plug-in.
Prefer-expanded-height	Only applicable to Tab plug-ins. Defines the preferred height if the plug-in is expanded. Default value is 500.
Prefer-expanded-width	Only applicable to Tab plug-ins. Defines the preferred width if the plug-in is expanded. Default value is 500.
Prefer-height	The preferred height of plug-in in pixels. Maximum value is 500. For example, Prefer-height=400.
Prefer-width	Preferred width of the plug-in in pixels. Maximum value is 500. For example, Prefer-width=400.
Sendfile	Specifies whether the plug-in can use file transfer service exposed by Messenger. By default, this is False.
Sendim	Specifies whether the plug-in can send instant messages. By default, this is False.
Timestamp	Timestamp of the plug-in, in ISO8601 format. YYYY-MM-DDThh::mm::ssTSD. For example, 2006-01-01T00:00::00+1:00
Users	The number of users that can use the plug-in.

Property Value	Value
Version	Specifies the version of the plug-in formatted as Major.Minor.Service.
Voice	Specifies whether the plug-in can use the voice service exposed by Messenger. By default, this is False.

Creating "Hello World" Plug-ins

This section will show you how to create basic plug-ins to get you up and running quickly. These plug-ins demonstrate common functionality and the main requirements are for each type for each type of plug-in.

- Creating a Chat Plug-in
- Creating a “Hello World” Tab Plug-in
- Creating a "Hello World" Conversation Plug-in
- Creating a “Hello World” ActiveX Plug-in

Creating a Chat Plug-in

This "Hello World" plug-in demonstrates how to create a chat plug-in that can send messages back and forth between users. Messages are displayed in the plug-in window. This can be either a Tab or Conversation plug-in, depending on how you have set Plugin-type in your manifest file. If you set Plugin-type to 4, this will be a Tab plug-in; if you have set Plugin-type to 1, this will be a Conversation plug-in.

Initializing the Plug-in

To begin, add initialization code, as follows:

```
function OnLoad{
  yahoo = window.external;

  yahoo.SetEventHandler("PluginMessage", onPluginMessage);
  yahoo.LocalReady();
}
```

The call to window.external gets a reference called yahoo to the Conversation plug-in. **`SetEventHandler`** sets up an event handler to catch messages received by the plug-in from the other user, and **`LocalReady`** lets the host know that it is up and ready. Communication to this plug-in host cannot happen until you call **`LocalReady`**.

Define a chat window with a **`div`** called chat to display the messages.

```
div#chat
{
   height:100px;
   overflow: scroll;
}
<div id="chat">
</div>
```

Create an input box and submit button for the user to enter and send their message.

```
<input id="text" type="text"></input><input id="submit" type="submit"
value="Send"></input>
```

Sending and Receiving Messages to and from Friends in the Buddy List

Call the **`SendPluginMessage`** function to send the message text to a specified list of friends. The buddy list is accessed through the collection object Friends.

In the following example, the **`SelectedFriends`** method gets a list of the friends selected by the user in the buddy list. When the user clicks on the submit button, the message is sent to this list of friends.

```
participants["friends"] = new Object();
var sf = new Enumerator(yahoo.SelectedFriends);
for (; !sf.atEnd(); sf.moveNext())
   {
      participants.friends[sf.item().ID] = sf.item();
   }

submit.onclick = function()
{
  for (var i = 0; i < participants.friends.length; ++i)
  {
    yahoo.SendPluginMessage(text.value, participants.friends[i]);
  }
  chat.innerHTML += "<b>Me: " + text.value + "</b><br/>";
  text.value = "";
}
```

The following **`onPluginMessage`** even handler is activated whenever the plug-in receives a message from another user. In this case, the friend’s name and their message are displayed in the chat window.

```
function onPluginMessage(str, friend)
{
  var str = "<b>" + friend.FirstName + " " + friend.LastName + ":</b> " +
    str + "<br/>";
  chat.innerHTML += str;
}
```

Creating a “Hello World” Tab Plug-in

This "Hello World" Tab plug-in demonstrates how to create a plug-in that open and communicate with a secondary window. Secondary windows are only available to Tab plug-ins. A secondary window is a window that opens on command from the Tab plug-in main window.

Initializing the Plug-in

To begin, add initialization code to the plug-in, as follows:

```
var yahoo = null;
function initYahoo()
{
  yahoo = window.external;

  yahoo.SetEventHandler('SecondaryWindowReady',
    onSecondaryWindowReady);
  yahoo.SetEventHandler('SecondaryWindowClosed',
    onSecondaryWindowClosed);
  yahoo.SetEventHandler('SecondaryWindowMessage',
    onSecondaryWindowMessage);
  yahoo.LocalReady();
}
```

The window.external property creates a reference to the main window Tab plug-in, **`SetEventHandler`** creates event handlers for secondary window events, and **`LocalReady`** lets the host know that it is up and ready. Communication to this plug-in host cannot happen until you call **`LocalReady`**. To open a secondary window, you must add event handlers so that you will be notified when the secondary window is opened and closed, and send messages back and forth between the secondary window and the main plug-in window.

Opening and Communicating with a Secondary Window

The Tab plug-in main window can open, send messages to, and close a secondary window. This can be done with the following function calls:

```
yhost.SecondaryWindowOpen(url);
yhost.SendMessageToSecondaryWindow(msg);
yhost.SecondaryWindowClose();
```

The secondary window can then call window.external to get a reference to the ContentTabSecondaryWindowPluginHost object, and set up an event handler to catch messages sent to it from the Tab main window.

```
var yahoo = window.external;
yahoo.SetEventHandler("MainPluginWindowMessage",
  onMainPluginWindowMessage);
yahoo.LocalReady();
```

The secondary window can send messages back to the main window, and close itself, as follows:

```
yhost.SendMessageToMainPluginWindow(msg);
yhost.SecondaryWindowCloseMyself();
```

Creating a "Hello World" Conversation Plug-in

This "Hello World" Conversation plug-in translates instant messages for the user.

Initializing the Plug-in

To begin, add initialization code to create the plug-in host, set event handlers and prepare it for input, as follows:

```
function on_load ()
{
  yhost = window.external;
  // If this is the user starting the conversation, get their language.
  if (host.Inviter) {
    lang = "nl_en";
    var text_input = document.getElementById('lang_choice');
    text_input.value = lang;
   }
  yhost.SetEventHandler ('IMReceived', on_IMReceived);
  yhost.SetEventHandler ('IMSent', on_IMSent);
  yhost.LocalReady(); // plug-in is up and ready
}
```

Now the plug-in is ready to communicate with another user.

Sending an Instant Message to a Friend

The plug-in can send instant messages to a friend using the **`SendIM`** method. Here, the plug-in calls **`SendIM`** to send an instant message, and then enters the sent text into the sender's conversation window's input window and history window.

```
function Send_Msg(msg)
{
  yhost.SendIM(msg);
  msg =  "<b>" + yhost.window.CurrentIdentity.Name + ": </b>" + msg;
  yhost.InsertInHistoryWindow(msg);
  yhost.InputWindowText = msg;
};
```

Event Handling of Sent and Received Instant Messages

You can set up event handlers to fire when an instant message is sent or received. The “IMReceived” event handler fires when a plug-in receives an instant message from another plug-in instance. The following code sends the incoming instant message to a translation website to be translated and puts the translated message into the conversation history window.

```
function on_IMReceived(msg) {
  var url_in =
"http://babelfish.altavista.com/tr?lp="+lang+"&trtext="+encodeURIComponen
t(msg);
  var translated = host.SendHTTPRequest(msg, url_in);
  yhost.InsertInHistoryWindow(translated);
}
```

Creating a “Hello World” ActiveX Plug-in

Tab ActiveX plug-ins reside in the content tab, while Conversation ActiveX plug-ins are embedded in the conversation window. Both types are hosted by Internet Explorer—a surrogate host for Messenger.

To create an ActiveX plug-in, you need to establish a connection between the plug-in and the IE web browser by calling **`IOleObjectImpl::SetClientSite`**. This tells the plug-in about its client site in the host. Next, you need to call **`IOleClientSite`** so the plug-in can call **`QueryInterface`** to access the Messenger Plug-in interfaces.

To create ActiveX Messenger Plug-in,

1. Get a pointer to the document, and then call **`QueryInterface`** for the **`IPluginHost`** interface.

```
IServiceProvider* pSrvProv = NULL;
pClientSite->QueryInterface(IID_IServiceProvider, (void**)&pSrvProv);

IWebBrowser2* pWebBrowser = NULL;
pSrvProv->QueryService( SID_SWebBrowserApp, IID_IWebBrowser2,
(void**)&pWebBrowser );

IHTMLDocument2* pDocument = NULL;
pWebBrowser->get_Document( (IDispatch**)&pDocument );

IHTMLWindow2* pWindow = NULL;
```

```
pDocument->get_parentWindow( &pWindow );

IDispatch* pExternal = NULL;
pWindow->get_external( &pExternal );
```

2. Call **`QueryInterface`** to start using the Messenger Plug-in APIs.

```
IPluginHost* pHost = NULL;
pExternal.QueryInterface( &pHost );

_variant_t varMessenger;
pHost->get_Messenger( &varMessenger.GetVARIANT() );

CheckVariantIsIUnknown( varMessenger );

IMessenger2* pMessenger2 = NULL;
varMessenger.punkVal->QueryInterface( IID_IMessenger2,
(void**)&pMessenger2 );

ICreatedWindows* spCreatedWindows = NULL;
pMessenger2.QueryInterface( &pCreatedWindows );

_variant_t varMainWindow;
pCreatedWindows->get_MainWindow( &varMainWindow.GetVARIANT());
CheckVariantIsIUnknown( varMainWindow );

IContactListUI* spContactListUI = NULL;
varMainWindow.punkVal->QueryInterface( IID_IContactListUI,
(void**)&pContactListUI );

IContentTabPluginServices* spServices = NULL;
pHost.QueryInterface( &pServices );

_variant_t varFileTransferMgr;
pServices->get_FileTransfer( &varFileTransferMgr );

CheckVariantIsIUnknown( varFileTransferMgr );

IFileTransferManager* spFileTransferManager = NULL;
varFileTransferMgr.punkVal->QueryInterface( IID_IFileTransferManager,
(void**)&pFileTransferManager);
```

Developer Guide

This section describes the architecture and usage of the Messenger Plug-in SDK objects and interfaces.

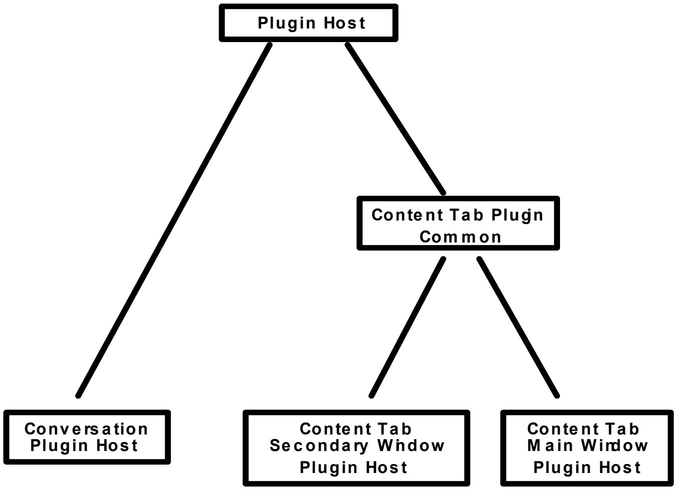
Architecture

The SDK APIs are organized according to their functionality. One set of objects provide functionality to create the plug-in hosts, while another set of objects provide common functionality used by the plug-ins. A third set are collection objects.

- Host Objects
- Common Objects
- Collection Objects

Host Objects

Host objects are the building blocks of the plug-in host. Objects are either abstract, or are instantiated as a plug-in host. The hierarchy of the host related objects is shown in the following diagram. The nodes of the tree represent objects that you instantiate as plug-ins, except that the Content Tab Secondary Window can only be instantiated from a secondary window launched by a Tab plug-in when it calls the **`SecondaryWindowOpen`** method.


Plugin Host
Content Tab Plugn Common
Conversation Plugin Host
Content Tab Secondary Window Plugin Host
Content Tab Main Window Plugin Host

Each tertiary object in the diagram is created by calling window.external. The type of object that is created depends on the context. When window.external is initially called in the HTML file, the type of plug-in created is determined by the Plugin-type setting in the manifest file with specifies Conversation (ConversationPluginHost) or Tab (ContentTabMainWindowPluginHost). However, if a Tab plug-in opens a secondary window, calling window.external from the HTML creates a ContentTabSecondaryWindow object.

Common Objects

Common objects are objects provide ancillary functionary to plug-ins and represent such entities as the user's Messenger buddy list, address book, a users status, an instant message, the main messenger window, and the conversation window. The complete list of common objects is as follows:

- Messenger
- MainWindow

- ConversationWindow
- Status
- Group
- Identity
- Me
- IMMessage
- Friend
- Plugin
- Error

Collection Objects

The Messenger Plug-in SDK contains collection objects that are standard COM enumeration interfaces. Each item in the collections are of type VARIANT.

Object	Description
IFriends	Each item is a Friend object.
IAddressBookEntries	Each item is AddressBookEntry object.
IGroups	Each item is Group object.
IIdentities	Each item is Identity object.
IWindows	Each item is an object supporting at least IWindow interface. It can support more interfaces depending on type. See ConversationWindow.
IPlugins	Item is a Plugin object.
IFileTransfers	Item is FileTransfer object.

Usage

This section describes basic functionality of the SDK. Topics include:

- LocalReady
- Object Lifetime
- Catching Errors
- Handling Events
- File Transfers

LocalReady

To properly initialize the plug-in window created by calling window.external, you must call LocalReady.

```
yahoo = window.external;
yahoo.localready();
```

LocalReady notifies other plug-in instances that this plug-in host has been created and is ready for input. For instance, when a secondary window is launched, it must call `LocalReady` to let the main plug-in window know that it exists and is ready to communicate. Likewise, Conversation plug-ins call LocalReady to notify the other user's plug-in instance that it exists and can communicate. Until `LocalReady` is called, local host will not receive any event notifications.

The following tables list the methods, events and errors that change or are affected by the status of a local host object.

Method	**Interface**
LocalReady	IPluginHost

Event	**Interface**
RemoteReady	_IEventsConversationPluginServices
RemoteUnloaded	_IEventsConversationPluginServices
PluginStatus	_EventsContentTabPluginServices
SecondaryWindowReady	_IEventsMainPluginWindow

Error Message	**Method**
“RemotePartyNotReady”	IConversationPluginServices.SendPluginMessage
“SecondaryWindowNotReady”	IMainPluginWindow.SendMessageToSecondaryWindow
"MainPluginWindowNotReady"	ISecondaryWindow.SendMessageToMainPluginWindow

Conversation Plug-in

To demonstrate conversation plug-in behavior, suppose we have a conversation plug-in with a user A and a user B.

1. After user A's plug-in instance calls `LocalReady`, a “RemoteReady” event is sent to the user B plug-in instance. B only receives the event if it has called `LocalReady`.

2. When user A closes the plug-in, and B has called LocalReady, then B receives the event “RemoteUnloaded”.
3. User B can send a plug-in message to user A only after it has received a **`RemoteReady`** event from user A. Otherwise, **`SendPluginMessage`** returns the error “RemotePartyNotReady”.

Tab Plug-in

To demonstrate Tab plug-in behavior when a secondary window is opened:

- When a secondary window calls **`LocalReady`**, a “SecondaryWindowReady” event is sent to the main plug-in window.
- Before the main plug-in window can call **`SendMessageToSecondaryWindow,`** it must receive a **`SecondaryWindowReady`** from the secondary plug-in window. Otherwise, the error “SecondaryWindowNotReady” is returned.
- If the main plug-in window did not call LocalReady when it was created, and the secondary window calls **`SendMessageToMainPluginWindow`**, the error "MainPluginWindowNotReady" is returned.

Object Lifetime

Each time you call a property or method that returns an object, it will return a different object. Therefore, if you install event handlers for this particular object, you should hold a reference to it. Otherwise, this object will be destroyed and installed event handlers (C++: sinks) will be released. That means that you will not get the events without holding a reference to the object that sends them out.

Catching Errors

`IMessenger2.LastError` property returns the last occurring error. See individual function descriptions for possible errors.

For plug-ins, **`window.external.LastError`** provides the same information as **`window.external.Messenger.LastError`**.

Handling Events

Events are handled in plug-ins by creating an event handler for each event you want to capture. To create an event handler, call the SetEventHandler function and specify the following two parameters:

Parameter	Description
Event Name	The name of the event function
Event Handler	The script function handling the event. This function prototype should be same as the prototype of event function as defined in Messenger COM interface type library. Specifying null removes the previously set event handler.

The following example, creates an event handler to process then a friend has changed in the buddy list. Please note that the **`On_FriendChanged`** function matches the prototype of **`_IEventsContactList.FriendChanged`**:

```
yhost.SetEventHandler("FriendChanged", On_FriendChanged);

function On_FriendChanged(buddy)
{
  document.FriendList.innerHTML = document.FriendList.innerHTML + "Event
    JScript: Friend changed: " + ", " +
    buddy.Name + "(" +
    buddy.UniqueSessionID + ")" + "(" +
    buddy.StatusMain + ": " +
    buddy.StatusMessage + ", " +
    buddy.CustomMessage + ")" +
}
```

COM Clients

COM clients can handle events exposed by Messenger COM interface objects by implementing an event sink with a reference to the source object.

Because all event interfaces are dispinterfaces only, you will need the dispatch ID of each event method to define your sink. The ID for each method can be found in the TLB file supplied with the SDK.

File Transfers

A plug-in sends a file to another plug-in using the FileTransferManager object, which manages the file transfer process by starting or stopping outgoing transfers, accepting or declining incoming files, and firing notifications for file transfer events. When the FileTransferManager object starts a file transfer, it creates a FileTransfer object, which contains all current information about the transfer.

Sending a File

In the following example, a plug-in calls the FileTransfer method to get a reference to a FileTransferManager object. Then, in the StartNewFileTransfer function, the FileTransferSend method sends the file to the specified friend.

```
yhost = window.external;
yhost.ContentTabTitle = "Tab Plug-in";
ftm = yhost.FileTransfer;

function StartNewFileTransfer(path, strFriend){
  var friend;
  friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft = ftm.FileTransferSend(path, friend);
  var pref = "File Transfer: ";
  append_log_msg(pref + ft.ID);
  append_log_msg(pref + ft.Path);
  append_log_msg(pref + ft.Size);
  append_log_msg(pref + ft.Transferred);
  append_log_msg(pref + ft.Status);
  append_log_msg(pref + ft.Direction);
}

StartNewFileTransfer(FilePath, MyFriend);
```

The **`strFriend`** parameter can be null, but **`_IEventsFileTransferManager`** events return a Friend object that always represents the other person in the conversation.

The plug-in is notified of the status of the file transfer by the following **`_IEventsFileTransferManager`** events:

FileTransferProgress	Fired when a file is in the process of being sent.
FileTransferDone	Fired when the file transfer is complete.
FileTransferAccepted	Fired when the outgoing file transfer is accepted by the other side.

FileTransferDeclined	Fired when the outgoing file transfer is declined by the other side.
FileTransferRemotePartyCancelled	Fired when accepted file transfer is cancelled by the other side.

Receiving a File

When a plug-in receives a file from another plug-in instance, the event “FileTransferIncoming” is fired. If an event handler has been created to handle his event, it can choose to accept or decline the file. In the following example, the event handler on_FileTransferIncoming accepts incoming files and saves them to “c:/temp/file123.jpg”.

The following example creates an event handler to accept incoming file transfers.

```
ftm.SetEventHandler('FileTransferIncoming', on_FileTransferIncoming);

function on_FileTransferIncoming(ft, friend)
{
  if (cb_IEventsFileTransfer.checked)
    append_log_msg("FileTransferIncoming, ID: " + ft.ID + ", From Friend:
    " + friend.ID);

  ftm.FileTransferAccept(ft, "c:/temp/file123.jpg");
}
```

ymsgr Protocol

The **`ymsgr`** protocol allows you to launch a plugin from a URL link.

Syntax

```
ymsgr:getplugin?type=plug-in type&id=plug-in id&name=plug-in
name[&locale=plug-in locale][&version=plug-in version][&message=message
text]
```

For example, the following link opens the calender plug-in.

```
<a href="ymsgr:getplugin?type=4&id={5DFB0298-28C0-6657-17EB-
16D8BA683817}&name=Y!%20Calendar">Calendar</a>
```

Parameters

Parameter	**Description**
type	Plugin-in type. 1: conversation window plug-in. 2: tab plug-in
id	Plug-in ID.
name	Plug-in name.
locale	Plug-in locale. Default value is “en-us”.
version	Plug-in version. Default value is “1.0.0”.
message	Message text received by the launched plug-in, even if it was already running.

Objects

AddressBook

The **`AddressBook`** object corresponds to the user's Address Book. This object is only available as a property of the plug-in objects ContentTabMainWindowPluginHost, ConversationPluginHost, and ContentTabSecondaryWindowPluginHost.

Access to the AddressBook object is only available if the AddressBook entry in the manifest file is set to True.

`AddressBook` implements the following interfaces:

- IAddressBook
- _IEventsAddressBook

Properties

Name	Description
Entries	Returns a collection of AddressBookEntry objects.

Methods

Name	Description
EntryFromFriend	Retrieves the address book entry for a friend.

Events

Name	Description
ABEntryRemoved	Fired when an address book entry is removed from the user's address book.
ABEntryUpdated	Fired when an address book entry is updated in the user's address book.
ABUpdated	Fired when the whole address book is updated.

AddressBookEntry

The **`AddressBookEntry`** object provides access to the details stored for individual entries in the user's address book. This object is only available to plug-ins. The **`AddressBookEntry`** object implements the following interface:

- IAddressBookEntry

Properties

Name	Description
Friend	**Returns a** Friend object representing the friend in the buddy list associated with this address book entry.
FirstName	`String`. First name.
MiddleName	`String`. Middle name.
LastName	`String`. Last name.
NickName	`String`. Nick name.
JobTitle	`String`. Job title.
PhoneNumberHome	`String`. Home phone number.
PhoneNumberWork	`String`. Work phone number.
PhoneNumberMobile	`String`. Mobile phone number.
PhoneNumberPager	`String`. Pager phone number.
PhoneNumberFax	`String`. Fax phone number.
PhoneNumberOther	`String`. Other phone number.
EmailAddress	`String`. Email address.
EmailAddress1	`String`. Second email address.
EmailAddress2	`String`. Other email address.
HomeAddressStreet	`String`. Home street address.
HomeAddressCity	`String`. City of home address
HomeAddressState	`String`. State of home address.
HomeAddressPostalCode	`String`. Postal code of home address.
HomeAddressPostalCountry	`String`. Country of home address.
Company	`String`. Company.
WorkAddressStreet	`String`. Work street address.
WorkAddressCity	`String`. City of the work address.
WorkAddressState	`String`. State of the work address.
WorkAddressPostalCode	`String`. Postal code of the work address.
WorkAddressPostalCountry	`String`. Country of the work address.
WorkPhone	`String`. Work phone number.
WorkWebSite	`String`. Work web site.
BirthdayDay	`Numeric`. Day of birth.

AddressBookEntry

Name	Description
BirthdayMonth	**`Numeric`**. Month of birth.
BirthdayYear	**`Numeric`**. Year of birth.
AnniversaryDay	**`Numeric`**. Day of anniversary.
AnniversaryMonth	**`Numeric`**. Month of anniversary.
AnniversaryYear	**`Numeric`**. Year of anniversary.
Notes	**`String`**. Notes.
NotesCustom1	**`String`**. Custom Notes1.
NotesCustom2	**`String`**. Custom Notes2.
NotesCustom3	**`String`**. Custom Notes3
NotesCustom4	**`String`**. Custom Notes4
WebSite	**`String`**. Website.
UniqueSessionID	**`String`**. Unique session ID.

ContentTabMainWindowPluginHost

The ContentTabMainPluginWindowPluginHost object represents the main window of a Tab plug-in. Call window.external to get an instance of this object, which is hosted by Messenger. For example, yhost will be an instance of a Tab plug-in object:

```
yhost = window.external;
yhost.localReady();
```

From the Tab plug-in, you can also launch a secondary window—**`ContentTabSecondaryWindowPluginHost`** object.

Opening a URL with pop-ups in the main plug-in window or a secondary window can cause a scripting or debug error to be shown to the user. Therefore, it is recommended that you do not open URL with pop-ups in the main plugin window or secondary windows.

The **`ContentTabMainWindowPluginHost`** object implements the following interfaces.

- IMainPluginWindow
- IPluginHost
- IPluginStorage
- IToast
- IColors
- IPlugin
- IContentTabPluginServices
- _IEventsPluginHost
- _IEventsContentTabPluginServices
- _IEventsToast
- _IEventsColors
- _IEventsMainPluginWindow

The properties, methods and events that are made available to this object by Messenger are listed in the following tables.

ContentTabMainWindowPluginHost

Properties

Name	Description
Messenger	Returns a Messenger object.
AddressBook	Returns an AddressBook object that represents the user’s address book. Returns null if access is restricted.
Cookies	Messenger cookies. Returns the messenger cookies. Returns null if plug-in was not loaded from the Yahoo! domain.
LastError	**`Error`** object. Returns the last error that occurred.
PluginID	**`String`**. Returns the plug-in ID.
PluginName	**`String`**. Returns the plug-in name.
PluginVersion	**`String`**. Returns the plug-in version.
PluginLocale	**`String`**. Returns the locale of the plug-in.
ContentTabTitle	The title that displays at the top of a Tab plug-in.
MainPluginWindow	**`Boolean`**. If True, this is the main plug-in window. If False, this is the main plug-in window.
FileTransfer	Sends a plug-in message to a friend.
MainWindowState	**`String`**. Returns the state of the main window of a Tab plug-in. Can be one of the following: Docked, Embedded, Minimized, and Overflow.

Methods

Name	Description
SendHTTPRequest	Returns the content of the provided URL.
OpenURL	Opens the provided URL in a separate window.
LocalReady	Tells the host that the plug-in is ready.
ShowFileBrowserDialog	Launches the Show File Browser Dialog box.
ShowBuddyPicker	Launches the Buddy Picker dialog box.
StorageWrite	Writes a key and its value to storage.
StorageRead	Reads the value for a given key.
StorageRemove	Removes a key and its value from storage.

Name	Description
ToastShow	Pops up the toast window.
ToastClose	Closes the toast window.
SendPluginMessage	Sends a plug-in message to a friend.
SetPluginStatus	Sets the plug-in status.
SecondaryWindowOpen	Opens the secondary window.
SecondaryWindowOpenWithDimensions	Opens the secondary window with the provided dimensions.
SecondaryWindowClose	Closes the secondary window.
SendMessageToSecondaryWindow	Sends a message to the secondary window.

Events

Name	Description
ToastLinkClicked	Fired when the user clicks a link in the toast window.
Unloaded	Fired when the host is about to unload the plug-in.
HTTPRequestCompleted	Fired when the URL content has completed downloading.
HTTPRequestError	Fired when the URL download fails.
PluginMessage	Fired when the message is received from the friend's instance of this plug-in. (is this correct?)
PluginStatus	Fired when the user clicks the status link in the buddy list.
SecondaryWindowMessage	Fired when secondary window sends a message.
SecondaryWindowReady	Fired when secondary window is opened and ready.
SecondaryWindowClosed	Fired when secondary window is closed.
MainWindowStateChanged	Fired when main plug-in window is popped out.

ContentTabPluginCommon

The **`ContentTabPluginCommon`** object provides functionality common to both types of Tab plugins: **`ContentTabSecondaryWindowPluginHost`** and **`ContentTabMainPluginHost`**. As such, you do not create an instance of this object, but of one of those two that inherit from this object.

This object is only available to plugins. The **`ContentTabPluginCommon`** object implements the following interfaces.

- IPluginHost
- IPluginStorage
- IToast
- IColors
- IPlugin
- IContentTabPluginServices
- _IEventsPluginHost
- _IEventsContentTabPluginServices
- _IEventsToast
- _IEventsColors

Properties

Name	Description
Messenger	Returns a Messenger object.
AddressBook	AddressBook object. Returns the address book. Returns null if access is restricted.
Cookies	Messenger cookies. Returns the messenger cookies. Returns null if plug-in was not loaded from the Yahoo! domain.
LastError	`Error`. Returns the last error that occured.
PluginID	`String`. Returns the plug-in ID.
PluginName	`String`. Returns the plug-in name.
PluginVersion	`String`. Returns the plug-in version.
PluginLocale	`String`. Returns the locale of the plug-in.
ContentTabTitle	The title that displays at the top of a Tab plug-in.
MainPluginWindow	`Boolean`. If True, this is the main plug-in window. If False, this is the main plug-in window.
FileTransfer	Sends a plug-in message to a friend.

Methods

Name	Description
SendHTTPRequest	Returns the content of the provided URL.
OpenURL	Opens the provided URL in a separate window.
LocalReady	Tells the host that the plug-in is ready.
ShowFileBrowserDialog	Lauches the Show File Browser Dialog box.
ShowBuddyPicker	Lauches the Buddy Picker dialog box.
StorageWrite	Writes a key and its value to storage.
StorageRead	Reads the value for a given key.
StorageRemove	Removes a key and its value from storage.
ToastShow	Pops up the toast window.
ToastClose	Closes the toast window.
SendPluginMessage	Sends a plug-in message to a friend.
SetPluginStatus	Sets the plug-in status.

Events

Name	Description
ToastLinkClicked	Fired when the user clicks a link in the toast window.
Unloaded	Fired when the host is about to unload the plug-in.
HTTPRequestCompleted	Fired when the URL content has completed downloading.
HTTPRequestError	Fired when the URL download fails.
PluginMessage	Fired when the message is received from the friend's instance of this plug-in.
PluginStatus	Fired when the user clicks the status link in the buddy list.
MIMEMessage	

ContentTabSecondaryWindowPluginHost

This object represents a secondary window launched from a Tab plug-in, and can only be created by a Tab plug-in. To launch a secondary window from the Tab plug-in, call **`SecondaryWindowOpen`** method. For example, yhost is a Tab plug-in, which then opens a secondary window.

```
yhost = window.external;
yhost.localReady();
yhost.SecondaryWindowOpen(url);
```

The code in the secondary window can then call window.external to get an instance of itself, as follows:

```
ysecondaryhost = window.external
ysecondaryhost.localReady();
```

The **`ContentTabSecondaryWindowPluginHost`** object implements the following interfaces.

- IPluginHost
- IPluginStorage
- IToast
- IColors
- IPlugin
- IContentTabPluginServices
- ISecondaryWindow
- _IEventsPluginHost
- _IEventsContentTabPluginServices
- _IEventsToast
- _IEventsColors
- _IEventsSecondaryWindow

The properties, methods and events that are made available to this object by Messenger are listed in the following tables.

Properties

Name	Description
Messenger	Returns a Messenger object with IDispatch interface.
AddressBook	Returns an AddressBook object containing the address book.Returns null if access is restricted.
Cookies	Messenger cookies. Returns the messenger cookies. Returns null if plug-in was not loaded from the Yahoo! domain.
LastError	Returns **`Error`** object containing the last error that occured.
PluginID	**`String`**. Returns the plug-in ID.
PluginName	**`String`**. Returns the plug-in name.
PluginVersion	**`String`**. Returns the plug-in version.
PluginLocale	**`String`**. Returns the locale of the plug-in.
ContentTabTitle	The title that displays at the top of a Tab plug-in.
MainPluginWindow	**`Boolean`**. If True, this is the main plug-in window. If False, this is the main plug-in window.
FileTransfer	Sends a plug-in message to a friend.

Methods

Name	Description
SendHTTPRequest	Returns the content of the provided URL.
OpenURL	Opens the provided URL in a separate window.
LocalReady	Tells the host that the plug-in is ready.
ShowFileBrowserDialog	Lauches the Show File Browser dialog box.
ShowBuddyPicker	Lauches the Buddy Picker dialog box.
StorageWrite	Writes a key and its value to storage.
StorageRead	Reads the value for a given key.
StorageRemove	Removes a key and its value from storage.
ToastShow	Pops up the toast window.
ToastClose	Closes the toast window.
SendPluginMessage	Sends a plug-in message to a friend.

Name	Description
SecondaryWindowCloseMyself	Secondary window closes itself.
SendMessageToMainPluginWindow	Secondary window sends a message to main plug-in window.

Events

Name	Description
ToastLinkClicked	Fired when the user clicks a link in the toast window.
Unloaded	Fired when the host is about to unload the plug-in.
HTTPRequestCompleted	Fired when the URL content has completed downloading.
HTTPRequestError	Fired when the URL download fails.
PluginMessage	Fired when the message is received from the friend's instance of this plug-in. (is this correct?)
MainPluginWindowMessage	Fired when Tab main plug-in window sends a message to the secondary window.

ConversationWindow

The `ConversationWindow` object represents the Conversation window launched from the content tab when a user wants to start a conversation with another user. The plug-in can send instant messages only if the Sendim entry in the manifest file is set to True.

`ConversationWindow` supports the following interfaces:

- IWindow
- IParticipants
- _IEventsWindow

Properties

Name	Description
CurrentIdentity	Returns an Identity object.
Friends	Returns Collection of Friend objects containing the friends participating in the conversation.
WinOpen	`Boolean`. Returns true if the window is opened or closed.
WinVisible	`Boolean`. Returns ture if the window is visible.
WinX	`Numeric`. Returns the top left corner horizontal position in pixels.
WinY	`Numeric`. Returns the top left corner vertical position in pixels.
WinWidth	`Numeric`. Returns the width of the window in pixels.
WinHeight	`Numeric`. Returns the height of the window in pixels.
WinHWND	`Numeric`. Returns a window handler for the window.
WinTitle	`String`. Returns the window title.
WinForeground	`Boolean`. Returns true if the window is in the foreground.
WinActive	`Boolean`. Returns true if the window has mouse of keyboard focus.
WinMinimized	`Boolean`. Returns true if the window is minimized.
WinMaximized	`Boolean`. Returns true if the window is maximized.
SendIM	Send the instant message.

Events

Name	Description
Moved	Fired when window is moved.
Resized	Fired when window is resized.
Maximized	Fired when window is maximized.
Minimized	Fired when window is minimized.
Activated	Fired when the window loses or gains input focus.
Closed	Fired when window is closed.

ConversationPluginHost

The **`ConversationPluginHost`** object represents an instance of a Conversation plug-in, launched from the user's conversation window. This object is only available to plugins. The **`ConversationPluginHost`** object implements the following interfaces:

- IPluginHost
- IPluginStorage
- IToast
- IColors
- IPlugin
- IConversationWindow
- IConversationPluginServices
- _IEventsPluginHost
- _IEventsToast
- _IEventsColors
- _IEventsConversationPluginServices
- _IEventsConversationWindow

Properties

Name	Description
Messenger	Returns an `IDispatch` Messenger object.
AddressBook	Returns an AddressBook object containing the user's address book. Returns null if access is restricted.
Cookies	Messenger cookies. Returns the messenger cookies. Returns null if plug-in was not loaded from the Yahoo! domain.
LastError	Returns `Error` object containing the last error that occured.
PluginID	`String`. Returns the plug-in ID.
PluginName	`String`. Returns the plug-in name.
PluginVersion	`String`. Returns the plug-in version.
PluginLocale	`String`. Returns the locale of the plug-in.
InputWindowText	`String`. Input text into the window.
Window	`ConversationWindow` object.
Inviter	`Boolean`. If True, this instance is the inviter; if False, this instance is the invitee.
FileTransfer	`Returns a FileTransferManager` object.

Methods

Name	Description
SendHTTPRequest	Returns the content of the provided URL.
OpenURL	Opens the provided URL in a separate window.
LocalReady	Tells the host that the plug-in is ready.
ShowFileBrowserDialog	Lauches the Show File Browser Dialog box.
ShowBuddyPicker	Lauches the Buddy Picker dialog box.
StorageWrite	Writes a key and its value to storage.
StorageRead	Reads the value for a given key.
StorageRemove	Removes a key and its value from storage.
InsertInHistoryWindow	Inserts text into the history window. Plain text is supported.
SendPluginMessage	Sends a message to other side's instance of this plug-in participating in same conversation.
SendIM	Sends an instant message to another user.

ToastShow	Pops up the toast window.
ToastClose	Closes the toast window.

Events

Name	Description
ToastLinkClicked	Fired when the user clicks a link in the toast window.
Unloaded	Fired when the host is about to unload the plug-in.
HTTPRequestCompleted	Fired when the URL content has completed downloading.
HTTPRequestError	Fired when the URL download fails.
PluginMessage	Fired when a new plug-in message arrives from the other plug-in instance.
RemoteReady	Fired when the remote plug-in is loaded an ready.
RemoteUnloaded	Fired when the remote plug-in is unloaded and no longer available.
UserIsTyping	Fired when the user types within the input window.
IMReceived	Fired when a new instant message is received.
IMSent	Fired when an instant message is sent.
IncomingIM	Fired when an incoming instant message has been received but not yet processed. The plug-in can modify the message using the passed parameter.

Error

The Error object stores information about a specific error and passes the error ID to certain event notifications. The **`Error`** object implements the following interfaces:

- IError

Properties

Name	Description
ErrorID	**`String`**. Returns the error ID. Returns “NoError” if no error has occured.

FileTransfer

The FileTransfer object contains all details about the current file transfer. To start a file transfer and create a file transfer object, you will call the**FileTransferManager’s** **FileTransferSend** method. For example:

```
function StartNewFileTransfer(path, strFriend){
  var friend;
  friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft;
  ft = ftm.FileTransferSend(path, friend);
  var pref = "File Transfer: ";
  append_log_msg(pref + ft.ID);
  append_log_msg(pref + ft.Path);
  append_log_msg(pref + ft.Size);
  append_log_msg(pref + ft.Transferred);
  append_log_msg(pref + ft.Status);
  append_log_msg(pref + ft.Direction);
}
```

This object is only available to plugins. File transfer is only available to a plug-in if the **Filesend** entry in the manifest file is set to True.

FileTransfer implements following interface.

- IFileTransfer

Properties

Name	Description
ID	**`String`**. U nique Id for the file transfer.
Path	**`String`**. Local file path.
Size	**`Numeric`**. File size in bytes.
Transferred	Numeric. The size of the information in bytes that was successfully transferred.
Status	**`String`**. File transfer status. Possible values are: “NotStarted”: file transfer has not started. “InProgress” file transfer is in progress. “Cancelled”: file transfer was cancelled by user. “Transferred”: file transfer has completed. “Error”: file transfer was unable to complete.
Direction	**`String`**. Direction of the file transfer: Possible values: “Incoming”: file is “Outgoing”

FileTransferManager

The File Transfer Manager object manages the file transfer process by starting or stopping outgoing transfers, accepting or declining incoming files, and firing notifications for file transfer events. When the file transfer manager starts a file transfer, it creates a File Transfer object, which contains all current information about the transfer.

In the following example, a Tab plug-in calls the FileTransfer method to get a reference to a FileTransferManager object. The StartNewFileTransfer function calls the FileTransferManager's FileTransferSend method to send the file to a specified friend.

```
yhost = window.external;
yhost.ContentTabTitle = "Tab Plug-in";
ftm = yhost.FileTransfer;

function StartNewFileTransfer(path, strFriend){
  var friend;
  friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft = ftm.FileTransferSend(path, friend);
  var pref = "File Transfer: ";
  append_log_msg(pref + ft.ID);
  append_log_msg(pref + ft.Path);
  append_log_msg(pref + ft.Size);
  append_log_msg(pref + ft.Transferred);
  append_log_msg(pref + ft.Status);
  append_log_msg(pref + ft.Direction);
}

StartNewFileTransfer(FilePath, MyFriend);
```

The **`strFriend`** parameter can be null, but the event _IEventsFileTransferManager returns a Friend object that always represents the other person in the conversation.

This object is only available to plugins. The **`FileTransferManager`** object implements the following interfaces.

- IFileTransferManager
- _IEventsFileTransferManager

Methods

Name	Description
FileTransferSend	Creates the outgoing FileTransfer object.
FileTransferAccept	Accepts the incoming file transfer.
FileTransferDecline	Declines the incoming file transfer.
FileTransferCancel	Cancels the file transfer.

Events

Name	Description
FileTransferProgress	Fired when a file is in the process of being sent.
FileTransferDone	Fired when the file transfer is complete.
FileTransferIncoming	Fired when a new file has been received from another user.
FileTransferAccepted	Fired when the outgoing file transfer is accepted by the other side.
FileTransferDeclined	Fired when the outgoing file transfer is declined by the other side.
FileTransferRemotePartyCancelled	Fired when accepted file transfer is cancelled by the other side.

Friend

The Friend object represents a friend that a user may communicate with in some way. The object contains information that is stored by Yahoo! about the friend, such as their current status, name, the plugins that they are running, and whether they are online.

The Friend object implements the following interface:

- IFriend

Properties

Name	Description
Groups	Returns Collection of Group objects containing the groups that the friend belongs to.
ID	`String`. Returns the friend's ID.
Status	Returns a Status object containing the friend's status information.
Image	`String`. Returns the file name of the friend's image.
UniqueSessionID	`String`. Returns a unique session ID for the object that can be used to verify the object is the correct one.
FirstName	`String`. Returns the first name of the friend.
LastName	`String`. Returns the last name of the friend.
PeerToPeer	`Boolean`. Returns True if peer to peer connection is available.
Plugins	Returns collection of Plugin objects containing all the plugins that the friend is running.
OnContactList	`Boolean`. Returns True if the friend is in the buddy list.

Group

The Group object represents a group in the buddy list. The Group object supports the following interfaces:

- IGroup

Properties

Name	Description
Friends	Returns collection of Friend objects containing a list of friends that belong to the group.
Name	`String`. Returns the name of the group.
UniqueSessionID	`String`. Returns the unique session ID of the Group object.

IMMessage

The **`IMMessage`** object contains the text of the user's message entered into the conversation window and sent to another user. The **`IMMessage`** object implements the following interface:

- IIMMessage

Properties

Name	Description
Text	**`String`**. The text sent in an instant message.

Me

The `Me` object implements the following interface:

- IMe

Properties

Name	Description
Identities	Returns a collection containing Identity objects of all identies associated with the user.
LoginIdentity	Returns Identity object containing the ID used to log the user in for this session.
Status	Returns Status object containing status information displayed to other users.
Image	**`String`**. Returns the file name of user’s image.
FirstName	**`String`**. Returns the first name of user.
LastName	**`String`**. Returns the last name of user.

Messenger

The **`Messenger`** object provides functionality to manage the buddy list, start IM, voice, text messaging, and web cam sessions with other users, and send plugins and files to other users. The **`Messenger`** object implements the following interfaces:

- IMessenger2
- IContactList
- IMessengerActions
- ICreatedWindows
- _IEventsMessenger2
- _IEventsContactList
- _IEventsWindowCreation

Properties

Name	Description
Me	References a Me object for the user currently logged in to Messenger
LastError	Error object. References an Error object containing information for the last occurring error.
AddressBook	Returns an AddressBook object.
Friends	Returns a collection of Friend objects.
Groups	Returns a collection of Group objects.
FriendsOnline	Returns a collection of Friend objects of for friends that are online.
MainWindow	Returns a MainWindow object.
Windows	Returns the collection of objects implementing the IWindow interface. Can be MainWindow and ConversationWindow objects.

Methods

Name	Description
IMConversation	Start an IM conversation
FileTransfer	Start a file transfer.
`WebCam`	Start a web cam session.
Conference	Start a conference.
PhotoSharing	Start a photo sharing session.
SendMyContactDetails	Launch the send my contact details UI.
PlayGame	Start a game session.
SendSMSToPhone	Start the process of sending SMS to phone.
SendSMSToFriend	Start the process of sending SMS to a friend.
Login	Login to Messenger. Only available to external applications; not available from within the plug-in.
LoginUser	Login to Messenger with the specified user ID and password. Only available to external applications; not available from within the plug-in.
Logout	Logout from Messenger. Only available to external applications; not available from within the plug-in.

Name	Description
GetFriendFromID	Returns a Friend object for the given friend ID.
GetGroupFromName	Returns the Group object given the group name.
AddFriend	Adds a friend to the user's buddy list in the specified group
DeleteFriend	Deletes a friend from the user's buddy list.

Events

Name	Description
LoggedIn	Fired when Messenger logs in to the network.
LoggedOut	Fired when Messenger logs out of the network.
Alert	Fired when an alert is received.
FriendAddedToGroup	Fired when new user is added to a group in the buddy list.
FriendDeletedFromGroup	Fired when friend is deleted from group in the buddy list.
FriendChanged	Fired when a friend in the buddy list's information changes.
GroupAdded	Fired when a new group is added to buddy list.
GroupDeleted	Fired when a group is deleted from the buddy list.
GroupRenamed	Fired when a group in the buddy list is renamed.
PluginLoaded	Fired when the plug-in is loaded.
PluginUnloaded	Fired when the plug-in is unloaded.
WindowCreated	Fired when a new window is created.

MainWindow

The **`MainWindow`** object represents the Conent Tab window that contains the Buddy List. You will use an instance of this object to control Tab plugins. With this object you can access the buddy list and UI elements of the Tab. The Main Window object supports the following interfaces:

- IWindow
- IContactListUI
- _IEventsWindow
- _IEventsContactListUI

Properties

Name	Description
SelectedFriends	Returns Collection of Friend objects containing the friends selected in the buddy list.
SelectedOnlineFriends	Returns Collection of type Friend object containing friends selected in the buddy list that are also online.
WinOpen	**`Boolean`**. Returns true if the window is opened or closed.
WinVisible	**`Boolean`**. Returns ture if the window is visible.
WinX	**`Numeric`**. Returns the top left corner horizontal position in pixels.
WinY	**`Numeric`**. Returns the top left corner vertical position in pixels.
WinWidth	**`Numeric`**. Returns the width of the window in pixels.
WinHeight	**`Numeric`**. Returns the height of the window in pixels.
WinHWND	**`Numeric`**. Returns a window handler for the window.
WinTitle	**`String`**. Returns the window title.
WinForeground	**`Boolean`**. Returns True if the window is in the foreground.
WinActive	**`Boolean`**. Returns true if the window has mouse of keyboard focus.
WinMinimized	**`Boolean`**. Returns True if the window is minimized.
WinMaximized	**`Boolean`**. Returns True if the window is maximized.

Events

Name	Description
Moved	Fired when window is moved.
Resized	Fired when window is resized.
Minimized	Fired when window is maximized.
Maximized	Fired when window is minimized.
Activated	Fired when the window loses or gains input focus.
Closed	Fired when window is closed.
FriendSelected	Fired when friend is selected in the user’s buddy list.
FriendUnselected	Fired when friend loses selection in the user’s buddy list.

Identity

The **`Identity`** object implements the following interfaces:

- IIdentity

Properties

Name	Description
Name	**`String`**. Returns the name of the user.
Active	**`Boolean`**. Returns True if the user is active.

Plugin

The `Plugin` object provides functionality common to all plug-ins. As such, you do not create an instance of this object, but one that inherits its functionality: `ConversationPluginHost`, `ContentTabSecondaryWindowPluginHost`, and `ContentTabMainPluginHost`. The `Plugin` object implements the following interface:

- IPlugin

Properties

Name	Description
PluginID	`String`. Returns the plug-in ID.
PluginName	`String`. Returns the plug-in name.
PluginVersion	`String`. Returns the plug-in version.
PluginLocale	`String`. Returns the locale of the plug-in.

PluginHost

The **`PluginHost`** object provides functionality common to all plug-ins. As such, you do not create an instance of this object, but one that inherits its functionality: **`ConversationPluginHost`**, **`ContentTabSecondaryWindowPluginHost`**, and **`ContentTabMainPluginHost`**. This object is only available to plug-ins. The **`PluginHost`** object implements the following objects.

- IPluginHost
- IPluginStorage
- IToast
- IColors
- IPluginHost
- IPlugin
- _IEventsPluginHost
- _IEventsToast
- _IEventsColors

Properties

Name	Description
Messenger	Returns a `Messenger` object.
AddressBook	Returns `AddressBook` object containing the user's address book. Returns null if access is restricted.
Cookies	Returns the messenger cookies. Returns null if plug-in was not loaded from the Yahoo! domain.
LastError	Returns an `Error` object containing the last error that occurred.
PluginID	`String`. Returns the plug-in ID.
PluginName	`String`. Returns the plug-in name.
PluginVersion	`String`. Returns the plug-in version.
PluginLocale	`String`. Returns the locale of the plug-in.

Methods

Name	Description
SendHTTPRequest	Returns the content of the provided URL.
SendHTTPRequestEx	Fetches and returns the content of the provided URL and optional headers
OpenURL	Opens the provided URL in a separate window.
LocalReady	Tells the host that the plug-in is ready.
ShowFileBrowserDialog	Launches the Show File Browser Dialog box.
ShowBuddyPicker	Launches the Buddy Picker dialog box.
StorageWrite	Writes a key and its value to storage.
StorageRead	Reads the value for a given key.
StorageRemove	Removes a key and its value from storage.
ToastShow	Pops up the toast window.
ToastClose	Closes the toast window.

Events

Name	Description
ToastLinkClicked	Fired when the user clicks a link in the toast window.
Unloaded	Fired when the host is about to unload the plug-in.

Name	Description
HTTPRequestCompleted	Fired when the URL content has completed downloading.
HTTPRequestError	Fired when the URL download fails.

Status

The Status object contains online status information for a Messenger user, such as “OnLunch”, “Offline” or “Busy”. You can obtain status information for the user or for friends in the buddy list. The **`Status`** object supports the following interfaces:

- IStatus

Properties

Name	Description
Status	**`String`**. Returns the status ID of the user or friend.
StatusLocalized	**`String`**. Returns the localized status ID of the user or friend.
DND	**`String`**. Returns the DND (Do Not Disturb) status ID of the user or friend.
Data	**`String`**. Returns the status description text for the user or friend.
LinkType	**`String`**. Returns the type of status link, if any.
Message	**`String`**. Returns the status message.
IdleTime	**`String`**. Returns the amount of time in seconds that the user or friend has been idle.

Methods

Name	Description
SetCustomStatus	Sets the **`Custom`** status.

Voice

The Voice object enables you to send text messages from a Messenger plug-in to a phone or computer. The `Voice` object supports the following interfaces:

- IVoiceActions

Methods

Name	Description
CallPSTN	Call a PSTN number.
CallFriendComputer	Call a friend’s computer.

Interfaces

IAddressBook

The **`IAddressBook`** interface is implemented by the following objects:

- AddressBook

Properties

Name	**Description**
Entries	Returns a collection of AddressBookEntry objects.

Methods

Name	**Description**
EntryFromFriend	Retrieves the AddressBookEntry object for a friend.

Entries

Returns a collection of address book entries from the users address book.

JavaScript Syntax

```
yhost.AddressBook.Entries;
```

JavaScript Example

Get all entries in the address book and shows first and last name for each.

```
yhost = window.external;
function ABRetrieveAll()
{
var plugAB = yhost.AddressBook;
  if (plugAB == null)
    document.eventsForm.triggered.value += "Address Book Entry Access is
    Denied" + "\n" + "\n";
  else
  {
    var ABiter = new Enumerator(plugAB.Entries);
    for (; !ABiter.atEnd(); ABiter.moveNext())
    {

      var frndentry = ABiter.item();
      document.eventsForm.triggered.value += "\tFirst Name: " +
        frndentry.FirstName + "\n";
      document.eventsForm.triggered.value += "\tLast Name: " +
        frndentry.LastName + "\n";
    }
  }
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT Entries();
```

EntryFromFriend

Retrieves the address book entry for a specified friend.

Parameters

Parameter	Description
`Friend`	Friend object. The friend to retrieve the address book entry for.

JavaScript Syntax

```
Addressbookentry = yhost.AddressBook.EntryFromFriend(Friend);
```

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABRetrieveEntry(ABFriend)
{
  var frndsABiter, listAB, ABentryout, ABFriend;

  listAB = phost.AddressBook;
  if (listAB == null)
```

```
    document.eventsForm.triggered.value += "Address Book Entry Access is
      Denied" + "\n" + "\n";
  else
  {
    ABentryout = listAB.EntryFromFriend(ABFriend);

    document.eventsForm.triggered.value += "Address Book Entry
      Retrieved:" + "\n";
    document.eventsForm.triggered.value += "\tFriend ID: " + ABFriend.id +
      "\n";
    document.eventsForm.triggered.value += "\tFirst Name: " +
      ABentryout.FirstName + "\n";
    document.eventsForm.triggered.value += "\tLast Name: " +
      ABentryout.LastName + "\n";
    }
  }
}
```

C++ Syntax

```
[id(0x00000002)]
        VARIANT EntryFromFriend([in] VARIANT vFriend);
```

IAddressBookEntry

The **`IAddressBookEntry`** interface provides the interface to a user's address book entry in the buddy list. The **`IAddressBookEntry`** Interface is implemented by the following objects:

- AddressBookEntry

Properties

Name	Description
Friend	Returns a Friend object for the friend in the buddy list associated with this address book entry.
FirstName	**`String`**. First name.
MiddleName	**`String`**. Middle name.
LastName	**`String`**. Last name.
NickName	**`String`**. Nick name.
JobTitle	**`String`**. Job title.
PhoneNumberHome	**`String`**. Home phone number.
PhoneNumberWork	**`String`**. Work phone number.
PhoneNumberMobile	**`String`**. Mobile phone number.
PhoneNumberPager	**`String`**. Pager phone number.
PhoneNumberFax	**`String`**. Fax phone number.
PhoneNumberOther	**`String`**. Other phone number.
EmailAddress	**`String`**. Email address.
EmailAddress1	**`String`**. Second email address.
EmailAddress2	**`String`**. Other email address.
HomeAddressStreet	**`String`**. Home street address.
HomeAddressCity	**`String`**. City of home address
HomeAddressState	**`String`**. State of home address.
HomeAddressPostalCode	**`String`**. Postal code of home address.
HomeAddressPostalCountry	**`String`**. Country of home address.
Company	**`String`**. Company.
WorkAddressStreet	**`String`**. Work street address.
WorkAddressCity	**`String`**. City of the work address.
WorkAddressState	**`String`**. State of the work address.
WorkAddressPostalCode	**`String`**. Postal code of the work address.
WorkAddressPostalCountry	**`String`**. Country of the work address.
WorkPhone	**`String`**. Work phone number.
WorkWebSite	**`String`**. Work web site.
BirthdayDay	**`Numeric`**. Day of birth.

Name	Description
BirthdayMonth	**`Numeric`**. Month of birth.
BirthdayYear	**`Numeric`**. Year of birth.
AnniversaryDay	**`Numeric`**. Day of anniversary.
AnniversaryMonth	**`Numeric`**. Month of anniversary.
AnniversaryYear	**`Numeric`**. Year of anniversary.
Notes	**`String`**. Notes.
NotesCustom1	**`String`**. Custom Notes1.
NotesCustom2	**`String`**. Custom Notes2.
NotesCustom3	**`String`**. Custom Notes3.
`NotesCustom4`	**`String`**. Custom Notes4.
WebSite	**`String`**. Website.
UniqueSessionID	**`String`**. Unique session ID.

Friend

Returns a **`Friend`** object. The friend in the buddy list associated with this address book entry.

JavaScript Syntax

`AddressBookEntry``.Friend;`

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABPlugin()
{
  var plugAB, ABiter, listAB, frndsABiter, frndentry;

  plugAB = phost.AddressBook;
  if (plugAB == null)
    ABResult.innerHtml = "Address Book Entry access is denied";
  else
  {
    ABiter = new Enumerator(plugAB.Entries);
  for (; !ABiter.atEnd(); ABiter.moveNext())
  {

    frndentry = ABiter.item();
    document.PluginForm.ABdetail.value += frndentry.Friend + " " +
      frndentry.Friend.id + "\n";
    document.PluginForm.ABdetail.value += frndentry.FirstName + " " +
      frndentry.MiddleName + " " + frndentry.LastName + " " +
      frndentry.NickName + "\n";
    document.PluginForm.ABdetail.value += frndentry.JobTitle +
      frndentry.PhoneNumberHome + frndentry.PhoneNumberWork + "\n";
}}
```

C++ Syntax

```
[id(0x000001f5), propget]
        IFriend* Friend();
```

FirstName

Returns the first name.

JavaScript Syntax

`AddressBookEntry``.FirstName;`

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABPlugin()
{
  var plugAB, ABiter, listAB, frndsABiter, frndentry;

  plugAB = phost.AddressBook;
  if (plugAB == null)
    ABResult.innerHtml = "Address Book Entry access is denied";
  else
  {
    ABiter = new Enumerator(plugAB.Entries);

  for (; !ABiter.atEnd(); ABiter.moveNext())
  {
    frndentry = ABiter.item();
    document.PluginForm.ABdetail.value += frndentry.Friend + " " +
      frndentry.Friend.id + "\n";
    document.PluginForm.ABdetail.value += frndentry.FirstName + " " +
      frndentry.MiddleName + " " + frndentry.LastName + " " +
      frndentry.NickName + "\n";
    document.PluginForm.ABdetail.value += frndentry.JobTitle +
      frndentry.PhoneNumberHome + frndentry.PhoneNumberWork + "\n";
    }}
```

C++ Syntax

```
[id(0x000000c9), propget]
        VARIANT FirstName();
```

MiddleName

Returns the middle name.

JavaScript Syntax

AddressBookEntry`.MiddleName;`

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABPlugin()
{
  var plugAB, ABiter, listAB, frndsABiter, frndentry;

  plugAB = phost.AddressBook;
  if (plugAB == null)
    ABResult.innerHtml = "Address Book Entry access is denied";
  else
  {
    ABiter = new Enumerator(plugAB.Entries);

  for (; !ABiter.atEnd(); ABiter.moveNext())
  {
    frndentry = ABiter.item();
    document.PluginForm.ABdetail.value += frndentry.Friend + " " +
      frndentry.Friend.id + "\n";
    document.PluginForm.ABdetail.value += frndentry.FirstName + " " +
      frndentry.MiddleName + " " + frndentry.LastName + " " +
      frndentry.NickName + "\n";
    document.PluginForm.ABdetail.value += frndentry.JobTitle +
      frndentry.PhoneNumberHome + frndentry.PhoneNumberWork + "\n";
    }}
```

C++ Syntax

```
[id(0x000000ca), propget]
        VARIANT MiddleName();
```

LastName

Returns the last name.

JavaScript Syntax

```
AddressBookEntry.LastName;
```

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABPlugin()
{
  var plugAB, ABiter, listAB, frndsABiter, frndentry;

  plugAB = phost.AddressBook;
  if (plugAB == null)
    ABResult.innerHtml = "Address Book Entry access is denied";
  else
  {
    ABiter = new Enumerator(plugAB.Entries);

  for (; !ABiter.atEnd(); ABiter.moveNext())
  {
    frndentry = ABiter.item();
    document.PluginForm.ABdetail.value += frndentry.Friend + " " +
      frndentry.Friend.id + "\n";
    document.PluginForm.ABdetail.value += frndentry.FirstName + " " +
      frndentry.MiddleName + " " + frndentry.LastName + " " +
    frndentry.NickName + "\n";
  document.PluginForm.ABdetail.value += frndentry.JobTitle +
    frndentry.PhoneNumberHome + frndentry.PhoneNumberWork + "\n";
  }}
```

C++ Syntax

```
[id(0x000000cb), propget]
        VARIANT LastName();
```

NickName

Returns the nick name.

JavaScript Syntax

`AddressBookEntry.NickName;`

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABPlugin()
{
  var plugAB, ABiter, listAB, frndsABiter, frndentry;

  plugAB = phost.AddressBook;
  if (plugAB == null)
    ABResult.innerHtml = "Address Book Entry access is denied";
  else
  {
    ABiter = new Enumerator(plugAB.Entries);

  for (; !ABiter.atEnd(); ABiter.moveNext())
  {
    frndentry = ABiter.item();
    document.PluginForm.ABdetail.value += frndentry.Friend + " " +
      frndentry.Friend.id + "\n";
    document.PluginForm.ABdetail.value += frndentry.FirstName + " " +
      frndentry.MiddleName + " " + frndentry.LastName + " " +
      frndentry.NickName + "\n";
    document.PluginForm.ABdetail.value += frndentry.JobTitle +
      frndentry.PhoneNumberHome + frndentry.PhoneNumberWork + "\n";
    }}
```

C++ Syntax

```
[id(0x000000cc), propget]
        VARIANT NickName();
```

JobTitle

Returns the job title.

JavaScript Syntax

AddressBookEntry`.JobTitle;`

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABPlugin()
{
  var plugAB, ABiter, listAB, frndsABiter, frndentry;

  plugAB = phost.AddressBook;
  if (plugAB == null)
    ABResult.innerHtml = "Address Book Entry access is denied";
  else
  {
    ABiter = new Enumerator(plugAB.Entries);

  for (; !ABiter.atEnd(); ABiter.moveNext())
  {
    frndentry = ABiter.item();
    document.PluginForm.ABdetail.value += frndentry.Friend + " " +
      frndentry.Friend.id + "\n";
    document.PluginForm.ABdetail.value += frndentry.FirstName + " " +
      frndentry.MiddleName + " " + frndentry.LastName + " " +
      frndentry.NickName + "\n";
    document.PluginForm.ABdetail.value += frndentry.JobTitle +
      frndentry.PhoneNumberHome + frndentry.PhoneNumberWork + "\n";
}}
```

C++ Syntax

```
[id(0x000000cd), propget]
        VARIANT JobTitle();
```

PhoneNumberHome

Returns the home phone number.

JavaScript Syntax

```
AddressBookEntry.PhoneNumberHome;
```

JavaScript Example

C++ Syntax

```
[id(0x000000ce), propget]
        VARIANT PhoneNumberHome();
```

PhoneNumberWork

Returns the work phone number.

JavaScript Syntax

```
AddressBookEntry.PhoneNumberWork;
```

JavaScript Example

Gets the address book entry for a specified friend.

```
function ABPlugin()
{
  var plugAB, ABiter, listAB, frndsABiter, frndentry;

  plugAB = phost.AddressBook;
  if (plugAB == null)
    ABResult.innerHtml = "Address Book Entry access is denied";
  else
  {
    ABiter = new Enumerator(plugAB.Entries);

  for (; !ABiter.atEnd(); ABiter.moveNext())
  {
    frndentry = ABiter.item();
    document.PluginForm.ABdetail.value += frndentry.Friend + " " +
      frndentry.Friend.id + "\n";
    document.PluginForm.ABdetail.value += frndentry.FirstName + " " +
      frndentry.MiddleName + " " + frndentry.LastName + " " +
      frndentry.NickName + "\n";
    document.PluginForm.ABdetail.value += frndentry.JobTitle +
      frndentry.PhoneNumberHome + frndentry.PhoneNumberWork + "\n";
}}
```

C++ Syntax

```
[id(0x000000cf), propget]
        VARIANT PhoneNumberWork();
```

PhoneNumberMobile

Returns the mobile phone number.

JavaScript Syntax

`AddressBookEntry``.PhoneNumberMobile;`

JavaScript Example

C++ Syntax

```
[id(0x000000d0), propget]
        VARIANT PhoneNumberMobile();
```

PhoneNumberPager

Returns the pager number.

JavaScript Syntax

```
AddressBookEntry.PhoneNumberPager;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d1), propget]
        VARIANT PhoneNumberPager();
```

PhoneNumberFax

Returns the fax number.

JavaScript Syntax

```
AddressBookEntry.PhoneNumberFax;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d2), propget]
        VARIANT PhoneNumberFax();
```

PhoneNumberOther

Returns other phone number.

JavaScript Syntax

```
AddressBookEntry.PhoneNumberOther;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d3), propget]
        VARIANT PhoneNumberOther();
```

EmailAddress

Returns the email address.

JavaScript Syntax

```
AddressBookEntry.EmailAddress;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d4), propget]
        VARIANT EmailAddress();
```

EmailAddress1

Returns a secondary email address.

JavaScript Syntax

```
AddressBookEntry.EmailAddress1;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d5), propget]
        VARIANT EmailAddress1();
```

EmailAddress2

Returns additional email address.

JavaScript Syntax

```
AddressBookEntry.EmailAddress2;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d6), propget]
        VARIANT EmailAddress2();
```

HomeAddressStreet

Returns the home street address.

JavaScript Syntax

```
AddressBookEntry.HomeAddressStreet;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d7), propget]
        VARIANT HomeAddressStreet();
```

HomeAddressCity

Returns the home city.

JavaScript Syntax

```
AddressBookEntry.HomeAddressCity;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d8), propget]
        VARIANT HomeAddressCity();
```

HomeAddressState

Returns the home state.

JavaScript Syntax

```
AddressBookEntry.HomeAddressState;
```

JavaScript Example

C++ Syntax

```
[id(0x000000d9), propget]
        VARIANT HomeAddressState();
```

HomeAddressPostalCode

Returns the home postal code.

JavaScript Syntax

```
AddressBookEntry.HomeAddressPostalCode;
```

JavaScript Example

C++ Syntax

```
[id(0x000000da), propget]
        VARIANT HomeAddressPostalCode();
```

HomeAddressPostalCountry

Returns the home country.

JavaScript Syntax

```
AddressBookEntry.HomeAddressPostalCountry;
```

JavaScript Example

C++ Syntax

```
[id(0x000000db), propget]
        VARIANT HomeAddressPostalCountry();
```

Company

Returns the Company the user works for.

JavaScript Syntax

```
AddressBookEntry.Company;
```

C++ Syntax

```
[id(0x000000dc), propget]
VARIANT Company();
```

WorkAddressStreet

Returns the street of the work address.

JavaScript Syntax

```
AddressBookEntry.WorkAddressStreet;
```

JavaScript Example

C++ Syntax

```
[id(0x000000dd), propget]
        VARIANT WorkAddressStreet();
```

WorkAddressCity

Returns the city of the work address.

JavaScript Syntax

```
AddressBookEntry.WorkAddressCity;
```

C++ Syntax

```
[id(0x000000de), propget]
        VARIANT WorkAddressCity();
```

WorkAddressState

Returns the state of the work address.

JavaScript Syntax

```
AddressBookEntry.WorkAddressState;
```

C++ Syntax

```
[id(0x000000df), propget]
        VARIANT WorkAddressState();
```

WorkAddressPostalCode

Returns the postal code of the work address.

JavaScript Syntax

```
AddressBookEntry.WorkAddressPostalCode;
```

C++ Syntax

```
[id(0x000000e0), propget]
        VARIANT WorkAddressPostalCode();
```

WorkAddressPostalCountry

Returns the country of the work address.

JavaScript Syntax

```
AddressBookEntry.WorkAddressPostalCountry;
```

C++ Syntax

```
[id(0x000000e1), propget]
        VARIANT WorkAddressPostalCountry();
```

WorkPhone

Returns the work phone number.

JavaScript Syntax

```
AddressBookEntry.WorkPhone;
```

C++ Syntax

```
[id(0x000000e2), propget]
        VARIANT WorkPhone();
```

WorkWebSite

Returns the work website.

JavaScript Syntax

```
AddressBookEntry.WorkWebSite;
```

C++ Syntax

```
[id(0x000000e3), propget]
        VARIANT WorkWebSite();
```

BirthdayDay

Returns the day of birth.

JavaScript Syntax

```
AddressBookEntry.BirthdayDay;
```

C++ Syntax

```
[id(0x000000e4), propget]
        VARIANT BirthdayDay();
```

BirthdayMonth

Returns the month of birth.

JavaScript Syntax

```
AddressBookEntry.BirthdayMonth;
```

C++ Syntax

```
[id(0x000000e5), propget]
        VARIANT BirthdayMonth();
```

BirthdayYear

Returns the year of birth.

JavaScript Syntax

```
AddressBookEntry.BirthdayYear;
```

C++ Syntax

```
[id(0x000000e6), propget]
        VARIANT BirthdayYear();
```

AnniversaryDay

Returns the anniversary day.

JavaScript Syntax

```
AddressBookEntry.AnniversaryDay;
```

C++ Syntax

```
[id(0x000000e7), propget]
        VARIANT AnniversaryDay();
```

AnniversaryMonth

Returns the anniversary month.

JavaScript Syntax

```
AddressBookEntry.AnniversaryMonth;
```

C++ Syntax

```
[id(0x000000e8), propget]
        VARIANT AnniversaryMonth();
```

AnniversaryYear

Returns the anniversary year.

JavaScript Syntax

```
AddressBookEntry.AnniversaryYear;
```

C++ Syntax

```
[id(0x000000e9), propget]
        VARIANT AnniversaryYear();
```

Notes

Returns notes.

JavaScript Syntax

```
AddressBookEntry.Notes;
```

C++ Syntax

```
[id(0x000000ea), propget]
        VARIANT Notes();
```

NotesCustom1

Returns custom notes.

JavaScript Syntax

```
AddressBookEntry.NotesCustom1;
```

C++ Syntax

```
[id(0x000000eb), propget]
        VARIANT NotesCustom1();
```

NotesCustom2

Returns custom notes.

JavaScript Syntax

```
AddressBookEntry.NotesCustom2;
```

C++ Syntax

```
[id(0x000000ec), propget]
        VARIANT NotesCustom2();
```

NotesCustom3

Returns custom notes.

JavaScript Syntax

```
AddressBookEntry.NotesCustom3;
```

C++ Syntax

```
[id(0x000000ed), propget]
        VARIANT NotesCustom3();
```

NotesCustom4

Returns custom notes.

JavaScript Syntax

```
AddressBookEntry.NotesCustom4;
```

C++ Syntax

```
[id(0x000000ee), propget]
        VARIANT NotesCustom4();
```

WebSite

Returns web site.

JavaScript Syntax

```
AddressBookEntry.WebSite;
```

C++ Syntax

```
[id(0x000000ef), propget]
        VARIANT WebSite();
```

UniqueSessionID

Returns a unique ID for this session. These can be compared decide if two COM objects represent same object.

JavaScript Syntax

`AddressBookEntry``.UniqueSessionID;`

JavaScript Example

C++ Syntax

```
[id(0x00000065), propget]
        VARIANT UniqueSessionID();
```

IColors

The `IColors` interface is implemented by the following objects. These colors reflect the user's Yahoo! Messenger appearance preference setting for Messenger Interface Style, or "skin". If the user chooses to change their skin, the 'ColorsChanged' event is fired.

- PluginHost
- ContentTabPluginCommon
- ContentTabMainWindowPluginHost
- ContentTabSecondaryWindowPluginHost
- ConversationPluginHost

Properties

Name	Description
ColorsBackground	The dialog box background color.
ColorsHighlight	The color of edges that face the light source.
ColorsShadow	The color of edges that face away from the light source.
ColorsSelected	The color of the selected item in a control.
ColorsButtonText	The color of text on push buttons.
ColorsSelectedText	Color of a selected item(s) text in a control.

ColorsBackground

Returns the dialog box background color.

JavaScript Syntax

```
yhost.ColorsBackground;
```

JavaScript Example

Creates and event handler to update the plug-in's main window and secondary window (if any) background color when the user changes their Messenger appearance settings.

```
yhost.SetEventHandler('ColorsChanged', on_UpdateColors);
```

```
function on_ColorsChanged);
{
  document.body.style.backgroundColor = yhost.ColorsBackground;

  if (g_currentSecondaryWindow != null)
  document.body.style.backgroundColor =
    g_currentSecondaryWindow.ColorsBackground;
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT ColorsBackground();
```

ColorsHighlight

The color of edges facing the light source.

JavaScript Syntax

```
yhost.ColorsHighlight;
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT ColorsHighlight();
```

ColorsShadow

Returns the color of edges facing away from the light source.

JavaScript Syntax

```
yhost.ColorsShadow;
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT ColorsShadow();
```

ColorsSelected

Returns the color of selected items in a control.

JavaScript Syntax

```
yhost.ColorsSelected;
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT ColorsSelected();
```

ColorsButtonText

Returns the color of text on push buttons.

JavaScript Syntax

```
yhost.ColorsButtonText;
```

C++ Syntax

```
[id(0x00000005), propget]
        VARIANT ColorsButtonText();
```

ColorsSelectedText

Returns the color of selected text on push buttons.

JavaScript Syntax

```
yhost.ColorsSelectedText;
```

JavaScript Example

C++ Syntax

```
[id(0x00000006), propget]
        VARIANT ColorsSelectedText();
```

IContactList

The **`IContactList`** interface is implemented by the following objects.

- Messenger

Properties

Name	**Description**
Friends	Returns a collection of **`Friend`** objects.
Groups	Returns a collection of **`Group`** objects.
FriendsOnline	Returns a collection of **`Friend`** objects of for friends that are online.

Friends

Use the **`Friends`** property to get a collection of Friend objects for the friends in a user's buddy list.

Returns

Collection of **`Friend`** objects.

JavaScript Syntax

```
myMessenger.Friends;
```

JavaScript Example

The following example displays information about each friend in the user's buddy list.

```
function DisplayFriends()
{
var listfrnd = new Enumerator(yhost.Messenger.Friends);

for (; !listfrnd.atEnd(); listfrnd.moveNext()){
  var lfrnd = listfrnd.item();
  document.MEForm.detail.value += "Friend ID: ";
  document.MEForm.detail.value += lfrnd.id + "\n";
  document.MEForm.detail.value += "Image : ";
  document.MEForm.detail.value += lfrnd.image + "\n";
}}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT Friends();
```

Groups

Use the **`Groups`** property to get a collection of Group objects for the groups in a user's buddy list.

Returns

Collection of **`Group`** objects.

JavaScript Syntax

`myMessenger.Groups`

JavaScript Example

The following example adds a friend to the specified group in the user's buddy list.

```
function AddFriend(ID, AddGroup)
{
var iter, agroup;
  iter = new Enumerator(myMessenger.Groups);
  for (; !iter.atEnd(); iter.moveNext()  {
    agroup = iter.item();
    if(agroup.Name == AddGroup){
      myMessenger.Addfriend(ID, agroup);
}}}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Groups();
```

FriendsOnline

Use the **`FriendsOnline`** property to get a collection of Friend objects for the friends in a user’s buddy list.

Returns

Collection of **`Friend`** objects.

JavaScript Syntax

```
myMessenger.FriendsOnline;
```

JavaScript Example

The following example sends a plug-in instant message to user’s from the buddy list that are online.

```
function SendPluginMessage_Call (msg)
{
var iter, friend;

iter = new Enumerator(yhost.Messenger.FriendsOnline);

for (; !iter.atEnd(); iter.moveNext())
{
  friend = iter.item();
  yhost.SendPluginMessage(msg, friend);
  log_last_error("SendPluginMessage");
}}
```

C++ Syntax

```
[id(0x00000003), propget]
       VARIANT FriendsOnline();
```

IContactListUI

The **`IContactListUI`** interface provides UI functionality for the Content Tab. This interface is implemented by the following object:

- MainWindow

Properties

Name	Description
SelectedFriends	Returns a collection of type Friend object containing friends selected in the buddy list.
SelectedOnlineFriends	Returns a collection of type Friend object containing friends selected in the buddy list that are also online.

SelectedFriends

Use the **`SelectedFriends`** property to get a collection of friends selected by the user in the buddy list.

Returns

Collection of Friend objects.

JavaScript Syntax

```
var FriendCollection = myMessenger.MainWindow.SelectedFriends;
```

JavaScript Example

The following example gets the collection of selected friends from the buddy list and displays their information.

```
function DisplayIContactListUI(){
  var selfrndlist, selfrnd;

  var mainwinobj = m1.MainWindow;
  selfrndlist=new Enumerator(m1.MainWindow.SelectedFriends);
```

```
    for (; !selfrndlist.atEnd(); selfrndlist.moveNext()){
     {
      selfrnd = selfrndlist.item();
      document.MEForm.detail.value += "IContactListUI, Selected Friends
      Collection, Friend Object: " + "\n";
      document.MEForm.detail.value +="\tFriend ID: " + selfrnd.id + "\n";
      document.MEForm.detail.value += "\tFriend Image: " + selfrnd.Image
        +"\n";
      document.MEForm.detail.value += "\tFriend Unique Session ID: " +
        selfrnd.UniqueSessionID + "\n";
      document.MEForm.detail.value += "\tFriend FirstName: " +
        selfrnd.FirstName + "\n";
      document.MEForm.detail.value += "\tFriend LastName: " +
        selfrnd.LastName + "\n";
      document.MEForm.detail.value += "\tFriend Peer to Peer: " +
        selfrnd.PeerToPeer + "\n";
      document.MEForm.detail.value += "\n";
}}
```

C++ Syntax

```
[id(0x00000001), propget]
       VARIANT SelectedFriends();
```

SelectedOnlineFriends

Use the **`SelectedFriends`** property to get a collection of online friends in the buddy list.

Returns

Collection of Friend objects.

JavaScript Syntax

```
var OnlineFriendCollection =
myMessenger.MainWindow.SelectedOnlineFriends;
```

JavaScript Example

The following example gets the collection of online friends from the buddy list and displays their information.

```
function DisplayIContactListUI(){
  var selfrndlist, uilistgrp, selonlist, selgrp;
  var selfrnd, uigrp, onfrnd;

  var mainwinobj = m1.MainWindow;
  selonlist=new Enumerator(m1.MainWindow.SelectedOnlineFriends);

  for (; !selonlist.atEnd(); selonlist.moveNext()) }
        onfrnd = selonlist.item();
        document.MEForm.detail.value += "IContactListUI, Friends
          Online Collection, Friend Object: " + "\n";
        document.MEForm.detail.value +="\tFriend ID: " + onfrnd.id +
          "\n";
        document.MEForm.detail.value += "\tFriend Image: " + onfrnd.Image
          +"\n";
        document.MEForm.detail.value += "\tFriend Unique Session ID: " +
          onfrnd.UniqueSessionID + "\n";
        document.MEForm.detail.value += "\tFriend FirstName: " +
          onfrnd.FirstName + "\n";
        document.MEForm.detail.value += "\tFriend LastName: " +
          onfrnd.LastName + "\n";
        document.MEForm.detail.value += "\tFriend Peer to Peer: " +
          onfrnd.PeerToPeer + "\n";
        document.MEForm.detail.value += "\n";
}}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT SelectedOnlineFriends();
```

IContentTabPluginServices

The **`IContentTabPluginServices`** interface is implemented by the following objects.

- ContentTabPluginCommon
- ContentTabMainWindowPluginHost

Properties

Name	Description
ContentTabTitle	**`String`**.
MainPluginWindow	**`Boolean`**. If True, this is the main plug-in window. If False, this is a secondary plug-in window.
FileTransfer	Returns a FileTransferManager object.

Methods

Name	Description
SendPluginMessage	Sends a plug-in message to a friend.
SetPluginStatus	Sets the plug-in status.

ContentTabTitle

Use the **`ContentTabTitle`** property to get or update the displayed title of the Tab plug-in.

Initially, ContentTabTitle returns null. Calling ContentTabTitle will add a hyphen to the existing plug-in name (specified in the PluginName entry in the manifest file), followed by the new title name. If ContentTabTitle has been set, it returns the portion of the title that follows the hyphen, but not the full name that displays in the title bar.

JavaScript Syntax

```
myHost.ContentTabTitle;
```

JavaScript Example

Sets the text in the text for the title bar of the Tab plug-in window.

```
yhost = window.external;
yhost.ContentTabTitle = "Tab Plug-in";
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT ContentTabTitle();
```

MainPluginWindow

Returns whether this is the main window, or a secondary window of the plug-in. If True, this is the main plug-in window; if False, this is a secondary window.

JavaScript Syntax

```
yhost.MainPluginWindow;
```

JavaScript Example

See what type of window the plug-in is.

```
yhost = window.external;
g_IsMainWnd = yhost.MainPluginWindow;
```

C++ Syntax

```
[id(0x00000005), propget]
        VARIANT MainPluginWindow();
```

FileTransfer

Returns a FileTransferManager object that enables the plug-in to send and receive files.

JavaScript Syntax

```
ftm = yhost.FileTransfer;
```

JavaScript Example

Here, a Tab plug-in calls the FileTransfer method to get a reference to a FileTransferManager object. The function StartNewFileTransfer function calls the FileTransferManager's FileTransferSend method to send the file to a specified friend.

```
yhost = window.external;
yhost.ContentTabTitle = "Tab Plug-in";
ftm = yhost.FileTransfer;

function StartNewFileTransfer(path, strFriend){
  var friend;
  friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft;
  ft = ftm.FileTransferSend(path, friend);
  var pref = "File Transfer: ";
  append_log_msg(pref + ft.ID);
  append_log_msg(pref + ft.Path);
  append_log_msg(pref + ft.Size);
  append_log_msg(pref + ft.Transferred);
  append_log_msg(pref + ft.Status);
  append_log_msg(pref + ft.Direction);
}

StartNewFileTransfer(FilePath, MyFriend);
```

C++ Syntax

```
[id(0x00000006), propget]
        VARIANT FileTransfer();
```

SendPluginMessage

Sends a message from this plug-in instance to the friend's plug-in instance. The maximum number of bytes that can be sent is 9000. An error is returned if the plug-in tries to send more than 9000 bytes.

Parameters

Parameter	Description
`Message`	**`String`**. The message to send to the other plug-in instance.
`Friend`	Friend object. The friend to send the message to.

JavaScript Syntax

```
yhost.SendPluginMessage(msg, friend);
```

JavaScript Example

Sends a message from the plug-in to friends that are online.

```
function SendPluginMessage_Call (msg)
{
  var iter, friend;
  iter = new Enumerator(yhost.Messenger.FriendsOnline);
  for (; !iter.atEnd(); iter.moveNext())
  {
    friend = iter.item();
    yhost.SendPluginMessage(msg, friend);
    log_last_error("SendPluginMessage");
  }
}
```

C++ Syntax

```
[id(0x00000004)]
        void SendPluginMessage(
                        [in] VARIANT mes,
                        [in] VARIANT varFriend);
```

SetPluginStatus

Sets the plug-in status.

Parameters

Parameter	Description
`Text`	**`String`**. Status message. Must be less that 256 characters. If this string is empty it removes the status.
`Data`	**`String`**. Status related data. Must be less that 450 characters.

JavaScript Syntax

```
yhost.SetPluginStatus(Text, Data);
```

C++ Syntax

```
void SetPluginStatus(
                            [in] VARIANT mes,
                            [in] VARIANT Data);
```

IConversationPluginServices

The **`IConversationPluginServices`** interface is implemented by the following objects.

- ConversationPluginHost

Properties

Name	**Description**
Inviter	**`Boolean`**. If True, this instance is the inviter; if False, this instance is the invitee.
FileTransfer	**`Returns a FileTransferManager`** object.

Methods

Name	**Description**
SendPluginMessage	Sends a message to other side's instance of this plug-in participating in same conversation.

Inviter

Returns True if this instance of the plug-in is the inviter. Returns False if this instance of the plug-in is the invitee.

JavaScript Syntax

```
yhost.Inviter;
```

JavaScript Example

If this is the host that has invited another user to use the plug-in, set the initial language to english.

```
if (host.Inviter) {
           lang = "nl_en";
  var text_input = document.getElementById('lang_choice');
  text_input.value = lang;
}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Inviter();
```

FileTransfer

Returns a FileTransferManager object. It will be null if the file transfer operation is blocked for this object. File transfer is blocked if the Sendfile entry in the manifest file is set to False.

JavaScript Syntax

```
yhost.FileTransfer;
```

JavaScript Example

The following example creates a reference to a FileTransferManager object, and creates an event handler to catch incoming file transfers.

```
yhost = window.external;
ftm = yhost.FileTransfer;

if (ftm != null)
  ftm.SetEventHandler('FileTransferIncoming', on_FileTransferIncoming);

function on_FileTransferIncoming(ft, friend)
{
  if (cb_IEventsFileTransfer.checked)
    append_log_msg("FileTransferIncoming, ID: " + ft.ID + ", From Friend:
       " + friend.ID);

  ftm.FileTransferAccept(ft, "c:/temp/file123.jpg");
}
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT FileTransfer();
```

SendPluginMessage

Sends a message to other side's instance of this plug-in participating in same conversation. The maximum that can be sent is 9KB. An error is returned if the plug-in tries to send more than 9KB. Possible LastError code is: "RemotePartyNotReady".

Parameters

Parameter	Type/Description
`Msg`	**`String`** or **`Byte array`**. The text to send to the other plug-in instance. Maximum is 9000 bytes.

JavaScript Syntax

```
yhost.SendPluginMessage(msg);
```

JavaScript Example

The following example sends a message to the other plug-in instance.

```
submit.onclick = function()
{
  for (var i = 0; i < participants.friends.length; ++i)
  {
    yhost.SendPluginMessage(text.value, participants.friends[i]);
  }
  chat.innerHTML += "<b>Me: " + text.value + "</b><br/>";
  text.value = "";
}
```

C++ Syntax

```
[id(0x00000001)]
        void SendPluginMessage([in] VARIANT Message);
```

IConversationWindow

The **`IConversationWindow`** interface provides the functionality for the Conversation window launched by Messenger when the user wants to start a conversation with a friend. `IConversationWindow` is implemented by the following objects:

- ConversationWindow
- ConversationPluginHost

Properties

Name	Description
Window	Returns a ConversationWindow object.
InputWindowText	**`String`**. Read an write text to and from the Conversation edit box.

Methods

Name	Description
InsertInHistoryWindow	Inserts text into the history window. Plain text is supported.
SendIM	Send the instant message.

InputWindowText

Use this function to read and write text to and from the Conversation Window input window.

JavaScript Syntax

```
yhost.InputWindowText;
```

JavaScript Example

Get the text from the window when the **`UserIsTyping`** event is fired.

```
yhost.SetEventHandler("UserIsTyping", On_UserIsTyping)
function OnUserIsTyping()
{
  var GetInputText;
  document.eventsForm.triggered.value += "Conversation Window Event: User
    is Typing" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
  // get what the user was typing
  GetInputText=yhost.InputWindowText;
}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT InputWindowText();
```

Window

Returns a **`ConversationWindow`** object.

JavaScript Syntax

```
yhost.Window;
```

JavaScript Example

Gets a reference to a conversation window and checks to see if it was successful.

```
function get_Window_Call ()
{
  wnd = yhost.Window;
  if (wnd == null)
    get_Window_Res.innerHTML = "result: null";
  else
    get_Window_Res.innerHTML = "result: valid obj.";
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT Window();
```

InsertInHistoryWindow

Use the **`InsertInHistoryWindow`** function to insert text into the history window. Plain text is supported. (Richer formatting – TBD).

Parameters

Parameter	Type/Description
`Msg`	**`String`**. The text to insert into the history window.

JavaScript Syntax

```
yhost.InsertInHistoryWindow(Msg);
```

JavaScript Example

When a user receives a message from another plug-in instance, the OnPluginMessage event handler puts some text in the user's history window.

```
yhost.SetEventHandler( 'PluginMessage', onPluginMessage );

function onPluginMessage(msg) {
  switch (msg){
    case "ACCEPT":
        yhost.InsertInHistoryWindow("Friend has accepted your invite.
          Please establishing voice call and click on \"Start
          sharing\".");
        break;
    case "REJECT":
       yhost.InsertInHistoryWindow("Friend has rejected your invite.");
       break;
}}
```

C++ Syntax

```
[id(0x00000003)]
        void InsertInHistoryWindow([in] VARIANT msg);
```

SendIM

Sends an instant message to another user. The plug-in can only send instant messages if the Sendim entry in the manifest file is set to True. Last Error will be set to “Restricted” if the plug-in is not allowed this functionality.

Calling this method does not cause the “IMSent” event to fire.

Parameters

Parameter	Description
IM	**`String`**. The instant message text. Can contain HTML formatting. See IM Tags in the reference section for a complete list of allowed tags.

JavaScript Syntax

```
yhost.SendIM(IM);
```

JavaScript Example

Sends an instant message using input from an entry into a form window.

```
Send_Msg(document.frm.txt.value);
function Send_Msg(msg)
{
  var some = host.window.CurrentIdentity.Name;
  yhost.SendIM(msg);
  msg =  "<b>" + some + ": </b>" + msg;
  yhost.InsertInHistoryWindow(msg);
  yhost.InputWindowText= msg;
  document.frm.txt.value="";
}
```

C++ Syntax

```
[id(0x00000004)]
        void SendIM([in] VARIANT im);
```

ICreatedWindows

The **`ICreatedWindows`** interface is implemented by the following objects.

- Messenger

Properties

Name	Description
MainWindow	Returns a MainWindow object.
Windows	Returns the collection of objects implementing the IWindow interface.

MainWindow

Use the **`MainWindow`** property to get the MainWindow object.

Returns

`MainWindow` object

JavaScript Syntax

References the messenger and plug-in windows, sets event handlers, and initializes the plug-in.

```
function on_load()
 {
  var yhost = window.external;
  var mainWin = yhost.Messenger.MainWindow;
  mainWin.SetEventHandler("FriendSelected", FriendSelected);
  yhost.SetEventHandler ('PluginMessage', onPlugInMesgRecieve);
  yhost.LocalReady();
 }
```

JavaScript Example

```
var mainwinobj = myMessenger.MainWindow;
```

C++ Syntax

```
[id(0x00000001), propget]
```

```
VARIANT MainWindow();
```

Windows

Use this property to get a collection of objects that of a specified type that implement the **`IWindows`** Interface. Currently, only **`ConversationWindow`** is supported.

Parameters

Parameter	Description
`Type`	**`String`**. Type of **`IWindows`** objects to return; currently, can be “Conversation” only.

Returns

Collection of **`IConversationWindow`** objects.

JavaScript Syntax

```
myMessenger.Windows(Type);
```

JavaScript Example

The following example gets the conversation window and displays its information.

```
function DisplayConversationWindowInfo()
{
var winset, winConversation, frnds, fpart,

winConversation = new Enumerator(m1.Windows("Conversation"));

for (; !winConversation.atEnd(); winConversation.moveNext()){

  winset = winConversation.item();
  frnds = new Enumerator(winset.Friends);

  for (; !frnds.atEnd(); frnds.moveNext()){

    fpart = frnds.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation Window
      Object"  + "\n"
    document.MEForm.detail.value += "\tIs Window Visible? " +
      winset.WinVisible + "\n"
    document.MEForm.detail.value += "\tTitle: " + winset.WinTitle + "\n"
    document.MEForm.detail.value += "\tOpen: " + winset.WinOpen + "\n"
    document.MEForm.detail.value += "\tX: " + winset.WinX + "\n"
    document.MEForm.detail.value += "\tY: " + winset.WinY  + "\n"
    document.MEForm.detail.value += "\tWidth: " + winset.WinWidth + "\n"
    document.MEForm.detail.value += "\tHeight: " + winset.WinHeight +
      "\n"
    document.MEForm.detail.value += "\tWindow Hanlder #: " +
      winset.WinHWND + "\n"
    document.MEForm.detail.value += "\tForeground: " +
      winset.WinForeground + "\n"
    document.MEForm.detail.value += "\tActive: " + winset.WinActive +
      "\n"
    document.MEForm.detail.value += "\tMinimized: " +
      winset.WinMinimized + "\n"
    document.MEForm.detail.value += "\tMaximized: " +
      winset.WinMaximized + "\n"
    document.MEForm.detail.value += "\tMy Current Identity: " +
      winset.CurrentIdentity.Name + "\n";
    document.MEForm.detail.value += "\tParticipating Friends: " +
      fpart.id + "\n";
    document.MEForm.detail.value + "\n" + "\n";
}}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Windows([in] VARIANT winType);
```

IError

The **`IError`** interface is implemented by the following objects:

- Error

Properties

Name	Description
ErrorID	**`String`**. Returns the error ID. Returns **`“NoError” if`** no error has occurred.

ErrorID

Returns the error ID. If no error occurs, “NoError” is returned; otherwise, the error ID is returned. The error returned will depend on which function caused the error. See the documentation for the function for possible errors.

JavaScript Syntax

```
myError.ErrorID;
```

JavaScript Example

```
function log_last_error(method_name)
{
  if (yhost.LastError.ErrorID != "NoError")
  {
    append_log_msg (method_name + " error: '" + yhost.LastError.ErrorID +
      "'");
  }
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT ErrorID();
```

IFileTransfer

The **`IFileTransfer`** interface is implemented by the following objects.

- FileTransfer

Properties

Name	Description
ID	**`String`**. Unique Id for the file transfer.
Path	**`String`**. Local file path.
Size	**`Numeric`**. File size in bytes.
Transferred	**`Numeric`**. The size of the information in bytes that was successfully transferred.
Status	**`String`**. File transfer status. Possible values are: “NotStarted”: file transfer has not started. “InProgress” file transfer is in progress. “Cancelled”: file transfer was cancelled by user. “Transferred”: file transfer has completed. “Error”: file transfer was unable to complete.
Direction	**`String`**. Direction of the file transfer: Possible values: “Incoming”: file is received from another user. “Outgoing”: file is sent to another user.

ID

Returns the unique ID for this file transfer.

JavaScript Syntax

```
FileTransfer.ID
```

JavaScript Example

File name and ID are displayed when a file transfer is received.

```
function On_FileTransferIncoming(itransfer)
```

```
{
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
    Event" + "\n";
  document.eventsForm.triggered.value += "New Incoming File Transfer
    Occurring." + " File name and location is: " + itransfer.Path + "\n";
  document.eventsForm.triggered.value += "File Transfer ID: " +
    itransfer.ID + "\n";
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT ID();
```

Path

Returns the local file path to the file to be transferred.

JavaScript Syntax

```
FileTransfer.Path;
```

JavaScript Example

Starts a file transfer and log information about the transfer, including the file name and location where the file was retrieved from.

```
function StartNewFileTransfer(path, strFriend)
{
  ftm = yhost.FileTransfer;
  var friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft = ftm.FileTransferSend(path, friend);
  var pref = "File Transfer: ";
  append_log_msg(pref + ft.ID);
  append_log_msg(pref + ft.Path);
  append_log_msg(pref + ft.Size);
  append_log_msg(pref + ft.Transferred);
  append_log_msg(pref + ft.Status);
  append_log_msg(pref + ft.Direction);
}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Path();
```

Size

Returns the file size in bytes of the file to be transferred.

JavaScript Syntax

```
FileTransfer.Size;
```

JavaScript Example

Send a file to a friend and log information about the transfer, including the file size.

```
function StartNewFileTransfer(path, strFriend)
{
  var friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ftm = yhost.FileTransfer;
  var ft = ftm.FileTransferSend(path, friend);
  var pref = "File Transfer: ";
  append_log_msg(pref + ft.ID);
  append_log_msg(pref + ft.Path);
  append_log_msg(pref + ft.Size);
  append_log_msg(pref + ft.Transferred);
  append_log_msg(pref + ft.Status);
  append_log_msg(pref + ft.Direction);
}
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT Size();
```

Transferred

Returns the size in bytes of the information that was successfully transferred.

JavaScript Syntax

```
FileTransfer.Transferred;
```

JavaScript Example

Send a file to a friend and log information about the transfer, including the number of bytes that were sent.

```
function StartNewFileTransfer(path, strFriend)
{
  ftm = yhost.FileTransfer;
  var friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft = ftm.FileTransferSend(path, friend);
  var pref = "File Transfer: ";
  append_log_msg(pref + ft.ID);
  append_log_msg(pref + ft.Path);
  append_log_msg(pref + ft.Size);
  append_log_msg(pref + ft.Transferred);
  append_log_msg(pref + ft.Status);
  append_log_msg(pref + ft.Direction);
}
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT Transferred();
```

Status

Returns the status of the file transfer. Can be one of the following:

Status	**Description**
NotStarted	File transfer as not yet started.
InProgress	File transfer is in progress.
Cancelled	File transfer was cancelled.
Transferred	File transfer has completed.
Error.	File transfer was not able to finish.

JavaScript Syntax

```
FileTransfer.Status;
```

JavaScript Example

```
function On_FileTransferIncoming(itransfer)
{
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
    Event" + "\n";
  document.eventsForm.triggered.value += "New Incoming File Transfer
    Occurring." + " File name and location is: " + itransfer.Path + "\n";
  document.eventsForm.triggered.value += "File Transfer Status: " +
    itransfer.Status + "File Transfer Direction: " + itransfer.Direction +
    "\n";
}
```

C++ Syntax

```
[id(0x00000005), propget]
        VARIANT Status();
```

Direction

Returns the direction type of the file transfer. Can be “Incoming” or “Outgoing”.

JavaScript Syntax

```
FileTransfer.Direction;
```

JavaScript Example

```
function On_FileTransferIncoming(itransfer)
{
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
    Event" + "\n";
  document.eventsForm.triggered.value += "File Transfer Status: " +
    itransfer.Status + "File Transfer Direction: " + itransfer.Direction +
    "\n";
}
```

C++ Syntax

```
[id(0x00000006), propget]
        VARIANT Direction();
```

IFileTransferManager

The `IFileTransferManager` interface is implemented by the following objects.

- FileTransferManager

Methods

Name	Description
FileTransferSend	Creates the outgoing FileTransfer object.
FileTransferAccept	Accepts the incoming file transfer.
FileTransferDecline	Declines the incoming file transfer.
FileTransferCancel	Cancels the file transfer.

FileTransferSend

Creates and returns the outgoing FileTransfer object. The `friend` parameter can be null, but the event `_IEventsFileTransferManager` returns a Friend object that always represents the other person in the conversation.

Parameters

Parameter	Description
file	`String`. The local file path location of the file being sent. Unicode file names are not supported.
friend	Friend object. The friend who will receive the file.

JavaScript Syntax

```
pFileTransfer = yhost.FileTransferSend(path, friend);
```

JavaScript Example

Sends a file to a friend.

```
function StartNewFileTransfer(path, strFriend)
```

```
{
  var friend;
  friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft;
  ft = ftm.FileTransferSend(path, friend, ft);
}
```

C++ Syntax

```
[id(0x00000001)]
        VARIANT FileTransferSend(
                        [in] VARIANT file,
                        [in] VARIANT varFriend);
```

FileTransferAccept

Accepts an incoming file transfer. Sets the local “Save as” file.

Parameters

Parameter	Description
`transfer`	FileTransfer object. This object is received as a parameter in the **`_IEventsFileTransferManager::FileTransferIncoming`** event call.
path	**`String`**. File path to use when saving the file. If this path is null or empty, the default file name will be used instead.

JavaScript Syntax

```
yhost.FileTransferAccept(transfer, path)
```

JavaScript Example

Creates an event handler to accept incoming file transfers.

```
ftm.SetEventHandler('FileTransferIncoming', on_FileTransferIncoming);

function on_FileTransferIncoming(ft, friend)
{
  if (cb_IEventsFileTransfer.checked)
```

```
    append_log_msg("FileTransferIncoming, ID: " + ft.ID + ", From Friend:
    " + friend.ID);

  ftm.FileTransferAccept(ft, "c:/temp/file123.jpg");
}
```

C++ Syntax

```
[id(0x00000002)]
        void FileTransferAccept(
                        [in] VARIANT FileTransfer,
                        [in] VARIANT fileLocal);
```

FileTransferDecline

Declines an incoming file transfer.

Parameters

Parameter	Description
`transfer`	FileTransfer object. Received as a parameter in the **`_IEventsFileTransferManager::FileTransferDeclined`** call.

JavaScript Syntax

`yhost``.FileTransferDecline(`*`transfer`*`);`

JavaScript Example

Creates an event handler for when the user declines a file sent to them by another user.

```
ftm.SetEventHandler('FileTransferIncoming', on_FileTransferIncoming);

function on_FileTransferIncoming(ft, friend)

{
  yhost.FileTransferDecline( );
  if (cb_IEventsFileTransfer.checked)
  append_log_msg("FileTransferDeclined, ID: " + ft.ID + ", From Friend: "
    + friend.ID);
}

yhost.FileTransferDecline(ft);
```

C++ Syntax

```
[id(0x00000003)]
        void FileTransferDecline([in] VARIANT FileTransfer);
```

FileTransferCancel

Cancels the file transfer. The receiving plug-in is sent a “FileTransferDone” event, and the sending plug-in is sent a “FileTransferRemotePartyCancelled” event.

Parameters

Parameter	Description
`transfer`	FileTransfer object.

JavaScript Syntax

```
yhost.FileTransferCancel(transfer);
```

C++ Syntax

```
[id(0x00000004)]
        void FileTransferCancel([in] VARIANT FileTransfer);
```

IFriend

The IFriend interface is implemented by the following objects:

- Friend

Properties

Name	Description
Groups	Returns collection of Group objects containing the groups that the friend belongs to.
ID	**`String`**. Returns the friend's ID.
Status	Returns a Status object containing the friend's status information.
Image	**`String`**. Returns the file name of the friend's image.
UniqueSessionID	**`String`**. Returns a unique session ID for the object that can be used to verify the object is the correct one.
FirstName	**`String`**. Returns the first name of the friend.
LastName	**`String`**. Returns the last name of the friend.
PeerToPeer	**`Boolean`**. Returns True if peer to peer connection is available.
Plugins	Collection of Plugin objects. Returns all the plug-ins that the friend is running.
OnContactList	**`Boolean`**. Returns True if the friend is in the buddy list.

Groups

Returns a collection of Group objects for each group that the friend belongs to in the buddy list.

JavaScript Syntax

```
myFriend.Groups;
```

JavaScript Example

Cycles through the groups that the friend belongs to and displays the name of the group.

```
for (var i = 0; i < myFriend.groups.length; i++) {
  document.MEForm.detail.value += "\tGroup Name: " +
    myFriend.groups[i].Name + "\n";
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT Groups();
```

ID

Returns the user ID of the friend.

JavaScript Syntax

`myFriend.ID;`

JavaScript Example

The following example logs a friend's information when selected on the buddy list.

```
onFriendSelected(friend) {
  LogClear();
  Log("friend.ID, FirstName, LastName: " + friend.ID + " " +
    friend.FirstName + " " + friend.LastName);
  entry = g_obj.EntryFromFriend(friend);
  if (entry) {
    Log(".First, middle, last, nick: " + entry.FirstName + " " +
      entry.MiddleName + " " + entry.LastName + " " + entry.NickName);
    Log("EmailAddress1, 2, 3: " + entry.EmailAddress + " " +
      entry.EmailAddress1 + " " + entry.EmailAddress2);
  }
  RefreshWishlist(friend.FirstName + "%20" + friend.LastName);
}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT ID();
```

Status

Returns a Status object with the current status information for the friend.

JavaScript Syntax

```
myFriend.Status;
```

JavaScript Example

When the **`FriendChanged`** event is triggered, the OnFriendChanged event handler displays the friend's status message text.

```
startevent.Messenger.SetEventHandler("FriendChanged", OnFriendChanged)

function OnFriendChanged(frndchange)
{
  document.eventsForm.triggered.value += "Contact List Event, Friend
    changed" + "\n";
  document.eventsForm.triggered.value += "\tData: " +
    frndchange.Status.Data + "\n";
}
```

C++ Syntax

```
[id(0x00000006), propget]
        VARIANT Status();
```

Image

Returns the file name of the friends display image.

JavaScript Syntax

```
myFriend.Image;
```

JavaScript Example

When the **`FriendChanged`** event is triggered, the event handler displays the friend's status message text.

```
startevent.Messenger.SetEventHandler("FriendChanged", OnFriendChanged)

function OnFriendChanged(frndchange)
{
  document.eventsForm.triggered.value += "Contact List Event, Friend
    changed" + "\n";
  document.eventsForm.triggered.value += "\tImage: " +
    frndchange.Image + "\n";
}
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT Image();
```

UniqueSessionID

Returns a unique session ID for the object that can be used to verify the object is the correct one.

JavaScript Syntax

`myFriend``.UniqueSessionID;`

JavaScript Example

When a friend in the buddy list changes, the event handler will be fired and display the friend's information.

```
startevent.Messenger.SetEventHandler("FriendChanged",
OnFriendChanged)

function OnFriendChanged(frndchange)
{
  document.eventsForm.triggered.value += "Contact List Event, Friend
    changed" + "\n";
  document.eventsForm.triggered.value += "\tFriend ID: " + frndchange.ID +
    "\n";
  document.eventsForm.triggered.value += "\tUnique Session ID: " +
    frndchange.UniqueSessionID  + "\n";
  document.eventsForm.triggered.value += "\tFirstName LastName " +
    frndchange.FirstName + " " + frndchange.LastName + "\n";
  document.eventsForm.triggered.value += "\tStatus: " +
    frndchange.Status.Status + " " + "Localized: " +
    frndchange.Status.Localized + " " + "DND: " + frndchange.Status.DND +
    " " + "\n";
  document.eventsForm.triggered.value += "\tData: " +
    frndchange.Status.Data + "\n";
  document.eventsForm.triggered.value += "\tLink Type: " +
    frndchange.Status.LinkType + "\n";
  document.eventsForm.triggered.value += "\tLink: " +
    frndchange.Status.Link + "\n";
  document.eventsForm.triggered.value += "\tIdle Time: " +
    frndchange.Status.IdleTime + "\n";
  document.eventsForm.triggered.value += "\tImage: " + frndchange.image +
    "\n";
  document.eventsForm.triggered.value += "\tPeerTOPeer: " +
    frndchange.PeerToPeer + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000005), propget]
        VARIANT UniqueSessionID();
```

FirstName

Returns the first name of the friend.

JavaScript Syntax

```
myFriend.FirstName;
```

JavaScript Example

When the **`FriendChanged`** event is triggered, the **`OnFriendChanged`** event handler displays the first and last name of the friend.

```
startevent.Messenger.SetEventHandler("FriendChanged", OnFriendChanged)

function OnFriendChanged(frndchange)
```

```
{
  document.eventsForm.triggered.value += "Contact List Event, Friend
    changed" + "\n";
  document.eventsForm.triggered.value += "\tFirstName LastName " +
    frndchange.FirstName + " " + frndchange.LastName + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000007), propget]
       VARIANT FirstName();
```

LastName

Returns the last name of the friend.

JavaScript Syntax

```
myFriend.LastName;
```

JavaScript Example

When the **`FriendChanged`** event is triggered, the **`OnFriendChanged`** event handler displays the first and last name of the friend.

```
startevent.Messenger.SetEventHandler("FriendChanged", OnFriendChanged)

function OnFriendChanged(frndchange)
{
  document.eventsForm.triggered.value += "Contact List Event, Friend
    changed" + "\n";
  document.eventsForm.triggered.value += "\tFirstName LastName " +
    frndchange.FirstName + " " + frndchange.LastName + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000008), propget]
       VARIANT LastName();
```

PeerToPeer

Returns whether a peer to peer connection with the friend is available.

JavaScript Syntax

```
myFriend.PeerToPeer;
```

JavaScript Example

When a friend in the buddy list changes, the event handler will be fired and display the friend's information.

```
startevent.Messenger.SetEventHandler("FriendChanged",
OnFriendChanged)

function OnFriendChanged(frndchange)
{
  document.eventsForm.triggered.value += "Contact List Event, Friend
    changed" + "\n";
  document.eventsForm.triggered.value += "\tFriend ID: " + frndchange.ID +
    "\n";
  document.eventsForm.triggered.value += "\tUnique Session ID: " +
    frndchange.UniqueSessionID  + "\n";
  document.eventsForm.triggered.value += "\tFirstName LastName " +
    frndchange.FirstName + " " + frndchange.LastName + "\n";
  document.eventsForm.triggered.value += "\tStatus: " +
    frndchange.Status.Status + " " + "Localized: " +
    frndchange.Status.Localized + " " + "DND: " + frndchange.Status.DND +
    " " + "\n";
  document.eventsForm.triggered.value += "\tData: " +
    frndchange.Status.Data + "\n";
  document.eventsForm.triggered.value += "\tLink Type: " +
    frndchange.Status.LinkType + "\n";
  document.eventsForm.triggered.value += "\tLink: " +
    frndchange.Status.Link + "\n";
  document.eventsForm.triggered.value += "\tIdle Time: " +
    frndchange.Status.IdleTime + "\n";
  document.eventsForm.triggered.value += "\tImage: " + frndchange.image +
    "\n";
  document.eventsForm.triggered.value += "\tPeerTOPeer: " +
    frndchange.PeerToPeer + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000009), propget]
        VARIANT PeerToPeer();
```

Plugins

Returns a collection of Plugin objects that represent the plugins that the friend is running.

JavaScript Syntax

```
myFriend.Plugins;
```

JavaScript Example

This example displays each plug-in that the friend is running.

```
function DisplayPlugins()
{
var pluginlist, pluginitem;
  pluginlist=new Enumerator(myFriend.Plugins);
  for (; !pluginlist.atEnd(); pluginlist.moveNext())
    pluginitem = pluginlist.item();
}
```

C++ Syntax

```
[id(0x0000000a), propget]
        VARIANT Plugins();
```

OnContactList

Returns True if the friend is in the buddy list.

JavaScript Syntax

```
myFriend.OnContactList;
```

JavaScript Example

This example displays a friend's information if they are in the buddy list.

```
if (myFriend.OnContactList)
{
  document.MEForm.detail.value += "Contact List Event, Friend in buddy
list" + "\n";
  document.MEForm.detail.value += "\tFriend ID: " + myFriend.ID +
    "\n";
  document.MEForm.detail.value += "\tUnique Session ID: " +
    myFriend.UniqueSessionID  + "\n";
  document.MEForm.detail.value += "\tFirstName LastName " +
    myFriend.FirstName + " " + myFriend.LastName + "\n";
  document.MEForm.detail.value += "\tStatus: " +
    myFriend.Status.Status + " " + "Localized: " +
    myFriend.Status.Localized + " " + "DND: " + myFriend.Status.DND +
    " " + "\n";
  document.MEForm.detail.value += "\tData: " + myFriend.Status.Data +
    "\n";
  document.MEForm.detail.value += "\tLink Type: " +
    myFriend.Status.LinkType + "\n";
  document.MEForm.detail.value += "\tLink: " + myFriend.Status.Link +
    "\n";
  document.MEForm.detail.value += "\tIdle Time: " +
    myFriend.Status.IdleTime + "\n";
  document.MEForm.detail.value += "\tImage: " + myFriend.image + "\n";
  document.MEForm.detail.value += "\tPeerTOPeer: " +
    myFriend.PeerToPeer + "\n";
  document.MEForm.detail.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x0000000b), propget]
        VARIANT OnContactList();
```

IGroup

The **`IGroup`** interface provides access to a group in the buddy list. The **`IGroup`** Interface is implemented by the following objects:

- Group

Properties

Name	Description
Friends	Returns Collection of Friend objects containing a list of friends that belong to the group.
Name	**`String`**. Returns the name of the group.
UniqueSessionID	**`String`**. Returns the unique session ID of the Group object.

Friends

Returns a collection of Friend objects belonging to this group.

JavaScript Syntax

`myMessenger``.Groups.Friends;`

JavaScript Example

The following example displays the information for each friend in each group in the buddy list

```
function DisplayGroups()
{
  var listgrp, lgroup, fgrp;
  listfrndgrp = new Enumerator(m1.Friends);
  listgrp = new Enumerator(m1.Groups);

  for (; !listgrp.atEnd(); listgrp.moveNext())
  {
    for (; !listfrndgrp.atEnd(); listfrndgrp.moveNext())
    {
    lgroup = listgrp.item();
    fgroup = listfrndgrp.item();
    if(fgrp == lgroup.Friends.fgroup.id)
      document.MEForm.detail.value += "fgrp: " + fgrp + " group id: " +
        lgroup.fgroup.id;
    else
      document.MEForm.detail.value += "fgrp: " + fgrp + " group id: " +
      lgroup.fgroup.id;
      document.MEForm.detail.value += "Group Collection Properties: " +
        "\n";
      document.MEForm.detail.value += "\tGroup Name: " + lgroup.Name +
        "\n";
      document.MEForm.detail.value += "\tUnique Session ID: " +
        lgroup.UniqueSessionID + "\n";
      document.MEForm.detail.value += "\n";
}}}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT Friends();
```

Name

Returns the name of the group.

JavaScript Syntax

```
myMessenger.Groups.Name;
```

JavaScript Example

The following example adds a new friend to a specified group.

```
function AddFriend(ID, AddGroup)
{
  var iter, agroup;
  iter = new Enumerator(m1.Groups);
  for (; !iter.atEnd(); iter.moveNext())
  {
    agroup = iter.item();
    if(agroup.Name == AddGroup)
    {
      m1.Addfriend(ID, agroup);
```

```
     }
}}
```

C++ Syntax

```
[id(0x00000002), propget]
          VARIANT name();
```

UniqueSessionID

Returns a unique session ID for the object to be used to compare with another object to determine if they represent the same object.

JavaScript Syntax

```
myMessenger.Groups.UniqueSessionID;
```

C++ Syntax

```
[id(0x00000065), propget]
          VARIANT UniqueSessionID();
```

IIdentity

The **`IIdentity`** interface is implemented by the following objects:

- Identity

Properties

Name	**Description**
Name	**`String`**. Returns the name of the user.
Active	**`Boolean`**. Returns True if the user is active.

Name

Returns the name of the user.

JavaScript Syntax

```
yhost.Messenger.Me.LoginIdentity.Name;
```

JavaScript Example

The following example displays the identity information for the logged in user.

```
function DisplayIdentities()
{
  var listidentity, lidentity;
  listident = new Enumerator(yhost.Messenger.Me.Identities);

  for (; !listident.atEnd(); listident.moveNext())
  {
    lidentity = listident.item();
    document.MEForm.detail.value += "Identity Name: ";
    document.MEForm.detail.value += lidentity.Name + "\n";
    document.MEForm.detail.value += "Active Identity: ";
    document.MEForm.detail.value += lidentity.Active + "\n";
}}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT name();
```

Active

Returns True if the user is active.

JavaScript Syntax

```
yhost.Messenger.Me.LoginIdentity.Active;
```

JavaScript Example

The following example displays the identity information for the logged in user.

```
function DisplayIdentities()
{
  var listidentity, lidentity;
  listident = new Enumerator(yhost.Messenger.Me.Identities);

  for (; !listident.atEnd(); listident.moveNext())
  {
    lidentity = listident.item();
    document.MEForm.detail.value += "Identity Name: ";
    document.MEForm.detail.value += lidentity.Name + "\n";
    document.MEForm.detail.value += "Active Identity: ";
    document.MEForm.detail.value += lidentity.Active + "\n";
}}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Active();
```

IIMMessage

The **`IIMMessage`** Interface is implemented by the following objects:

- IMMessage

Properties

Name	Description
Text	**`String`**. The text sent in an instant message. Text can be HTML formatted. See IM Tags in the reference section for a complete list of allowed tags.

Text

The text sent in the instant message. The text can contain HTML formatting information. See IM Tags in the reference section for a complete list of allowed tags.

JavaScript Syntax

```
IMMessage.Text;
```

JavaScript Example

Displays the text in the instant message.

```
startevent.Messenger.MainWindow.SetEventHandler("IncomingIM",
On_IncomingIM);
function On_IncomingIM(IM)
{
  document.eventsForm.triggered.value += "Conversation Window Event:
    Incoming IM: " + IM.Text + "\n";
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT Text();
[id(0x00000001), propput]
        void Text([in] VARIANT rhs);
```

IIMMessage
Text

IMe

The `IMe` interface is implemented by the following object:

- Me

Properties

Name	Description
Identities	Returns collection containing Identity objects for all identities associated with the user.
LoginIdentity	Returns Identity Object containing the ID used to log the user in for this session.
Status	Returns a Status Object containing status information displayed to other users.
Image	**`String`**. Returns the file name of user's image.
FirstName	**`String`**. Returns the first name of user.
LastName	**`String`**. Returns the last name of user.

Identities

Returns a collection of Identity objects for all identities belonging to the user.

JavaScript Syntax

```
yhost.Messenger.Me.Identities;
```

JavaScript Example

The following example gets the list of identities for the user and displays information for each identity.

```
listme = new Enumerator(m1.Me.Identities);
for (; !listme.atEnd(); listme.moveNext())
{
  meidentity = listme.item();
  document.MEForm.detail.value += "ME Objects, LoginIdentity, Identity
```

```
    Object:" + "\n";
  document.MEForm.detail.value += "\tIdenity Name: " + meidentity.Name +
    "\n";
  document.MEForm.detail.value += "\tIdentity, Active: " +
    meidentity.Active + "\n";
  document.MEForm.detail.value += "\n";
}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Identities();
```

LoginIdentity

Returns the user ID used to log the user in to Messenger.

JavaScript Syntax

```
yhost.Messenger.Me.LoginIdentity;
```

JavaScript Example

The following example displays the login identity for the user.

```
document.MEForm.detail.value += "Login Identity: " + m1.Me.LoginIdentity +
"\n";
```

C++ Syntax

```
[id(0x00000005), propget]
        VARIANT LoginIdentity();
```

Status

Returns a Status object for the user.

JavaScript Syntax

```
yhost.Messenger.Me.Status;
```

JavaScript Example

Displays the user's current status information.

```
document.MEForm.detail.value += "User Status: " + m1.Me.Status.Status +
"\n";
document.MEForm.detail.value += "DND: " + m1.Me.Status.DND + "\n";
document.MEForm.detail.value += "Status text: " + m1.Me.Status.Data +
"\n";
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT Status();
```

Image

Returns the filename of the user's image, if any.

JavaScript Syntax

```
yhost.Messenger.Me.Image;
```

JavaScript Example

Displays the user's status and image file name.

```
document.MEForm.detail.value += "User Status: " + m1.Me.Status.Status +
"\n";
document.MEForm.detail.value += "User Image: " + m1.Me.Image + "\n";
```

C++ Syntax

```
[id(0x00000006), propget]
        VARIANT Image();
```

FirstName

Returns the user's first name.

JavaScript Syntax

```
yhost.Messenger.Me.FirstName;
```

JavaScript Example

Gets the status and first and last name of the user.

```
document.MEForm.detail.value += "User Status: " + m1.Me.Status.Status +
"\n";
document.MEForm.detail.value += "First Name: " + m1.Me.FirstName + "\n";
document.MEForm.detail.value += "Last Name: " + m1.Me.LastName + "\n";
```

C++ Syntax

```
[id(0x00000007), propget]
        VARIANT FirstName();
```

LastName

Returns the user's last name.

JavaScript Syntax

```
yhost.Messenger.Me.LastName;
```

JavaScript Example

Gets the status and first and last name of the user.

```
document.MEForm.detail.value += "User Status: " + m1.Me.Status.Status +
"\n";
document.MEForm.detail.value += "First Name: " + m1.Me.FirstName + "\n";
document.MEForm.detail.value += "Last Name: " + m1.Me.LastName + "\n";
```

IMe
LastName

C++ Syntax

```
[id(0x00000008), propget]
        VARIANT LastName();
```

IMessenger2

The `IMessenger2` interface is implemented by the following objects.

- Messenger

Properties

Name	Description
Me	References a Me object for the user currently logged in to Messenger.
LastError	Returns an Error object containing information for the last occurring error.
AddressBook	Returns an AddressBook object.

Methods

Name	Description
Login	Login to Messenger. Only available to external applications; not available from within the plug-in.
LoginUser	Login to Messenger with the specified user ID and password. Only available to external applications; not available from within the plug-in.
Logout	Logout from Messenger. Only available to external applications; not available from within the plug-in.
GetFriendFromID	Returns a Friend object for the given friend ID.
GetGroupFromName	Returns the Group object given the group name.

Me

Use the `Me` property to get the current user's information.

Returns

Me object reference.

JavaScript Syntax

```
yhost.Messenger.Me;
```

JavaScript Example

The following example gets an enumerated list of the Yahoo! identities of the current user.

```
function buttonMeOnClick()
{
  var user = m1.Me.User
  var loginIdentity = m1.Me.LoginIdentity
  var status = m1.Me.Status

  LogData("Me:")
  LogData("User: " + user.ID)
  LogData("LoginIdentity: " + loginIdentity.Name)
  LogData("Status: " + status.Status + ", " + status.StatusLocalized + ",
    " + status.DND)

  var identity;

  var iter = new Enumerator(m1.Me.Identities);

  LogData("Identities:")
  for (; !iter.atEnd(); iter.moveNext())
  {
    identity = iter.item();
    LogData(identity.Name)
}}
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT Me();
```

AddressBook

Returns an Address book object corresponding to the user's address book.

Note: The Address book only exposes C++ interfaces. The plug-in can get an AddressBook object, if permitted, through the host object. The AddressBook setting in the manifest file must be set to True, to get the address book.

JavaScript Syntax

```
yhost.Messenger.AddressBook;
```

JavaScript Example

C++ Syntax

```
[id(0x00000008), propget]
        void AddressBook(VARIANT* pABook);
```

LastError

Use the `LastError` property to get the last occurring error. Returns an `Error` object reference

JavaScript Syntax

```
yhost.Messenger.LastError;
```

JavaScript Example

The following example checks for the last occurring error, and displays it.

```
function log_last_error(method_name) {
  if (yhost.LastError.ErrorID != "NoError"){
    append_log_msg (method_name + " error: '" +
      yhost.LastError.ErrorID + "'");
  }
}
```

C++ Syntax

```
[id(0x00000005), propget]
VARIANT LastError();
```

Login

Use the **`Login`** method to log the user in to Messenger. This function is only available to external applications.

JavaScript Syntax

```
yhost.Messenger.Login();
```

C++ Syntax

```
[id(0x00000001)]
        void Login();
```

LoginUser

Use the **`LoginUser`** method to log the user in providing the user's ID and password. his function is only available to external applications.

Parameters

Parameter	Description
User	**`String`**. Messenger user ID. If empty, the current user ID is used.
Password	**`String`**. Password. Password to use during the login.

JavaScript Syntax

```
yhost.Messenger.LoginUser(ID, Password);
```

C++ Syntax

```
[id(0x00000002)]
        void LoginUser(
                        [in] VARIANT vUser,
                        [in] VARIANT vPassword);
```

Logout

Use the **`Logout`** method to log the user out from Messenger. This function is only available to external applications.

JavaScript Syntax

```
yhost.Messenger.Logout()
```

C++ Syntax

```
[id(0x00000003)]
        void Logout();
```

GetFriendFromID

Use the **`GetFriendFromID`** method to get the Friend object given an ID.

Parameters

Parameter	**Description**
ID	**`String`**. User's ID.

JavaScript Syntax

```
yhost.Messenger.GetFriendFromID(strFriend);
```

JavaScript Example

The following example gets a **`Friend`** object and sends a file to that user.

```
function StartNewFileTransfer(path, strFriend)
{
  var friend = yhost.Messenger.GetFriendFromID(strFriend);
  var ft = ftm.FileTransferSend(path, friend);
  DumpFileTransfer(ft)
}
```

C++ Syntax

```
[id(0x00000006)]
        VARIANT GetFriendFromID([in] VARIANT sFriendID);
```

GetGroupFromName

Use the **`GetGroupFromName`** method to get the Group object given the group name.

Parameters

Parameter	Description
Name	**`String`**. Group name.

JavaScript Syntax

```
yhost.Messenger.GetGroupFromName(strGroupName);
```

C++ Syntax

```
[id(0x00000007)]
        VARIANT GetGroupFromName([in] VARIANT name);
```

IMainPluginWindow

The **`IMainPluginWindow`** interface is implemented by the following objects:

- ContentTabMainWindowPluginHost

Properties

Name	Description
MainWindowState	**`String`**. Returns the state of the main window of a Tab plug-in. Can be one of the following: Docked, Embedded, Minimized, and Overflow.

Methods

Name	Description
SecondaryWindowOpen	Opens the secondary window.
SecondaryWindowOpenWithDimensions	Opens the secondary window with the provided dimensions.
SecondaryWindowClose	Closes the secondary window.
SendMessageToSecondaryWindow	Sends a message to the secondary window.

MainWindowState

Returns the state of the main window of the Tab plug-in as a text string. Possible states are docked, embedded, Minimized, and Overflow. A change in state will fire the MainWindowStateChanged event.

State	Description
Docked	Separate window opened and docked to the side of the main messenger window.
Embedded	Window is embedded inside the main messenger window.

State	Description
Minimized	Window is minimized inside the main messenger window.
Overflow	Window is in the overflow menu.

JavaScript Syntax

```
yhost.MainWindowState;
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT MainWindowState();
```

SecondaryWindowOpen

Opens the secondary window. While the main window and secondary windows are part of the same plug-in, an HTML file displayed in the secondary window must call the **`LocalReady`** method independently in order to communicate with the main window.

Opening a URL with pop-ups in a secondary window can cause a scripting or debug error to be shown to the user. Therefore, it is recommended that you do not open URL with pop-ups in the secondary or main plugin window.

Parameters

Parameter	Description
URL	**`String`**. URL to navigate when the window opens. The URL can be relative to the main plug-in page location. If you provide the relative file path, the file is opened from the directory (for local plug-ins) or URL (for remote plug-ins) where the main plug-in page is located. Do not specify "file://" for a local file—only the relative path. The "file://" directory path format is not allowed. If you provide a URL, it will attempt to load without additional manipulation.
Title	**`String`**. Optional. Title of the secondary window. If blank or null, the plug-in name is used.

JavaScript Syntax

```
yhost.SecondaryWindowOpen(url);
```

JavaScript Example

Main window opens a secondary plug-in window.

```
window.external.SecondaryWindowOpen("ym-stocks-tab.htm");
```

C++ Syntax

```
[id(0x00000002)]
        void SecondaryWindowOpen(
                        [in] VARIANT url,
                        [in, optional] VARIANT title);
```

SecondaryWindowOpenWithDimensions

Opens the secondary window using the provided dimensions. These dimensions will overwrite the previously stored dimensions. The dimensions are stored and will be used next time **`SecondaryWindowOpen`** is called.

While the main window and secondary windows are part of the same plug-in, an HTML file displayed in the secondary window must call the **`LocalReady`** method independently in order to communicate with the main window.

Parameters

Parameter	**Description**
URL	**`String`**. URL to navigate when the window opens. The URL can be relative to the main plug-in page location. If you provide the relative file path, the file is opened from the directory (for local plug-ins) or URL (for remote plug-ins) where the main plug-in page is located. Do not specify “file://” for a local file—only the relative path. The “file://” directory path format is not allowed. If you provide a URL, it will attempt to load without additional manipulation.
Width	**`Numeric`**. Width of the window.
Height	**`Numeric`**. Height of the window.
Title	**`String`**. Optional. Title of the secondary window. If blank or null, the plug-in name is used.

JavaScript Syntax

```
yhost.SecondaryWindowOpenWithDimensions(URL, Width, Height, Title);
```

JavaScript Example

Opens the secondary window to 320 by 600 pixels for a list of friends.

```
function on_load()
{
  yhost.SecondaryWindowOpenWithDimensions("friendlist.html",
    320,600,"Friend List");
  yhost.SetEventHandler ('SecondaryWindowClosed',
    on_SecondaryWindowClosed);
  yhost.LocalReady();
}
```

C++ Syntax

```
[id(0x00000003)]
        void SecondaryWindowOpenWithDimensions(
                        [in] VARIANT url,
                        [in] VARIANT width,
                        [in] VARIANT height,
                        [in, optional] VARIANT title);
```

SecondaryWindowClose

Closes the secondary window.

JavaScript Syntax

```
yhost.SecondaryWindowClose();
```

C++ Syntax

```
[id(0x00000004)]
        void SecondaryWindowClose();
```

SendMessageToSecondaryWindow

Sends a message to the secondary window. If fails, Last Error will return "**`SecondaryWindowNotReady`**".

Parameters

Parameter	Description
Message	**`String`**. Message to send.

JavaScript Syntax

```
yhost.SendMessageToSecondaryWindow(Message);
```

JavaScript Example

Sends a message to the secondary window and logs the last error to see if it was successful.

```
yahoo = window.external;
```

```
yahoo.SetEventHandler('SecondaryWindowOpened', onSecondaryWindowOpened);
var htmlToSend = "Nothing";
function on_SecondaryWindowOpened()
{
  yhost.SendMessageToSecondaryWindow(htmlToSend);
}
```

C++ Syntax

```
[id(0x00000005)]
        void SendMessageToSecondaryWindow([in] VARIANT Message);
```

IMessengerActions

The **`IMessengerActions`** interface is implemented by the following objects.

- Messenger

Methods

Name	Description
IMConversation	Start an IM conversation
FileTransfer	Start a file transfer.
WebCam	Start a web cam session.
Conference	Start a conference.
PhotoSharing	Start a photo sharing session.
SendMyContactDetails	Launch the send my contact details UI.
PlayGame	Start a game session.
SendSMSToPhone	Start the process of sending SMS to phone.
SendSMSToFriend	Start the process of sending SMS to a friend.
AddFriend	Adds a friend to the user's buddy list in the specified group
DeleteFriend	Deletes a friend from the user's buddy list.

IMConversation

Use the **`IMConversation`** method to start an Instant Message conversation. Asks user to confirm before the message is sent. When you specify empty or null, it launches the Buddy Picker.

Parameters

Parameter	Description
`Friend`	Friend object. User to start the IM conversation with. If empty, the Buddy Picker is launched.
`MessageText`	The text to send to the other user.

JavaScript Syntax

```
yhost.Messenger.IMConversation(Friend, MessageText);
```

JavaScript Example

Starts a conversation with a friend with the text “Hello!”.

```
m1 = window.external.Messenger;
var MessageTxt = “Hello!”;
var FID = m1.GetFriendFromID(“kat133“);
m1.IMConversation(FID, MessageTxt);
```

C++ Syntax

```
[id(0x00000001)]
        void IMConversation(
                        [in] VARIANT vFriend,
                        [in] VARIANT vMessage);
```

FileTransfer

Use the **`FileTransfer`** method to send a file to another IM user.

Parameters

Parameter	Description
`Friend`	**`String`** or Friend object. User to send the file to. Can be string friend ID or **`Friend`** object. If empty, the Buddy Picker is launched.
`Filelist`	**`String`**. The file to send to the other user.

JavaScript Syntax

```
yhost.Messenger.FileTransfer(Friend, FileList);
```

JavaScript Example

```
yhost.Messenger.FileTransfer("kat133", "ft.txt");
```

C++ Syntax

```
[id(0x00000002)]
        void FileTransfer(
                        [in] VARIANT vFriend,
                        [in] VARIANT files);
```

WebCam

Use the **`WebCam`** method to start a web camera session with another user.

Parameters

Parameter	Description
`Friend`	**`String`** or Friend object. User to start the web camera session with. Can be string friend ID or Friend object. If empty, the Buddy Picker is launched.

JavaScript Syntax

```
yhost.Messenger.WebCam(Friend);
```

JavaScript Example

```
yhost.Messenger.WebCam("kat133");
```

C++ Syntax

```
[id(0x00000004)]
        void WebCam([in] VARIANT vFriend);
```

Conference

Use the **`Conference`** method to start a conference session with another user.

Parameters

Parameter	Description
`Friend`	**`String`** or Friend object. User to start the conference session with. Can be string friend ID or **`Friend`** object. If empty, the Buddy Picker is launched.

JavaScript Syntax

```
yhost.Messenger.Conference(Friend);
```

JavaScript Example

```
yhost.Messenger.Conference("kat133");
```

C++ Syntax

```
[id(0x00000005)]
        void Conference([in] VARIANT vFriend);
```

PhotoSharing

Use the `PhotoSharing` method to start a photo sharing session with another user.

Parameters

Parameter	Description
`Friend`	**`String`** or Friend object. User to start the photo sharing session with. Can be string friend ID or **`Friend`** object. If empty, the Buddy Picker is launched.

JavaScript Syntax

```
yhost.Messenger.PhotoSharing(Friend);
```

JavaScript Example

```
yhost.Messenger.PhotoSharing("kat133");
```

C++ Syntax

```
[id(0x00000006)]
        void PhotoSharing([in] VARIANT vFriend);
```

SendMyContactDetails

Use the `SendMyContactDetails` method to send user's contact details other users.

Parameters

Parameter	Description
`Friends`	String or Friend object. The friend ID or Friend object to send the contact details to. If empty, the Buddy Picker is launched.

JavaScript Syntax

```
yhost.Messenger.SendMyContactDetails(Friends);
```

JavaScript Example

```
yhost.Messenger.SendMyContactDetails("kat133");
```

C++ Syntax

```
[id(0x00000007)]
```

```
        void SendMyContactDetails([in] VARIANT vFriends);
```

PlayGame

Use the `PlayGame` method to start a gaming session with another user.

Parameters

Parameter	Description
`Friend`	**`String`** or Friend object. User to start the gaming session with. Can be string friend ID or **`Friend`** object. If empty, the Buddy Picker is launched.

JavaScript Syntax

```
yhost.Messenger.PlayGame(Friend);
```

JavaScript Example

```
yhost.Messenger.PlayGame(“kat133”);
```

C++ Syntax

```
[id(0x00000008)]
        void PlayGame([in] VARIANT vFriend);
```

SendSMSToPhone

Use the `SendSMSToPhone` method to send an SMS text message to a phone.

Parameters

Parameter	Description
`PhoneNumber`	**`String.`** Phone number to send the text message to.

JavaScript Syntax

```
yhost.Messenger.SendSMSToPhone(PhoneNumber);
```

JavaScript Example

```
yhost.Messenger.SendSMSToPhone(“4087894561”);
```

C++ Syntax

```
[id(0x00000009)]
        void SendSMSToFriend([in] VARIANT vFriend);
```

SendSMSToFriend

Use the `SendSMSToFriend` method to send a SMS text message to another user.

Parameters

Parameter	Description
`Friend`	**`String`** or Friend object. User to start the gaming session with. Can be string friend ID or **`Friend`** object. If empty, the Buddy Picker is launched.

JavaScript Syntax

```
yhost.Messenger.SendSMSToFriend(Friend);
```

JavaScript Example

```
var m1 = yhost.Messenger;
m1.SendSMSToFriend(“kat133”);
```

C++ Syntax

```
[id(0x0000000a)]
```

```
        void SendSMSToPhone([in] VARIANT sPhoneNumber);
```

AddFriend

Use the **`AddFriend`** method to pop up the Add Friend dialog box. The passed parameters preset the dialog box.

Parameters

Parameter	Description
`friendID`	**`String`**. Messenger user ID. If empty, the current user ID is used.
`Group`	Group object. Group the friend will be added to in the buddy list.

JavaScript Syntax

```
yhost.Messenger.AddFriend(ID, Group);
```

JavaScript Example

The following example adds a friend to the specified group in the user's buddy list.

```
function AddFriend(ID, AddGroup)
{
var iter, agroup;
  iter = new Enumerator(yhost.Messenger.Groups);
  for (; !iter.atEnd(); iter.moveNext()  {
    agroup = iter.item();
    if(agroup.Name == AddGroup){
      yhost.Messenger.AddFriend(ID, agroup);
}}}
```

C++ Syntax

```
[id(0x0000000b)]
        void AddFriend(
                        [in] VARIANT friendID,
                        [in] VARIANT group);
```

DeleteFriend

Use the **`DeleteFriend`** method to Pop up the delete friend dialog box so that the user can delete a friend from their buddy list.

Parameters

Parameter	Description
`friendID`	**`String`**. Friend's Messenger user ID.
`Group`	Group object. Group the friend will be removed from in the buddy list.

JavaScript Syntax

```
yhost.Messenger.DeleteFriend(ID, friendGroup);
```

C++ Syntax

```
[id(0x0000000c)]
        void DeleteFriend(
                        [in] VARIANT vFriend,
                        [in] VARIANT group);
```

IParticipants

The **`IParticipants`** interface provides the functionality for the buddy list in the content tab. The **`IParticipants`** Interface is implemented by the following objects:

- ConversationWindow

Properties

Name	Description
CurrentIdentity	**`Returns an`** Identity Object containing information about the user's current identity.
Friends	Collection of Friend objects. Returns the friends participating in the conversation.

CurrentIdentity

The **`CurrentIdentity`** property returns an Identity object that contains the user's information.

JavaScript Syntax

```
yhost.Window.CurrentIdentity;
```

JavaScript Example

The following example displays the name of the user.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tMy Current Identity: " +
      winset.CurrentIdentity.Name + "\n";} }
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT CurrentIdentity();
```

Friends

The **`Friends`** property returns a collection of Friend objects that represent the friends in the users buddy list.

JavaScript Syntax

```
yhost.Window.Friends;
```

JavaScript Example

The following example will display all the participating friends in the buddy list.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));
  for (; !winsetIter.atEnd(); winsetIter.moveNext()){
    winset = winsetIter.item();
    MyFriends = new Enumerator(winset.Friends);
    MyFriend = Myfriends.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tParticipating Friends: " +
      MyFriend.id + "\n";
}}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Friends();
```

IPlugin

The **`IPlugin`** interface is implemented by the following objects.

- Plugin
- PluginHost
- ConversationPluginHost
- ContentTabSecondaryWindowPluginHost
- ContentTabMainWindowPluginHost
- ContentTabPluginCommon

Properties

Name	Description
PluginID	**`String`**. Returns the plug-in ID.
PluginName	**`String`**. Returns the plug-in name.
PluginVersion	**`String`**. Returns the plug-in version.
PluginLocale	**`String`**. Returns the locale of the plug-in.

PluginID

Returns the ID for the plug-in. Corresponds to the **`Id`** entry in the plug-in's manifest file.

JavaScript Syntax

```
yhost.PluginID
```

JavaScript Example

The following example creates an event handler to display a plug-in's information when it is loaded.

```
startevent.Messenger.SetEventHandler("PluginLoaded", OnPluginLoaded)

function OnPluginLoaded(plfrnd, plplugin)
{
  document.eventsForm.triggered.value += "\n" + "Plug-in Loaded Event" +
    "\n" + "Plug-in Loaded By " + plfrnd.id + "\n";
  document.eventsForm.triggered.value += "Plug-in ID: " +
plplugin.PluginID     + "Plug-in Name: " + plplugin.PluginName + "\n";
  document.eventsForm.triggered.value += "Plug-in Version: " +
    plplugin.PluginVersion + "Plug-in Locale: " + plplugin.PluginLocale +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT PluginID();
```

PluginName

Returns the name of the plug-in. This is only defined when called from the plug-in host; in all other cases it will be empty. Corresponds to the **`Name`** entry in the manifest file.

JavaScript Syntax

```
yhost.PluginName;
```

JavaScript Example

The following example creates an event handler to display a plug-in's information when it is loaded.

```
startevent.Messenger.SetEventHandler("PluginLoaded", OnPluginLoaded)

function OnPluginLoaded(plfrnd, plplugin)
{
  document.eventsForm.triggered.value += "\n" + "Plug-in Loaded Event" +
    "\n" + "Plug-in Loaded By " + plfrnd.id + "\n";
  document.eventsForm.triggered.value += "Plug-in ID: " +
plplugin.PluginID     + "Plug-in Name: " + plplugin.PluginName + "\n";
  document.eventsForm.triggered.value += "Plug-in Version: " +
    plplugin.PluginVersion + "Plug-in Locale: " + plplugin.PluginLocale +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT PluginName();
```

PluginVersion

Returns the plug-in version formatted as Major.Minor.Service. Corresponds to the **`Version`** entry in the manifest file.

JavaScript Syntax

yhost.PluginVersion

JavaScript Example

The following example creates an event handler to display a plug-in's information when it is loaded.

```
startevent.Messenger.SetEventHandler("PluginLoaded", OnPluginLoaded)

function OnPluginLoaded(plfrnd, plplugin)
{
  document.eventsForm.triggered.value += "\n" + "Plug-in Loaded Event" +
    "\n" + "Plug-in Loaded By " + plfrnd.id + "\n";
  document.eventsForm.triggered.value += "Plug-in ID: " +
plplugin.PluginID     + "Plug-in Name: " + plplugin.PluginName + "\n";
  document.eventsForm.triggered.value += "Plug-in Version: " +
    plplugin.PluginVersion + "Plug-in Locale: " + plplugin.PluginLocale +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT PluginVersion();
```

PluginLocale

Returns the locale of the plug-in. Defaults to en-us. For example, Locale=en-us. Corresponds to the **`Locale`** entry in the manifest file.

JavaScript Syntax

```
yhost.PluginLocale;
```

JavaScript Example

The following example creates an event handler to display a plug-in's information when it is loaded.

```
startevent.Messenger.SetEventHandler("PluginLoaded", OnPluginLoaded)

function OnPluginLoaded(plfrnd, plplugin)
{
  document.eventsForm.triggered.value += "\n" + "Plug-in Loaded Event" +
    "\n" + "Plug-in Loaded By " + plfrnd.id + "\n";
  document.eventsForm.triggered.value += "Plug-in ID: " +
plplugin.PluginID     + "Plug-in Name: " + plplugin.PluginName + "\n";
  document.eventsForm.triggered.value += "Plug-in Version: " +
    plplugin.PluginVersion + "Plug-in Locale: " + plplugin.PluginLocale +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT PluginLocale();
```

IPluginHost

`IPluginHost` is implemented by the following objects.

- PluginHost
- ConversationPluginHost
- ContentTabSecondaryWindowPluginHost
- ContentTabMainWindowPluginHost
- ContentTabPluginCommon

Properties

Name	Description
Messenger	Returns a Messenger object.
AddressBook	Returns AddressBook object containing the user's address book. Returns null if access is restricted.
Cookies	Returns the messenger cookies. Returns null if plug-in was not loaded from the Yahoo! domain.
LastError	`Error` object. Returns the last error that occurred.

Methods

Name	Description
SendHTTPRequest	Returns the content of the provided URL.
SendHTTPRequestEx	Fetches and returns the content of the provided URL and optional headers
OpenURL	Opens the provided URL in a separate window.
LocalReady	Tells the host that the plug-in is ready.
ShowFileBrowserDialog	Launches the Show File Browser Dialog box.
ShowBuddyPicker	Launches the Buddy Picker dialog box.

Messenger

Returns a Messenger object to access the user's address book, buddy list, cookies, and instant messaging.

JavaScript Syntax

```
yhost.Messenger;
```

JavaScript Example

The following function gets all online friends from the buddy list and sends them an instant message.

```
function SendPluginMessage_Call (msg)
{
  var iter, friend;

  iter = new Enumerator(yhost.Messenger.FriendsOnline);

  for (; !iter.atEnd(); iter.moveNext())
  {
    friend = iter.item();
    yhost.SendPluginMessage(msg, friend);
    log_last_error("SendPluginMessage");
  }
}
```

C++ Syntax

```
[id(0x00000001), propget]
  VARIANT Messenger();
```

AddressBook

Returns an AddressBook object of the current user, if allowed. This is only allowed if the AddressBook entry in the manifest file is set to True.

JavaScript Syntax

```
yhost.AddressBook;
```

JavaScript Example

The following function attempts to get the address book and checks to see if it was successful.

```
function AddressBook_Call ()
{
  ab = yhost.AddressBook;
  if (ab == null)
    AddressBook_Res.innerHTML = "result: null";
  else
    AddressBook_Res.innerHTML = "result: valid obj.";
}
```

C++ Syntax

```
[id(0x00000002), propget]
  VARIANT AddressBook();
```

Cookies

If plug-in was loaded from the Yahoo! domain, returns the Messenger cookies for this session.

JavaScript Syntax

```
yhost.Cookies
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT Cookies();
```

LastError

Returns an Error object for the last occurring error.

JavaScript Syntax

```
yhost.LastError;
```

JavaScript Example

The following example checks and displays the last occurring error.

```
if (yhost.LastError.ErrorID != "NoError")
{
  append_log_msg (method_name + " error: '" + yhost.LastError.ErrorID +
  "'");
}
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT Cookies();
```

SendHTTPRequest

Fetches and returns the content of the specified URL. When completed, a HTTPRequestCompleted event is fired. If an error occurs, an HTTPRequestError event is fired.

SendHTTPRequest is asynchronous, so you may make multiple calls to SendHTTPRequest before the first one completes, provided a different RequestID is specified for each call. Otherwise, only the response from the last request of the RequestID is returned.

The URL fetched must use UTF-8 characters, otherwise, the fetched information will be incorrect.

Parameters

Parameter	Description
`RequestID`	**`String`**. Plug-in related request ID returned as an event parameter on completion.
URL	**`String`**. URL to get. URL scheme must be specified: **`ftp://`** or **`http://`**

JavaScript Syntax

```
yhost.SendHTTPRequest(RequestID, URL);
```

JavaScript Example

Get a weather RSS feed for Sunnyvale California from xml.weather.yahoo.com.

```
yhost = window.external;

function On_HTTPRequestCompleted(id, textstring) {
  // do stuff with the textstring }

function On_HTTPRequestError(id, errortype) {
  alert("Failed to connect!"); }

yhost.SetEventHandler("HTTPRequestCompleted", On_HTTPRequestCompleted);
yhost.SetEventHandler("HTTPRequestError", On_HTTPRequestError);

yhost.LocalReady();
// gets RSS feed for Sunnyvale, CA
var xmlhttp = yhost.SendHTTPRequest(1,
"http://xml.weather.yahoo.com/forecastrss?p=94089");
```

C++ Syntax

```
[id(0x00000005)]
        void SendHTTPRequest(
                        [in] VARIANT sRequestID,
                        [in] VARIANT sURL);
```

SendHTTPRequestEx

Fetches and returns the content of the provided URL and optional headers. When completed, a HTTPRequestCompleted event is fired. If an error occurs, an HTTPRequestError event is fired.

SendHTTPRequestEX is asynchronous, so you may make multiple calls to SendHTTPRequestEX before the first one completes, provided a different RequestID is specified for each call. Otherwise, only the response from the last request of the RequestID is returned.

The URL fetched must use UTF-8 characters, otherwise, the fetched information will be incorrect.

Parameters

Parameter	Description
RequestID	**String**. Plug-in supplied ID used to identify the request in events.
RequestMethod	**String**. HTTP method to use for the request. Possible values: GET POST
URL	**String**. URL to get. URL scheme must be specified: **ftp://** or **http://**
PostData	**String**. For POST requests, this must be a non-empty string. Sent as message body. **Please note**: Content-Type header will NOT be automatically sent. You need to set it explicitly.
RequestHeaders	**String**. Should be NULL. Headers to send with the request in format "Name: Value", separated by CRFL characters ("\r\n" strings in JavaScript). The call generates some headers (such as Content-Length and "Pragma: no-cache") automatically; it is possible to replace them using this parameter. You can also add new headers.
AskForResponseHeaders	**Boolean**. Set this parameter to True to have response headers returned.
AllowRedirect	**Boolean**. Set to True to handle redirects transparently to the caller. Set to False to return **3xx** redirect status codes in **_IEventsPluginHost::HTMLGetRequestError** event.

JavaScript Syntax

```
yhost.SendHTTPRequestEx(RequestID, RequestMethod, URL, PostData,
RequestHeaders, AskForResponseHeaders, AllowRedirect);
```

C++ Syntax

```
[id(0x0000000a)]
        void SendHTTPRequestEx(
                        [in] VARIANT sRequestID,
                        [in] VARIANT sMethod,
                        [in] VARIANT sURL,
                        [in] VARIANT vPostData,
                        [in] VARIANT vRequestHeaders,
                        [in] VARIANT vReturnResponseHeaders,
                        [in] VARIANT vAllowRedirect);
```

OpenURL

Opens a URL in a separate window.

Parameters

Parameter	Description
`URL`	**`String`**. The URL to open. If you provide the relative file path, the file is opened from the directory (for local plug-ins) or URL (for remote plug-ins) where the main plug-in page is located. Do not specify “file://” for a local file—only the relative path. The “file://” directory path format is not allowed. If you provide a URL, it will attempt to load without additional manipulation.
`Login`	**`Boolean`**. If True, performs automatic login to URL.

JavaScript Syntax

```
yhost.OpenURL(URL, Login);
```

JavaScript Example

Goes to the website custom.autos.yahoo.com without logging the user in.

```
yhost.OpenURL('http://custom.autos.yahoo.com/', false)
```

C++ Syntax

```
[id(0x00000003)]
        void OpenURL(
                        [in] VARIANT sURL,
                        [in] VARIANT bLogin);
```

LocalReady

LocalReady notifies other plug-in instances that the plug-in host has been created and is ready for input. Until **`LocalReady`** is called, local host will not receive any event notifications.

For instance, when a secondary window is launched, it must call **`LocalReady`** to let the main plug-in window know that it exists and is ready to communicate. Likewise, Conversation plug-ins call LocalReady to notify the other user's plug-in instance that it exists and is ready to communicate.

JavaScript Syntax

```
yhost.LocalReady();
```

JavaScript Example

Set some event handlers and notify the host that the plug-in is ready.

```
var yahoo = null;

function initYahoo()
{
  yahoo = window.external;
  yahoo.SetEventHandler('SecondaryWindowReady',
    onSecondaryWindowReady);
  yahoo.SetEventHandler('SecondaryWindowClosed',
    onSecondaryWindowClosed);
  yahoo.SetEventHandler('SecondaryWindowMessage',
    onSecondaryWindowMessage);

  yahoo.LocalReady();
}
```

C++ Syntax

```
[id(0x00000006)]
        void LocalReady();
```

ShowFileBrowserDialog

Launches the File Open dialog box. Returns the file path selected by the user as a string.

Parameters

Parameter	Description
`Open`	Boolean. Specify True for Open, False to SaveAs.
`Path`	**`String`**. Initial directory path. If empty, the initial directory will be according to Win32 GetOpenFile function rules.

JavaScript Syntax

```
yhost.ShowFileBrowserDialog(Open, Path, SelectedFilePath);
```

JavaScript Example

Open the file browser dialog and get a file path from the user.

```
var filePath

filePath = yhost.ShowFileBrowserDialog(true, "c:\\program files");
ShowFileBrowserDialog_Res.innerHTML = filePath;
```

C++ Syntax

```
[id(0x00000008)]
        VARIANT ShowFileBrowserDialog(
                        [in] VARIANT bOpen,
                        [in] VARIANT sInitalDir);
```

ShowBuddyPicker

Launches the Buddy Picker dialog box. Returns a collection of Friend objects for the friends selected by the user.

Parameters

Parameter	**Description**
Title	**`String`**. Text to appear in the title bar.
Action	**`String`**. Text to appear on the dialog to instruct the user.

JavaScript Syntax

```
yhost.ShowBuddyPicker(Title, Action)
```

JavaScript Example

Gets a list of friends to play a game with.

```
var myGamingFriends = yhost.ShowBuddyPicker(“Play a game”, "Select a buddy
to play a game with, and then click OK.");
```

C++ Syntax

```
[id(0x0000000b)]
        VARIANT ShowBuddyPicker(
                        [in] VARIANT sTitle,
                        [in] VARIANT sAction);
```

IPluginStorage

The `IPluginStorage` interface is implemented by the following objects.

- PluginHost
- ConversationPluginHost
- ContentTabSecondaryWindowPluginHost
- ContentTabMainWindowPluginHost
- ContentTabPluginCommon

Methods

Name	Description
StorageWrite	Writes a key and its value to storage.
StorageRead	Reads the value for a given key.
StorageRemove	Removes a key and its value from storage.

StorageWrite

Writes a key and its value to storage. Data is written to a file, one each per plug-in. The number of entries is limited only by disk size. If fails, `LastError` will return "PluginStorageWriteFail".

Parameters

Parameter	Description
`Key`	`String`. Key to a storage value.
Value	`String`. Value to be stored. Maximum is 2K bytes per key.

JavaScript Syntax

```
yhost.StorageWrite(Key, Value);
```

JavaScript Example

```
<tr class="interface_method">
  <td><input value="call" type="button" onclick="yhost.StorageWrite
    (StorageWrite_Arg1.value, StorageWrite_Arg2.value);"></td>
  <td>StorageWrite</td>
  <td><input id="StorageWrite_Arg1" value="key" size="10"
    type="text"></td>
  <td><input id="StorageWrite_Arg2" value="value" size="10"
    type="text"></td>
</tr>
```

C++ Syntax

```
[id(0x00000001)]
        void StorageWrite(
                        [in] VARIANT key,
                        [in] VARIANT value);
```

StorageRead

Returns the value from storage given a key. Data is stored in a file, one each per plug-in. If fails, **`LastError`** will return **`PluginStorageReadFail`**.

Parameters

Parameter	Description
`Key`	**`String`**. Key with a storage value.

JavaScript Syntax

```
yhost.StorageRead(Key);
```

JavaScript Example

Reads a value from storage.

```
<tr class="interface_method">
  <td><input value="call" type="button" onclick="javascript:
    StorageRead_Call(StorageRead_Arg1.value)"></td>
  <td>StorageRead</td>
  <td><input id="StorageRead_Arg1" value="key" size="10"
    type="text"></td>
  <td id="StorageRead_Res"></td>
</tr>

function StorageRead_Call (key)
```

```
{
  val = yhost.StorageRead (key);
  if (val != null)
    StorageRead_Res.innerHTML = "result: '" + val + "'";
  else
    StorageRead_Res.innerHTML = "";
}
```

C++ Syntax

```
[id(0x00000002)]
        VARIANT StorageRead([in] VARIANT key);
```

StorageRemove

Removes a key and it value from storage. Data is stored in a file, one each per plug-in.

Parameters

Parameter	Description
`Key`	**`String`**. Key to a storage value.

JavaScript Syntax

```
yhost.StorageRemove(Key);
```

JavaScript Example

When the user clicks on the button, the specified key and value are remove from storage.

```
<tr class="interface_method">
  <td><input value="call" type="button" onclick="yhost.StorageRemove
    (StorageRemove_Arg1.value);"></td>
  <td>StorageRemove</td>
  <td><input id="StorageRemove_Arg1" value="key" size="10"
    type="text"></td>
</tr>
```

C++ Syntax

```
[id(0x00000003)]
        void StorageRemove([in] VARIANT key);
```

ISecondaryWindow

The ISecondaryWindow interface is used to open a secondary window from the main window of a Tab plug-in.

The **`ISecondaryWindow`** interface is implemented by the following object:

- ContentTabSecondaryWindowPluginHost

Methods

Name	Description
SecondaryWindowCloseMyself	Secondary window closes itself.
SendMessageToMainPluginWindow	Secondary window sends a message to main plug-in window.

SecondaryWindowCloseMyself

The secondary window closes itself.

JavaScript Syntax

```
yhost.SecondaryWindowCloseMyself();
```

C++ Syntax

```
[id(0x00000001)]
        void SecondaryWindowCloseMyself();
```

SendMessageToMainPluginWindow

Secondary window sends a message to the main plug-in window. While the main window and secondary windows are one plug-in, an HTML file displayed in the secondary window must call the **`LocalReady`** method independently in order to communicate with the main window.

Parameters

Parameter	Description
`msg`	**`String`**. Message to send to main plug-in window.

JavaScript Syntax

```
yhost.SendMessageToMainPluginWindow(msg);
```

C++ Syntax

```
[id(0x00000002)]
        void SendMessageToMainPluginWindow([in] VARIANT Message);
```

IStatus

The **`IStatus`** interface is implemented by the following objects.

- Status

Properties

Name	Description
Status	**`String`**. Returns the status ID of the user or friend.
StatusLocalized	**`String`**. Returns the status of the user or friend in the language of their locale.
DND	**`String`**. Returns the DND (Do Not Disturb) status ID of the user or friend.
Data	**`String`**. Returns the status description text for the user or friend.
LinkType	**`String`**. Returns the type of status link, if any.
Message	**`String`**. Returns the status message.
IdleTime	**`String`**. Returns the amount of time in seconds that the user or friend has been idle.

Methods

Name	Description
SetCustomStatus	Sets the **`Custom`** status.

Status

Returns and writes the status of the user as a string as it is displayed to other users. The write operation is only supported when the Status object obtained from the Me object. For example,

```
yhost.Messenger.Friend.Status.Status = “OnLunch”;
```

IStatus
Status

Possible values are:

Read	**Write**
Offline	Offline
Online	Online
BeRightBack	BeRightBack
Busy	Busy
NotAtHome	NotAtHome
NotAtDesk	NotAtDesk
NotInOffice	NotInOffice
OnVacation	OnVacation
OnPhone	OnPhone
OnLunch	OnLunch
SteppedOut	SteppedOut
Idle	Idle
Custom	

JavaScript Syntax

```
Friend.Status.Status;
```

JavaScript Example

The following example displays the status ID of the

```
function DisplayStatus() {
  var lstatus, frndstat;

  frndstat = new Enumerator(m1.Friends);
  lstatus = frndstat.item();

  for (; !frndstat.atEnd(); frndstat.moveNext())  {

    document.MEForm.detail.value += "\tStatus :" + lstatus.Status.Status +
"\n";
} }
```

C++ Syntax

```
[id(0x00000001), propget]
```

```
        VARIANT Status();
[id(0x00000001), propput]
        void Status([in] VARIANT rhs);
```

StatusLocalized

Returns the status of the user as a string in their locale's language.

JavaScript Syntax

```
Friend.Status.StatusLocalized;
```

JavaScript Example

The following example displays the localized status of a friend.

```
function DisplayStatus()
{
  var lstatus, frndstat;

  frndstat = new Enumerator(m1.Friends);
  lstatus = frndstat.item();

  for (; !frndstat.atEnd(); frndstat.moveNext())
  {

    document.MEForm.detail.value += "\tStatus :" + lstatus.Status.Status +
      "\n";
    document.MEForm.detail.value += "\tStatus Localized: " +
      lstatus.Status.StatusLocalized + "\n";
} }
```

C++ Syntax

```
[id(0x00000005), propget]
        VARIANT StatusLocalized();
```

DND

Returns the DND (Do Not Disturb) status ID. Possible values are “Available”, “DND”, and “Idle”. This is an additional status associated with the status ID that indicates the reachability of the user.

JavaScript Syntax

```
Status.DND;
```

JavaScript Example

The following example

```
function DisplayStatus()
{
  var lstatus, frndstat;

  frndstat = new Enumerator(m1.Friends);
  lstatus = frndstat.item();

  for (; !frndstat.atEnd(); frndstat.moveNext())
  {

    document.MEForm.detail.value += "\tStatus :" + lstatus.Status.Status +
      "\n";
    document.MEForm.detail.value += "\tStatus DND: " + lstatus.Status.DND
      + "\n";
} }
```

C++ Syntax

```
[id(0x00000006), propget]
        VARIANT DND();
```

Data

Returns the status description text for the user or friend.

JavaScript Syntax

```
Friend.Status.Data
```

JavaScript Example

The following example displays the status and status text of each friend in the buddy list.

```
function DisplayStatus()
{
  var lstatus, frndstat;

  frndstat = new Enumerator(m1.Friends);
  lstatus = frndstat.item();

  for (; !frndstat.atEnd(); frndstat.moveNext())
  {

    document.MEForm.detail.value += "\tStatus :" + lstatus.Status.Status +
      "\n";
    document.MEForm.detail.value += "\tStatus DND: " + lstatus.Status.DND
      + "\n";
    document.MEForm.detail.value += "\tStatus Data: " +
      lstatus.Status.Data + "\n";
} }
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT Data();
```

LinkType

Returns as a string the type of link in the user’s or friends status. Possible values are

None	Game	Custom
Webcam	OpenTalk	LaunchCast

JavaScript Syntax

```
Friend.Status.LinkType;
```

JavaScript Example

The following example displays the status of each friend and any link type associate with it.

```
function DisplayStatus()
{
var liststatus, lstatus;
var frndid, fid;
liststatus = new Enumerator(m1.FriendsOnline);
frndstat = new Enumerator(m1.Friends);

for (; !frndstat.atEnd(); frndstat.moveNext())
{

fid = frndstat.item();
document.MEForm.detail.value += "\tStatus :" + lstatus.Status.Status +
"\n";
document.MEForm.detail.value += "\tStatus Data: " + lstatus.Status.Data +
"\n";
document.MEForm.detail.value += "\tStatus LinkType: " +
lstatus.Status.LinkType + "\n";
document.MEForm.detail.value += "\tStatus Link: " + lstatus.Status.Link +
"\n";
} }
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT LinkType();
```

Message

Returns the status message, provided in the user's or friend's status.

JavaScript Syntax

```
Friend.Status.Message;
```

JavaScript Example

The following example displays the status of each friend and any associated message.

```
function DisplayStatus()
{
  var lstatus, frndstat;

  frndstat = new Enumerator(m1.Friends);
  lstatus = frndstat.item();

  for (; !frndstat.atEnd(); frndstat.moveNext())
  {

    document.MEForm.detail.value += "\tStatus :" + lstatus.Status.Status +
      "\n";
    document.MEForm.detail.value += "\tStatus DND: " + lstatus.Status.DND
      + "\n";
    document.MEForm.detail.value += "\tStatus Data: " +
      lstatus.Status.Data + "\n";
    document.MEForm.detail.value += "\tStatus LinkType: " +
      lstatus.Status.LinkType + "\n";
    document.MEForm.detail.value += "\tStatus Link: " +
      lstatus.Status.Message + "\n";
} }
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT Message();
```

IdleTime

Returns the amount of time in seconds that the user or friend has been idle.

JavaScript Syntax

```
Friend.Status.IdleTime;
```

JavaScript Example

The following example displays the status of each friend and the amount of time they have been idle.

```
function DisplayStatus()
{
```

```
var lstatus, frndstat;

frndstat = new Enumerator(m1.Friends);
lstatus = frndstat.item();

for (; !frndstat.atEnd(); frndstat.moveNext())
{
  document.MEForm.detail.value += "\tStatus :" + lstatus.Status.Status +
    "\n";
  document.MEForm.detail.value += "\t Status Idle Time: " +
    lstatus.Status.IdleTime + "\n";
 } }
```

C++ Syntax

```
[id(0x00000007), propget]
        VARIANT IdleTime();
```

SetCustomStatus

Sets the **`Custom`** status. **`SetCustomStatus`** is only supported when the Status object obtained from the Me object.

Parameters

Parameter	Description
`Message`	**`String`**. Status message.
`Busy`	**`Boolean`**. If True, busy; if False, available.
`Reserved1`	**`String`**. Reserved for future use. Must be empty.
`Reserved2`	**`String`**. Reserved for future use. Must be empty.

JavaScript Syntax

```
yhost.Messenger.Me.Status.SetCustomStatus(Message,Busy, , ,);
```

JavaScript Syntax

```
yhost.Messenger.Me.Status.SetCustomStatus(“Out to lunch, call my
cell!”,True, , ,);
```

C++ Syntax

```
[id(0x00000008)]
        void SetCustomStatus(
                        [in] VARIANT mes,
                        [in] VARIANT busy,
                        [in] VARIANT res1,
                        [in] VARIANT res2);
```

IToast

The **`IToast`** interface is implemented by the following objects.

- Messenger

Methods

Name	Description
ToastShow	Pops up the toast window.
ToastClose	Closes the toast window.

ToastShow

Pops up the toast window for a specified amount of time containing the specified text.

Parameters

Parameter	Description
`ID`	**`String.`** Toast ID. Can uniquely identify this particular toast. Can be used to close the toast and receive toast events.
`Time`	**`Numeric`**. Time in seconds to show the toast.
`Text`	**`String`**. HTML text to put in the toast web control.

JavaScript Syntax

```
myHost.ToastShow(ID, Time, Text);
```

JavaScript Example

When a friend is selected in the user's buddy list, show a toast message.

```
yhost = window.external;
yhost.LocalReady();
var mainWindow = yhost.Messenger.MainWindow;
mainWindow.SetEventHandler("FriendSelected", onFriendSelected);
function onFriendSelected(friend) {
  yhost.ToastShow("friendSelected", 5, "Content tab has noticed that you
clicked on: "+friend.ID);
     }
```

C++ Syntax

```
[id(0x00000001)]
        void ToastShow(
                        [in] VARIANT ToastID,
                        [in] VARIANT TimeSec,
                        [in] VARIANT html);
```

ToastClose

Closes the toast window.

Parameters

Parameter	Description
`ID`	**`String`**. Toast ID specified in ToastShow() function. If multiple toasts were opened using the same ID, Messenger will try to close them all.

JavaScript Syntax

```
yhost.ToastClose(ID);
```

C++ Syntax

```
[id(0x00000002)]
        void ToastClose([in] VARIANT ToastID);
```

IVoiceActions

IVoiceActions provides functionality to send text messages to a phone or computer. The **`IVoice`** interface is implemented by the following objects.

- IVoiceActions

Methods

Name	Description
CallPSTN	Call a PSTN number.
CallFriendComputer	Call a friend's computer.

CallPSTN

Call a PSTN number.

Parameters

Parameter	Description
`Number`	**`String`**. Phone number to call.

JavaScript Syntax

```
myHost.CallPSTN(number );
```

C++ Syntax

```
[id(0x00000001)]
        void CallPSTN([in] VARIANT sNumber);
```

CallFriendComputer

Calls the selected friend's computer.

Parameters

Parameter	Description
`friend`	Friend object representing the friend to call.

JavaScript Syntax

```
myHost.CallFriendComputer(friend);
```

C++ Syntax

```
[id(0x00000002)]
        void CallFriendComputer([in] VARIANT vFriend);
```

IWindow

The **`IWindow`** interface provides basic window functionality for content tab, conversation, and text message windows. The **`IWindow`** interface is implemented by the following objects.

- MainWindow
- ConversationWindow

Properties

Name	**Description**
WinOpen	**`Boolean`**. Returns true if the window is opened or closed.
WinVisible	**`Boolean`**. Returns true if the window is visible.
WinX	**`Numeric`**. Returns the top left corner horizontal position in pixels.
WinY	**`Numeric`**. Returns the top left corner vertical position in pixels.
WinWidth	**`Numeric`**. Returns the width of the window in pixels.
WinHeight	**`Numeric`**. Returns the height of the window in pixels.
WinHWND	**`Numeric`**. Returns a window handler for the window.
WinTitle	**`String`**. Returns the window title.
WinForeground	**`Boolean`**. Returns true if the window is in the foreground.
WinActive	**`Boolean`**. Returns true if the window has mouse of keyboard focus.
WinMinimized	**`Boolean`**. Returns true if the window is minimized.
WinMaximized	**`Boolean`**. Returns true if the window is maximized.

WinOpen

Use the **`WinOpen`** property to see if the window is open. Returns True if window is open.

JavaScript Syntax

```
myMessenger.MainWindow.WinOpen;
```

JavaScript Example

The following example displays whether the window is open.

```
function DisplayConversationWindowInfo()
{
var winset, winConversation, frnds, fpart,

  winConversation = new Enumerator(m1.Windows("Conversation"));

  for (; !winConversation.atEnd(); winConversation.moveNext()){

    winset = winConversation.item();

    document.MEForm.detail.value += "\tTitle: " + winset.WinTitle + "\n"
    document.MEForm.detail.value += "\tOpen: " + winset.WinOpen + "\n"
}}
```

C++ Syntax

```
[id(0x00000001), propget]
        VARIANT WinOpen();
```

WinVisible

Use the **`WinVisible`** method to see if the window is visible.

JavaScript Syntax

myMessenger.*WinObj*`.WinVisible;`

JavaScript Example

The following example displays whether the window is visible.

```
function DisplayConversationWindowInfo()
{
var winset, winConversation, frnds, fpart,

winConversation = new Enumerator(m1.Windows("Conversation"));

for (; !winConversation.atEnd(); winConversation.moveNext()){

  winset = winConversation.item();
  frnds = new Enumerator(winset.Friends);

  for (; !frnds.atEnd(); frnds.moveNext()){
    fpart = frnds.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation Window
      Object"  + "\n"
    document.MEForm.detail.value += "\tIs Window Visible? " +
      winset.WinVisible + "\n"
}}
```

C++ Syntax

```
[id(0x0000000c), propget]
        VARIANT WinVisible();
```

WinX

Returns the numeric value for the top left corner horizontal position of the window in pixels.

JavaScript Syntax

myMessenger.*WinObj*`.WinX;`

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays the X and Y location of the window.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tX: " + winset.WinX + "\n"
    document.MEForm.detail.value += "\tY: " + winset.WinY  + "\n"

} }
```

C++ Syntax

```
[id(0x00000002), propget]
        VARIANT WinX();
```

WinY

Returns the numeric value for the top left corner horizontal position of the window in pixels.

JavaScript Syntax

```
myMessenger.WinObj.WinY;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays the X and Y location of the window.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tX: " + winset.WinX + "\n"
    document.MEForm.detail.value += "\tY: " + winset.WinY  + "\n"

} }
```

C++ Syntax

```
[id(0x00000003), propget]
        VARIANT WinY();
```

WinWidth

Returns the windows width in pixels.

JavaScript Syntax

```
myMessenger.WinObj.WinWidth;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays the window’s height and width.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tWidth: " + winset.WinWidth +
      "\n"
    document.MEForm.detail.value += "\tHeight: " + winset.WinHeight +
      "\n"
} }
```

C++ Syntax

```
[id(0x00000004), propget]
        VARIANT WinWidth();
```

WinHeight

Returns the windows height in pixels.

JavaScript Syntax

```
myMessenger.WinObj.WinHeight;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays the window height and width.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tWidth: " + winset.WinWidth +
      "\n"
    document.MEForm.detail.value += "\tHeight: " + winset.WinHeight +
      "\n"} }
```

C++ Syntax

```
[id(0x00000005), propget]
        VARIANT WinHeight();
```

WinHWND

Returns the numeric value of the event handler for the window.

JavaScript Syntax

```
myWindow.WinObj.WinHWND;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays its Window Handler.

```
function DisplayConversationWindowInfo()
```

```
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tWindow Handler #: " +
      winset.WinHWND + "\n";
} }
```

C++ Syntax

```
[id(0x00000006), propget]
        VARIANT WinHWND();
```

WinTitle

Use **`WinTitle`** to get the window title of the window.

JavaScript Syntax

`myWindow``.`*`WinObj`*`.WinTitle;`

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays the window title.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
```

```
      Window Object"  + "\n"
       document.MEForm.detail.value += "\tTitle: " + winset.WinTitle +
      "\n";
} }
```

C++ Syntax

```
[id(0x00000007), propget]
        VARIANT WinTitle();
```

WinForeground

Returns True if the window is in the foreground.

JavaScript Syntax

```
myWindow.WinObj.Foreground;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays its foreground status.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tForeground: " +
      winset.WinForeground + "\n"
} }
```

C++ Syntax

```
[id(0x00000008), propget]
        VARIANT WinForeground();
```

WinActive

Returns True if the window is active.

JavaScript Syntax

```
myWindow.WinObj.WinActive;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays whether the window is active.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
    document.MEForm.detail.value += "\tActive: " + winset.WinActive +
      "\n"
} }
```

C++ Syntax

```
[id(0x00000009), propget]
        VARIANT WinActive();
```

WinMinimized

Returns True if the window is minimized.

JavaScript Syntax

```
myWindow.WinObj.WinMinimized;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays whether the window is minimized or maximized.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;

  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
     document.MEForm.detail.value += "\tMinimized: " +
      winset.WinMinimized + "\n"
    document.MEForm.detail.value += "\tMaximized: " +
      winset.WinMaximized + "\n"
} }
```

C++ Syntax

```
[id(0x0000000a), propget]
        VARIANT WinMinimized();
```

WinMaximized

Returns True if the window is maximized.

JavaScript Syntax

```
myWindow.WinObj.WinMaximized;
```

JavaScript Example

The following example gets all the current conversation windows in this messenger session and displays whether the window is minimized or maximized.

```
function DisplayConversationWindowInfo()
{
  var winset, winsetIter;
  winsetIter = new Enumerator(m1.Windows("Conversation"));

  for (; !winsetIter.atEnd(); winsetIter.moveNext()){

    winset = winsetIter.item();
    document.MEForm.detail.value += "ICreatedWindows Conversation
      Window Object"  + "\n"
     document.MEForm.detail.value += "\tMinimized: " +
      winset.WinMinimized + "\n"
    document.MEForm.detail.value += "\tMaximized: " +
      winset.WinMaximized + "\n"
} }
```

C++ Syntax

```
[id(0x0000000b), propget]
        VARIANT WinMaximized();
```

_IEventsAddressBook

The **`_IEventsAddressBook`** interface is implemented by the following objects:

- AddressBook

Events

Name	Description
ABEntryRemoved	Fired when an address book entry is removed from the user's address book.
ABEntryUpdated	Fired when an address book entry is updated in the user's address book.
ABUpdated	Fired when the whole address book is updated.

ABEntryRemoved

Fired when an address book entry is removed from the user's address book

Parameters

Parameter	Description
`Entry`	**`String`**. Unique ID of the removed address book entry

JavaScript Syntax

```
yhost.AddressBook.SetEventHandler("ABEntryRemoved", On_ABEntryRemoved)
function On_ABEntryRemoved(ABRemove)
```

JavaScript Example

Create an event to handle when an address book entry is removed from the address book.

```
yhost.AddressBook.SetEventHandler("ABEntryRemoved",
On_ABEntryRemoved)
function On_ABEntryRemoved(ABRemove)
```

```
{
  document.eventsForm.triggered.value += "Address Book Event " + "Address
    Book Entry Removed: " + ABRemove.Friend + "Unique ID: " + ABRemove +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT ABEntryRemoved([in] VARIANT abEntry);
```

ABEntryUpdated

Fired when a new address book entry is created or and existing entry is updated.

Parameters

Parameter	Description
`Entry`	AddressBookEntry object. The address book entry that was updated.

JavaScript Syntax

```
yhost.AddressBook.SetEventHandler("ABEntryUpdated", On_ABEntryUpdated)
function On_ABEntryUpdated(ABRemove)
```

JavaScript Example

Create an event handler for when an address book entry is updated.

```
startevent.AddressBook.SetEventHandler("ABEntryUpdated",
On_ABEntryUpdated)

function On_ABEntryUpdated(ABUpdate)
{
  document.eventsForm.triggered.value += "Address Book Entry Updated" +
    "\n";
  document.eventsForm.triggered.value += "First/Middle/Last Name: " +
  document.eventsForm.triggered.value += "First/Middle/Last Name: " +
    ABUpdate.FirstName + ABUpdate.MiddleName + ABUpdate.LastName +
    ABUpdate.NickName + "\n";
}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT ABEntryUpdated([in] VARIANT abEntry);
```

ABUpdated

Fired when the whole address book is updated.

JavaScript Syntax

```
yhost.AddressBook.SetEventHandler("ABUpdated", On_ABUpdated)
function On_ABUpdated()
```

JavaScript Example

Creates an event handler for when the whole address book is updated.

```
yhost.AddressBook.SetEventHandler("ABUpdated", On_ABUpdated)

function On_ABUpdated()
{
  document.eventsForm.triggered.value += "Address Book Event " + "Entire
    AB Updated" + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT ABUpdated();
```

_IEventsColors

The **`_IEventsColors`** interface is implemented by the following objects.

- PluginHost
- ContentTabPluginCommon
- ContentTabMainWindowPluginHost
- ContentTabSecondaryWindowPluginHost
- ConversationPluginHost

Events

Name	Description
ColorsChanged	Fired when the color scheme changes.

ColorsChanged

Fired when the color scheme changes.

JavaScript Syntax

```
yhost.SetEventHandler("ColorsChanged", onColorsChanged);
```

JavaScript Example

Creates and event handler to update the plug-in’s main window and secondary window (if any) background color when the user changes their Messenger appearance settings.

```
yhost.SetEventHandler('ColorsChanged', on_UpdateColors);

function on_ColorsChanged);
{
  document.body.style.backgroundColor = yhost.ColorsBackground;

  if (g_currentSecondaryWindow != null)
  document.body.style.backgroundColor =
    g_currentSecondaryWindow.ColorsBackground;
}
```

_IEventsColors
ColorsChanged

C++ Syntax

```
[id(0x00000001)]
        HRESULT ColorsChanged();
```

_IEventsContactList

The **`IEventsContactList`** interface is implemented by the following objects:

- Messenger

Events

Name	Description
FriendAddedToGroup	Fired when new user is added to a group in the buddy list.
FriendDeletedFromGroup	Fired when friend is deleted from group in the buddy list.
FriendChanged	Fired when a friend in the buddy list's information changes.
GroupAdded	Fired when a new group is added to buddy list.
GroupDeleted	Fired when a group is deleted from the buddy list.
GroupRenamed	Fired when a group in the buddy list is renamed.
PluginLoaded	Fired when the plug-in is loaded.
PluginUnloaded	Fired when the plug-in is unloaded.

FriendAddedToGroup

Set an event handler for **`FriendAddedToGroup`** to be notified when a friend is added to a group in the user's buddy list.

Parameters

Parameter	Description
`Friend`	Friend object. The friend added to the buddy list.
`Group`	Group object. The group to which the friend was added.

JavaScript Syntax

```
MyMessenger.SetEventHandler("FriendAddedToGroup", OnFriendAddedToGroup)
function OnFriendAddedToGroup(fafrnd, fagroup)
```

JavaScript Example

```
Messenger.SetEventHandler("FriendAddedToGroup", OnFriendAddedToGroup)

function OnFriendAddedToGroup(fafrnd, fagroup)
{
  document.eventsForm.triggered.value += "\n" + "Contact List Event" +
    "\n" + "Friend Added To Group: ";
  document.eventsForm.triggered.value += "Group Name & Friend ID: " +
    fagroup.name + fafrnd.id + "\n";
  document.eventsForm.triggered.value += "Session ID: " + "(" +
    fagroup.UniqueSessionID + ")" + "\n";
  document.eventsForm.triggered.value += "FirstName LastName " +
    fafrnd.FirstName + " " + fafrnd.LastName + "\n";
  document.eventsForm.triggered.value += "Status: " +
    fafrnd.Status.Status + " " + "Localized: " + fafrnd.Status.Localized +
" " + "DND: " + fafrnd.Status.DND + " " + "\n";
  document.eventsForm.triggered.value += "Data: " + fafrnd.Status.Data + "
    " + "Link Type: " + fafrnd.Status.LinkType + " " + "Link: " +
    fafrnd.Status.Link + " " + "Idle Time: " + fafrnd.Status.IdleTime +
    "\n";
  document.eventsForm.triggered.value += "Image : " + fafrnd.image + "\n";
  document.eventsForm.triggered.value += "PeerTOPeer : " +
    fafrnd.PeerToPeer + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT FriendAddedToGroup(
                        [in] VARIANT vFriend,
                        [in] VARIANT vGroup);
```

FriendDeletedFromGroup

Create an event handler for **`FriendDeletedFromGroup`** to be notified when a friend is deleted from the user’s buddy list.

Parameters

Parameter	Description
`Friend`	Friend object. The friend deleted from the buddy list.
`Group`	Group object. The group from which the friend was deleted.

JavaScript Syntax

```
Messenger.SetEventHandler("FriendDeletedFromGroup",
  OnFriendDeletedFromGroup)
function OnFriendDeletedFromGroup(fdfrnd, fdgroup)
```

JavaScript Example

```
function OnFriendDeletedFromGroup(fdfrnd, fdgroup)
{

  document.eventsForm.triggered.value += "\n" + "Contact List Event" +
    "\n" + "Friend Deleted From Group: " + fdfrnd.id + "\n";
  document.eventsForm.triggered.value += "FirstName LastName " +
    fdfrnd.FirstName + " " + fdfrnd.LastName + "\n";
  document.eventsForm.triggered.value += "\n" + "Contact List Event" +
    "\n"
  document.eventsForm.triggered.value += "Status: " +
    fdfrnd.Status.Status + " " + "Localized: " + fdfrnd.Status.Localized +
    " " + "DND: " + fdfrnd.Status.DND + " " + "\n";
  document.eventsForm.triggered.value += "Data: " + fdfrnd.Status.Data + "
    " + "Link Type: " + fdfrnd.Status.LinkType + " " + "Link: " +
    fdfrnd.Status.Link + " " + "Idle Time: " + fdfrnd.Status.IdleTime +
    "\n";
  document.eventsForm.triggered.value += "Image : " + fdfrnd.image + "\n";
  document.eventsForm.triggered.value += "PeerTOPeer : " +
    fdfrnd.PeerToPeer + "\n";
  document.eventsForm.triggered.value += "Group Name: " + fdgroup.name;
  document.eventsForm.triggered.value += "Session ID: " + "(" +
    fdgroup.UniqueSessionID + ")" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";

}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT FriendDeletedFromGroup(
                        [in] VARIANT vFriend,
                                   [in] VARIANT vGroup);
```

FriendChanged

Create an event handler for FriendChanged to be notified when a friend's information has changed in the user's buddy list.

Parameters

Parameter	Description
`Friend`	Friend object. The friend who's information was changed in the buddy list.
Change	**`String`**. What changed. Possible values: Image: the display image was changed. General: any other change

JavaScript Syntax

```
myMessenger.SetEventHandler("FriendChanged", OnFriendChanged)
function OnFriendChanged(Friend, Change)
```

JavaScript Example

When the **`FriendChanged`** event is triggered, the **`OnFriendChanged`** event handler displays various status information for the friend.

```
startevent.Messenger.SetEventHandler("FriendChanged", OnFriendChanged)

function OnFriendChanged(frndchange, theChange)
{
  document.eventsForm.triggered.value += "Contact List Event, Friend
    changed" + "\n";
  document.eventsForm.triggered.value += "\tFriend ID: " + frndchange.ID +
    "\n";
  document.eventsForm.triggered.value += "\tWhat Changed: " + theChange +
    "\n";
  document.eventsForm.triggered.value += "\tUnique Session ID: " +
    frndchange.UniqueSessionID  + "\n";
  document.eventsForm.triggered.value += "\tFirstName LastName " +
    frndchange.FirstName + " " + frndchange.LastName + "\n";
  document.eventsForm.triggered.value += "\tStatus: " +
    frndchange.Status.Status + " " + "Localized: " +
    frndchange.Status.Localized + " " + "DND: " + frndchange.Status.DND +
    " " + "\n";
  document.eventsForm.triggered.value += "\tData: " +
    frndchange.Status.Data + "\n";
  document.eventsForm.triggered.value += "\tLink Type: " +
    frndchange.Status.LinkType + "\n";
  document.eventsForm.triggered.value += "\tLink: " +
    frndchange.Status.Link + "\n";
  document.eventsForm.triggered.value += "\tIdle Time: " +
    frndchange.Status.IdleTime + "\n";
  document.eventsForm.triggered.value += "\tImage: " + frndchange.image +
    "\n";
  document.eventsForm.triggered.value += "\tPeerTOPeer: " +
    frndchange.PeerToPeer + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT FriendChanged(
                        [in] VARIANT vFriend,
                        [in] VARIANT sWhat);
```

GroupAdded

Set an event handler for **`GroupAdded`** to be notified when a new Group is added in the user's buddy list.

Parameters

Parameter	Description
`Group`	Group object. The Group added to the buddy list.

JavaScript Syntax

```
myMessenger.SetEventHandler("GroupAdded", OnGroupAdded)
function OnGroupAdded(Group)
```

JavaScript Example

The following example sets an event handler for GroupAdded to print out the information of group when it is added.

```
Messenger.SetEventHandler("GroupAdded", OnGroupAdded)

function OnGroupAdded(gagroup)
{
  document.eventsForm.triggered.value += "\n" + "Contact List Event" +
    "\n" + "Group Added: " + gagroup.name + "\n";
  document.eventsForm.triggered.value += "Session ID: " + "(" +
    gagroup.UniqueSessionID + ")" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000004)]
        HRESULT GroupAdded([in] VARIANT vFriend);
```

GroupDeleted

Set an event handler for **`GroupDeleted`** to be notified when a Group is deleted from the user's buddy list.

Parameters

Parameter	Description
`Group`	Group object. The Group removed from the buddy list.

JavaScript Syntax

```
myMessenger.SetEventHandler("GroupDeleted", OnGroupDeleted)
```

JavaScript Example

The following example creates an event handler to print the information of a group when it is deleted from the user's buddy list.

```
Messenger.SetEventHandler("GroupDeleted", OnGroupDeleted)
```

```
function OnGroupDeleted(gdgroup)
{
  document.eventsForm.triggered.value += "\n" + "Contact List Event" +
    "\n" + "Group Deleted: " + gdgroup.name + "\n";
  document.eventsForm.triggered.value += "Session ID: " + "(" +
    gdgroup.UniqueSessionID + ")" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000005)]
        HRESULT GroupDeleted([in] VARIANT vFriend);
```

GroupRenamed

Set an event handler for **`GroupRenamed`** to be notified when a Group is renamed in the user's buddy list.

Parameters

Parameter	Description
`grgroup`	Group object. The Group renamed in the buddy list.

JavaScript Syntax

```
myMessenger.SetEventHandler("GroupRenamed", OnGroupRenamed)
function OnGroupRenamed(grgroup)
```

JavaScript Example

The following example sets an event handler for the GroupRenamed event and prints information for the group that is renamed.

```
myMessenger.SetEventHandler("GroupRenamed", OnGroupRenamed)

function OnGroupRenamed(grgroup)
{
  document.eventsForm.triggered.value += "\n" + "Contact List Event" +
    "\n" + "Group Renamed To: " + grgroup.name +      "\n";
  document.eventsForm.triggered.value += "Session ID: " + "(" +
    grgroup.UniqueSessionID + ")" + " New Session ID: " + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
    }
```

C++ Syntax

```
[id(0x00000006)]
        HRESULT GroupRenamed([in] VARIANT vFriend);
```

PluginLoaded

Set and event handler for the **`PluginLoaded`** event to be notified when a plug-in is loaded by a friend.

Parameters

Parameter	Description
`Friend`	Friend object. The friend from the buddy list who has loaded a plug-in.
`Plugin`	Plugin object. The plugin object that was loaded by a friend in the buddy list.

JavaScript Syntax

```
myMessenger.SetEventHandler("PluginLoaded", OnPluginLoaded)
function OnPluginLoaded(Friend, plugin)
```

JavaScript Example

The following example sets an event handler for **`PluginLoaded`** and will print information for a plug-in when it is loaded by a friend in the buddy list.

```
myMessenger.SetEventHandler("PluginLoaded", OnPluginLoaded)

function OnPluginLoaded(Friend, plugin)
{
  document.eventsForm.triggered.value += "\n" + "Plug-in Loaded Event" +
    "\n" + "Plug-in Loaded By " + plfrnd.id + "\n";
  document.eventsForm.triggered.value += "Plug-in ID: " +
plplugin.PluginID     + "Plug-in Name: " + plplugin.PluginName + "\n";
  document.eventsForm.triggered.value += "Plug-in Version: " +
    plplugin.PluginVersion + "Plug-in Locale: " + plplugin.PluginLocale +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000007)]
        HRESULT PluginLoaded(
                        [in] VARIANT vFriend,
                        [in] VARIANT Plugin);
```

PluginUnloaded

The following example sets an event handler for **`PluginUnloaded`** and will print information for a plug-in when it is unloaded.

Parameters

Parameter	Description
`Friend`	Friend object. Friend from the buddy list who has unloaded a plug-in.
`Plugin`	Plugin object. The plug-in that was unloaded by the friend from the buddy list.

JavaScript Syntax

```
myMessenger.SetEventHandler("PluginLoaded", OnPluginLoaded)
function OnPluginUnLoaded(friend, plugin)
```

JavaScript Example

The following example creates an event handler to be notified when a friend in the buddy list unloads a plug-in and displays its information.

```
myMessenger.SetEventHandler("PluginLoaded", OnPluginLoaded)

function OnPluginUnLoaded(friend, plugin){
  document.eventsForm.triggered.value += "\n" + "Plug-in Unloaded Event" +
    "\n" + "Plug-in UnLoaded By " + pufrnd.id + "\n";
  document.eventsForm.triggered.value += "Plug-in ID: " +
puplugin.PluginID     + "Plug-in Name: " + puplugin.PluginName + "\n";
```

```
  document.eventsForm.triggered.value += "Plug-in Version: " +
    puplugin.PluginVersion + "Plug-in Locale: " + puplugin.PluginLocale +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000008)]
        HRESULT PluginUnloaded(
                        [in] VARIANT vFriend,
                        [in] VARIANT Plugin);
```

_IEventsFileTransferManager

The **`_IEventsFileTransferManager`** interface is implemented by the following objects:

- Messenger

Events

Name	Description
FileTransferProgress	Fired when a file is in the process of being sent.
FileTransferDone	Fired when the file transfer is complete.
FileTransferIncoming	Fired when a new file has been received from another user.
FileTransferAccepted	Fired when the outgoing file transfer is accepted by the other side.
FileTransferDeclined	Fired when the outgoing file transfer is declined by the other side.
FileTransferRemotePartyCancelled	Fired when accepted file transfer is cancelled by the other side.

FileTransferProgress

Fired when the file transfer is in progress.

Parameters

Parameter	Description
`transfer`	FileTransfer object.

JavaScript Syntax

```
yhost.SetEventHandler("FileTransferInProgress", On_FileTransferProgress)

function On_FileTransferProgress(transfer){
...
}
```

JavaScript Example

Creates an event handler that fires when a file transfer has been started.

```
yhost.SetEventHandler("FileTransferProgress", On_FileTransferInProgress)

function On_FileTransferInProgress(starttransfer)
{
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
    Event" + "\n";
  document.eventsForm.triggered.value += "File Transfer In Progress" +
    "\n";
  document.eventsForm.triggered.value += "File Transfer Name & ID: " +
    starttransfer.NameOriginal + starttransfer.id + "\n";
  document.eventsForm.triggered.value += "File Renamed: " +
    starttransfer.NameRenamed + "File Size: " + starttransfer.Size + "\n";
  document.eventsForm.triggered.value += "Error String, if any: " +
    starttransfer.Error + "\n";
  document.eventsForm.triggered.value += "File Transfer Status: " +
    starttransfer.Status + "File Transfer Direction: " +
    starttransfer.Direction + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000002)]
HRESULT FileTransferProgress(
  [in] VARIANT FileTransfer);
```

FileTransferDone

Sent to the receiving plug-in when the file transfer has completed, even if the transfer is cancelled.

Parameters

Parameter	Description
`transfer`	FileTransfer object.

JavaScript Syntax

```
yhost.SetEventHandler("FileTransferDone", On_FileTransferDone)

function On_FileTransferDone(transfer){
...
}
```

JavaScript Example

Creates an event handler for when a file transfer has completed.

```
yhost.SetEventHandler("FileTransferDone", On_FileTransferDone)

function On_FileTransferDone(transfercomplt)
{
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
    Event" + "\n";
  document.eventsForm.triggered.value += "File Transfer Completed for
    Filename: " + transfercomplt.NameOriginal + "\n";
  document.eventsForm.triggered.value += "File Transfer ID: " +
    transfercomplt.id + "\n";
  document.eventsForm.triggered.value += "File Renamed: " +
    transfercomplt.NameRenamed + "File Size: " + transfercomplt.Size +
    "\n";
  document.eventsForm.triggered.value += "Error String, if any: " +
    transfercomplt.Error + "\n";
  document.eventsForm.triggered.value += "File Transfer Status: " +
    transfercomplt.Status + "File Transfer Direction: " +
    transfercomplt.Direction + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000003)]
  HRESULT FileTransferDone([in] VARIANT FileTransfer);
```

FileTransferIncoming

Fired when a new incoming file transfer has been received.

Parameters

Parameter	Description
`transfer`	FileTransfer object.
`friend`	Friend object. The friend who sent the file.

JavaScript Syntax

```
yhost.SetEventHandler("FileTransferIncoming", On_FileTransferIncoming)

function On_FileTransferIncoming(transfer, friend){
...
}
```

JavaScript Example

Creates an event handler for when an incoming file transfer has been received and accepts the file.

```
yhost.SetEventHandler("FileTransferIncoming", On_FileTransferIncoming)

function on_FileTransferIncoming(ft, friend)
{
  if (cb_IEventsFileTransfer.checked)
  append_log_msg("FileTransferIncoming, ID: " + ft.ID + ", From Friend: "
    + friend.ID);

  ftm.FileTransferAccept(ft, "c:/temp/file123.jpg");
}
```

C++ Syntax

```
[id(0x00000001)]
HRESULT FileTransferIncoming(
  [in] VARIANT FileTransfer,
  [in] VARIANT varFriend);
```

FileTransferAccepted

Fired when outgoing file transfer is accepted by the other side.

Parameters

Parameter	Description
transfer	FileTransfer object.
friend	Friend object. The friend who sent the file.

JavaScript Syntax

```
yhost.SetEventHandler("FileTransferStarted", On_FileTransferAccepted)
function On_FileTransferAccepted(transfer, friend) {
...
}
```

JavaScript Example

Creates an event handler for when a file transfer has started.

```
yhost.SetEventHandler("FileTransferAccepted", On_FileTransferAccepted)

function On_FileTransferAccepted(transfer) {
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
    Event" + "\n";
  document.eventsForm.triggered.value += "File Transfer Started for
    Filename: " + transfer.NameOriginal + "File ID: " + transfer.id +
    "\n";
  document.eventsForm.triggered.value += "File Saved To Path: " +
    transfer.NameRenamed + "File Size: " + transfer.Size + "\n";
  document.eventsForm.triggered.value += "File Size Successfully
    Transferred: " + transfer.Transferred + "Error String, if any: " +
    transfer.Error + "\n";
  document.eventsForm.triggered.value += "File Transfer Status: " +
    transfer.Status + "File Transfer Direction: " + transfer.Direction +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000004)]HRESULT FileTransferAccepted(
  [in] VARIANT FileTransfer,
  [in] VARIANT varFriend);
```

FileTransferDeclined

Fired when outgoing file transfer is declined by the other side.

Parameters

Parameter	Description
`transfer`	FileTransfer object.
`friend`	Friend object. The friend who sent the file.

JavaScript Syntax

```
yhost.SetEventHandler("FileTransferDeclined", On_FileTransferDeclined)
function On_FileTransferDeclined(transfer, friend) {
...
}
```

JavaScript Example

Creates an event handler for when a file transfer has started.

```
yhost.SetEventHandler("FileTransferDeclined", On_FileTransferDeclined)

function On_FileTransferDeclined(transfer, friend) {
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
    Event" + "\n";
  document.eventsForm.triggered.value += "File Transfer Started for
    Filename: " + transfer.NameOriginal + "File ID: " + transfer.id +
    "\n";
  document.eventsForm.triggered.value += "File Saved To Path: " +
    transfer.NameRenamed + "File Size: " + transfer.Size + "\n";
  document.eventsForm.triggered.value += "File Size Successfully
    Transferred: " + transfer.Transferred + "Error String, if any: " +
    transfer.Error + "\n";
  document.eventsForm.triggered.value += "File Transfer Status: " +
    transfer.Status + "File Transfer Direction: " + transfer.Direction +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000005)]
  HRESULT FileTransferDeclined(
    [in] VARIANT FileTransfer,
    [in] VARIANT varFriend);
```

FileTransferRemotePartyCancelled

Sent to the sending plug-in when accepted file transfer is cancelled by the other side.

Parameters

Parameter	Description
`transfer`	FileTransfer object.
`friend`	Friend object. The friend who sent the file.

JavaScript Syntax

```
yhost.SetEventHandler("FileTransferRemotePartyCancelled",
On_FileTransferRemotePartyCancelled)
function On_FileTransferRemotePartyCancelled(transfer, friend) {
...
}
```

JavaScript Example

Creates an event handler for when a file transfer has started.

```
yhost.SetEventHandler("FileTransferRemotePartyCancelled",
  On_FileTransferRemotePartyCancelled)

function On_FileTransferRemotePartyCancelled(transfer, friend) {
  document.eventsForm.triggered.value += "\n" + "File Transfer Manager
```

```
    Event" + "\n";
  document.eventsForm.triggered.value += "File Transfer Started for
    Filename: " + transfer.NameOriginal + "File ID: " + transfer.id +
    "\n";
  document.eventsForm.triggered.value += "File Saved To Path: " +
    transfer.NameRenamed + "File Size: " + transfer.Size + "\n";
  document.eventsForm.triggered.value += "File Size Successfully
    Transferred: " + transfer.Transferred + "Error String, if any: " +
    transfer.Error + "\n";
  document.eventsForm.triggered.value += "File Transfer Status: " +
    transfer.Status + "File Transfer Direction: " + transfer.Direction +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000006)]
  HRESULT FileTransferRemotePartyCancelled(
    [in] VARIANT FileTransfer,
    [in] VARIANT varFriend);
```

_IEventsContactListUI

The **`_IEventsContactListUI`** interface is implemented by the following objects.

- MainWindow

Events

Name	Description
FriendSelected	Fired when friend is selected in the user’s buddy list.
FriendUnselected	Fired when friend loses selection in the user’s buddy list.

FriendSelected

Fired when a friend is selected in the buddy list.

Parameters

Parameter	Description
Friend	Friend Object. The friend selected in the buddy list.

JavaScript Syntax

```
MyWindow.SetEventHandler("FriendSelected", OnFriendSelected)

function onFriendSelected(Friend) {
...
}
```

JavaScript Example

The following example creates an event handler for the **`FriendSelected`** event.

```
var mainWindow = yhost.Messenger.MainWindow;
mainWindow.SetEventHandler("FriendSelected", onFriendSelected);

function onFriendSelected(friend)
{
   LogClear();
   Log(friend.ID + " " + friend.FirstName + " " + friend.LastName);
}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT FriendSelected([in] VARIANT Friend);
```

FriendUnselected

Fired when a friend loses focus in the buddy list.

Parameters

Parameter	Description
Friend	Friend Object. The friend that becomes deselected in the buddy list.

JavaScript Syntax

```
MyWindow.SetEventHandler("FriendUnselected", OnFriendUnselected)

function onFriendUnselected(Friend){
...
}
```

JavaScript Example

The following example creates an event handler for the **`FriendUnselected`** event.

```
var mainWindow = yhost.Messenger.MainWindow;
mainWindow.SetEventHandler("FriendUnselected", onFriendUnselected);

function onFriendUnselected(friend)
{
   LogClear();
   Log(friend.ID + " " + friend.FirstName + " " + friend.LastName);
}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT FriendUnselected([in] VARIANT Friend);
```

_IEventsConversationPluginServices

The **`_IEventsConversationPluginServices`** interface is implemented by the following objects:

- ConversationPluginHost

Events

Name	**Description**
PluginMessage	Fired when a new plug-in message arrives from the other plug-in instance.
RemoteReady	Fired when the remote plug-in is loaded an ready.
RemoteUnloaded	Fired when the remote plug-in is unloaded and no longer available.

PluginMessage

Fired when a new plug-in message arrives from the other plug-in instance.

Parameters

Parameter	**Type/Description**
`message`	String. The text of the received message.

JavaScript Syntax

```
yhost.SetEventHandler("PluginMessage", OnPluginMessage)
```

JavaScript Example

Redraws a map based on input sent by the other plug-in instance.

```
yhost.SetEventHandler( 'PluginMessage', onPluginMessage );
function onPluginMessage( msg )
{
  trace( "Received: " + msg );
  alert("Received");
  var regexp = /^CHANGELOC (-?\d*\.\d+),(-?\d*\.\d+),(\d+)$/;
  var matches = regexp.exec( msg );
  if ( matches != null )
  {
    var lat  = Number( matches[1] );
    var lon  = Number( matches[2] );
    var zoom = Number( matches[3] );
    map.drawZoomAndCenter( new YGeoPoint( lat, lon ), zoom );
  }
}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT PluginMessage([in] VARIANT Message);
```

RemoteReady

Fired when the remote plug-in is loaded and ready for input.

JavaScript Syntax

```
yhost.SetEventHandler("RemoteReady", OnRemoteReady)
```

JavaScript Example

Sets a global variable when the remote plug-in is loaded and ready.

```
yhost.SetEventHandler( 'RemoteReady', onRemoteReady );
function onRemoteReady()
{
  remoteActive = true;
  alert("remote ready");
}
```

C++ Syntax

```
[id(0x00000004)]
        HRESULT RemoteReady();
```

RemoteUnloaded

Fired when the remote plug-in is unloaded and no longer available.

JavaScript Syntax

```
yhost.SetEventHandler("RemoteUnloaded", OnRemoteUnloaded)
```

JavaScript Example

Sets a global variable when the remote plug-in is unloaded and displays an alert message.

```
yhost.SetEventHandler( 'RemoteUnloaded', onRemoteUnloaded );
function onRemoteUnloaded()
{
  remoteActive = false;
  alert("Your Buddy closed the IM Map!");
}
```

C++ Syntax

```
[id(0x00000005)]
        HRESULT RemoteUnloaded();
```

_IEventsConversationWindow

The **`_IEventsConversationWindow`** interface is implemented by the following objects:

- ConversationPluginHost

Events

Name	**Description**
UserIsTyping	Fired when the user types within the input window.
IMReceived	Fired when a new instant message is received.
IMSent	Fired when an instant message is sent.
IncomingIM	Fired when an incoming instant message has been received but not yet processed. The plug-in can modify the message using the passed parameter.

UserIsTyping

Fired when user types inside the input window.

JavaScript Syntax

```
yhost.SetEventHandler("UserIsTyping", On_UserIsTyping)
function On_UserIsTyping()
```

JavaScript Example

Creates an event handler to be notified when the user is typing in the input window of the Conversation window.

```
yhost.SetEventHandler("UserIsTyping", On_UserIsTyping)
function OnUserIsTyping()
{
  var GetInputText;
  document.eventsForm.triggered.value += "Conversation Window Event: User
    is Typing" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
  // get what the user was typing
  GetInputText=yhost.InputWindowText;
}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT UserIsTyping();
```

IMReceived

Fired when the new instant message is received.

Parameters

Parameter	Type/Description
`Msg`	**`String`**. The text of the received message. Can be formatted with HTML tags. See IM Tags in the reference section for a complete list of allowed tags.

JavaScript Syntax

```
yhost.SetEventHandler("IMReceived", On_IMReceived)
function On_IMReceived(Msg)
```

JavaScript Example

Creates an event handler to catch when an instant message is received by the Conversation plug-in.

```
startevent.Messenger.Windows.SetEventHandler("IMReceived",
  On_IMReceived)

function On_IMReceived(msgrecv)
{
  document.eventsForm.triggered.value += "Conversation Window Event: IM
    Received:" + msgrecv + "\n";
  msgrecv = "Changing the IM text for IM Received Event";
  document.eventsForm.triggered.value += "Conversation Window Text
    Changed: " + msgrecv + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT IMReceived([in] VARIANT msg);
```

IMSent

Fired when an instant message is sent to another user. This is not fired when the SendIM method is used to send an instant message; it is only fired when the user sends the message.

Parameters

Parameter	Type/Description
`Msg`	**`String`**. The text of the sent message. Text can be HTML formatted. See IM Tags in the reference section for a complete list of allowed tags.

JavaScript Syntax

```
yhost.SetEventHandler("IMSent", On_IMSent)
function On_IMSent(Msg)
```

JavaScript Example

```
startevent.Messenger.Windows.SetEventHandler("IMSent", On_IMSent)
function On_IMSent(msgsent)
{
  document.eventsForm.triggered.value += "Conversation Event: IM Sent" +
    msgsent.Text + "\n";
  msgsent.Text = "Changing the IM text for IM Sent Event";
  document.eventsForm.triggered.value += "Conversation Event: IM Sent" +
    msgsent.Text + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT IMSent([in] VARIANT msg);
```

IncomingIM

Fired when an incoming instant message has been received but not yet processed. The plug-in can modify the message using the passed parameter.

Parameters

Parameter	Type/Description
`IM`	**`IMMessage`** object.

JavaScript Syntax

```
startevent.Messenger.Windows.SetEventHandler("IncomingIM", On_IncomingIM)
function On_IncomingIM(IM)
```

JavaScript Example

Displays the text in the instant message.

```
startevent.Messenger.MainWindow.SetEventHandler("IncomingIM",
On_IncomingIM);
function On_IncomingIM(IM)
{
  document.eventsForm.triggered.value += "Conversation Window Event:
    Incoming IM: " + IM.Text + "\n";
}
```

C++ Syntax

```
[id(0x00000004)]
        HRESULT IncomingIM([in] VARIANT IMMessage);
```

_IEventsContentTabPluginServices

The **`_IEventsContentTabPluginServices`** interface is implemented by the following objects.

- ContentTabPluginCommon
- ContentTabMainWindowPluginHost

Events

Name	Description
PluginMessage	Fired when message is received by the plug-in.
PluginStatus	Fired when the user clicks the status link in the buddy list.
MIMEMessage	Fired when the user clicks on a ymsgr:getplugin link with a message specified.

PluginMessage

Fired when a message is received by the plug-in.

Parameters

Parameter	Description
`Message`	**`String`**. The text content of the message.
`FromFriend`	Friend object. The friend who sent the message.

JavaScript Syntax

```
yhost.SetEventHandler ('PluginMessage', on_PluginMessage);
function on_PluginMessage(Message, FromFriend)
```

JavaScript Example

Set and event handler to fire when the message is received by the user from the friends plug-in.

```
yhost.SetEventHandler ('PluginMessage', on_PluginMessage);
```

```
function on_PluginMessage(mes, friend)
{
  if (cb_IEventsPluginHost.checked)
    append_log_msg ("PluginMessage: '" + mes + "', From Friend: " +
      friend.ID);
}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT PluginMessage(
                        [in] VARIANT mes,
                        [in] VARIANT varFriend);
```

PluginStatus

Fired when the user clicks the status link in the buddy list.

Parameters

Parameter	Description
`Friend`	Friend object. Friend in the buddy list whose status was clicked.
`Text`	**`String`**. Status string.
`Data`	**`String`**. Status related data that was passed in.

JavaScript Syntax

```
yhost.SetEventHandler ('PluginStatus', on_PluginStatus);
function on_PluginStatus(Friend, Text, Data);
```

C++ Syntax

```
HRESULT PluginStatus(
                        [in] VARIANT varFriend,
                        [in] VARIANT Text,
                        [in] VARIANT Data);
```

MIMEMessage

Fired when user clicks on a **`ymsgr`** get plug-in link that has a message specified. The plug-in receives the message whether is was already running or not.

Parameters

Parameter	Description
`Text`	**`String`**. Message content.

JavaScript Syntax

```
yhost.SetEventHandler ('MIMEMessage', on_MIMEMEssage);
function on_MIMEMessage(Text);
```

JavaScript Example

Creates an event handler for receiving a MIME message.

```
var Globals = { yahoo: window.external };
window.onload = function()
{
  Globals.yahoo.SetEventHandler("MIMEMessage", onMimeMessage);
  Globals.yahoo.LocalReady();
}

function onMimeMessage(str)
{
  alert('received MimeMessage "' + str + '"');
}
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT MIMEMessage([in] VARIANT mes);
```

_IEventsMainPluginWindow

The _IEventsMainPluginWindow interface is implemented by the following objects.

- ContentTabMainWindowPluginHost

Events

Name	Description
SecondaryWindowMessage	Fired when secondary window sends a message.
SecondaryWindowReady	Fired when secondary window is opened and ready.
SecondaryWindowClosed	Fired when secondary window is closed.
MainWindowStateChanged	Fired when main plug-in window state has changed.

SecondaryWindowMessage

Fired when the secondary window sends a message

JavaScript Syntax

```
yhost.SetEventHandler ('SecondaryWindowMessage',
on_SecondaryWindowMessage);

function on_SecondaryWindowMessage(msg)
```

JavaScript Example

Creates an event handler for when the secondary window sends a message.

```
yhost.SetEventHandler ('SecondaryWindowMessage',
on_SecondaryWindowMessage);

function on_SecondaryWindowMessage(msg)
{
  if (cb_IEventsMainPluginWindow.checked)
  append_log_msg ("SecondaryWindowMessage: '" + msg + "'");
}
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT SecondaryWindowMessage([in] VARIANT mes);
```

SecondaryWindowReady

Fired when the secondary window is opened and ready.

JavaScript Syntax

```
yhost.SetEventHandler ('SecondaryWindowReady', on_SecondaryWindowReady);
function on_SecondaryWindowReady()
```

JavaScript Example

Creates an event handler for when the secondary window is opened.

```
function on_SecondaryWindowReady()
{
  if (cb_IEventsMainPluginWindow.checked)
  append_log_msg ("SecondaryWindowReady");
}
```

C++ Syntax

```
HRESULT SecondaryWindowReady();
        [id(0x00000005)]
```

SecondaryWindowClosed

Fired when the secondary window closes.

JavaScript Syntax

```
yhost.SetEventHandler ('SecondaryWindowClosed',
on_SecondaryWindowClosed);
function on_SecondaryWindowClosed()
```

JavaScript Example

Creates an event handler for when the secondary window closes.

```
yhost.SetEventHandler ('SecondaryWindowClosed',
on_SecondaryWindowClosed);

function on_SecondaryWindowClosed()
{
  if (cb_IEventsMainPluginWindow.checked)
  append_log_msg ("SecondaryWindowClosed");
}
```

C++ Syntax

```
[id(0x00000005)]
        HRESULT SecondaryWindowClosed();
```

MainWindowStateChanged

Fired when the main plug-in window's state changes from one state to another. Possible states are docked or embedded. The docked state refers to a separate window opened from docked to the main window. The embedded state means the plug-in is embedded in the content tab window. To get the current state of the main plug-in window, call **`yhost.MainWindowState`**.

JavaScript Syntax

```
yhost.SetEventHandler ('MainWindowPoppedOut', on_MainWindowPoppedOut);
function on_MainWindowPoppedOut()
```

JavaScript Example

Creates an event handler for when the Tab plug-in's main window pops out.

```
yhost.SetEventHandler ('MainWindowPoppedOut', on_MainWindowPoppedOut);
function on_MainWindowPoppedOut()
{
  if (cb_IEventsMainPluginWindow.checked)
    append_log_msg ("MainWindowPoppedOut");
}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT MainWindowStateChanged();
```

_IEventsMessenger2

The **`_IEventsMessenger2`** interface is implemented by the following objects.

- Messenger

Events

Name	Description
LoggedIn	Fired when Messenger logs in to the network.
LoggedOut	Fired when Messenger logs out of the network.
Alert	Fired when an alert is received.

LoggedIn

Set an event handler for LoggedIn to be notified when messenger logs in to the network.

JavaScript Syntax

```
MyMessenger.SetEventHandler("LoggedIn", OnLoggedIn);

function OnLoggedIn() {
...
}
```

JavaScript Example

The following example sets an event handler for the **`LoggedIn`** event.

```
MyMessenger.SetEventHandler("LoggedIn", OnLoggedIn);

function OnLoggedIn() {
  document.eventsForm.triggered.value += "\n" + "Messenger Event" + "\n" +
    "Messenger logged into the network" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000001)]
```

```
        HRESULT LoggedIn();
```

LoggedOut

Set an event handler for **`LoggedOut`** to be notified when messenger logs out of the network. This event is only triggered in external applications, not inside of a plug-in.

JavaScript Syntax

```
MyMessenger.SetEventHandler("LoggedOut", OnLoggedOut)

function OnLoggedOut() {
...
}
```

JavaScript Example

The following example sets an event handler for the **`LoggedOut`** event.

```
MyMessenger.SetEventHandler("LoggedOut", OnLoggedOut)

function OnLoggedOut() {
   document.eventsForm.triggered.value += "\n" + "Messenger Event " +
     "Messenger Logged Out of the Network" + "\n";
   document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT LoggedOut();
```

Alert

Create an Alert event handler to be notified when Messenger fires an alert.

Parameters

Parameter	Description
`AlertID`	**`Number`**. Type of alert returned. Possible values: 1: Mail (Reserved) 2: Calendar (Reserved) 3: Personals (Reserved) 4: Reserved 5: Reserved 6: Reserved 7: Mingle 8: HTML alert (Alerts team) 9: VoiceMail Alerts
`AlertData`	**`String`**. Format: **`<alert id="alert id" editurl="url_to_go_to" w="min width" h="min height" newwindow="open_new_window">`** **`<![CDATA[unescaped_html_content]]>`** **`</alert>`** where, **`id`**: predefined alert id ('n'-news, 's'-shopping, 'a'-auctions, 'h'-horoscope etc.) For unrecognized IDs, Messenger will show text "Yahoo! Alerts" **`editurl`**: Contains url to go to when the edit button is clicked. If empty, we can go to a default url. **`w`**: Minimum width needed to show the alert html. **`h`**: Minimum height needed to show the alert html. **`unescaped_html_content`**: unescaped HTML content to be shown in the alert. **`newwindow`**: opens new window if set to '1', else shows the alert in existing window of that type.

JavaScript Syntax

```
MyMessenger.SetEventHandler("Alert", OnAlert)
function OnAlert(AlertID, AlertData)
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT Alert(
                        [in] VARIANT vID,
                        [in] VARIANT vData);
```

_IEventsPluginHost

The _IEventsPluginHost interface is implemented by the following objects.

- PluginHost

Events

Name	Description
Unloaded	Fired when the host is about to unload the plug-in.
HTTPRequestCompleted	Fired when the URL content has completed downloading.
HTTPRequestError	Fired when the URL download fails.

Unloaded

Fired that the host is about to unload the plug-in. The plug-in should perform all related clean up when handling this event. This is the last event the plug-in will receive.

JavaScript Syntax

```
myHost.SetEventHandler (‘Unloaded’, On_Unloaded);

function on_Unloaded() {
...
}
```

JavaScript Example

Creates an event handler for the `Unloaded` event.

```
yhost.SetEventHandler ('Unloaded', on_Unloaded);

function on_Unloaded ()
{
// the last event a plug-in receives
if (cb_IEventsPluginHost.checked)
append_log_msg ("Unloaded");
}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT Unloaded();
```

HTTPRequestCompleted

Fired when the URL content has completed downloading after a SendHTTPRequest or SendHTTPRequestEx has been made.

Parameters

Parameter	Description
id	**`String`**. Plug-in related request ID. The same ID that was passed in **`SendHTTPRequest and SendHTTPRequestEx`**.
data	**`String`**. The data that has been sent.

JavaScript Syntax

```
myHost.SetEventHandler (‘HTTPRequestCompleted’, On_HTTPRequestCompleted);

function on_HTTPRequestCompleted (id, data){
...
}
```

JavaScript Example

Get a weather RSS feed for Sunnyvale California from xml.weather.yahoo.com.

```
yhost = window.external;

function On_HTTPRequestCompleted(id, textstring) {
  // do stuff with the textstring }

function On_HTTPRequestError(id, errortype) {
  alert("Failed to connect!"); }

yhost.SetEventHandler("HTTPRequestCompleted", On_HTTPRequestCompleted);
yhost.SetEventHandler("HTTPRequestError", On_HTTPRequestError);

yhost.LocalReady();
```

```
// gets RSS feed for Sunnyvale, CA
var xmlhttp = yhost.SendHTTPRequest(1,
"http://xml.weather.yahoo.com/forecastrss?p=94089");
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT HTTPRequestCompleted(
                        [in] VARIANT sRequestID,
                        [in] VARIANT sData,
                        [in] VARIANT sHeaders);
```

HTTPRequestError

Fired when the URL download fails after a SendHTTPRequest or SendHTTPRequestEx has been made.

Parameters

Parameter	Description
id	**`String`**. Plug-in related request ID. The ID passed in **`SendHTMLGetRequest`**.
ErrorType	**`String`**. The type of error returned.
ResponseHTTPHeaders	**`String`**. The complete headers section of HTTP response, including HTTP status code and terminating empty string. Individual headers are separated by CR/LF characters. Supplied only if requested in SendHTTPRequestEx.
ResponseData	**`String`**.

JavaScript Syntax

```
myHost.SetEventHandler (‘HTTPRequestError’, On_HTTPRequestError);

function on_HTTPRequestError (id, ErrorType){
...
}
```

JavaScript Example

Get a weather RSS feed for Sunnyvale California from xml.weather.yahoo.com.

```
yhost = window.external;

function On_HTTPRequestCompleted(id, textstring) {
  // do stuff with the textstring }

function On_HTTPRequestError(id, errortype) {
  alert("Failed to connect!"); }

yhost.SetEventHandler("HTTPRequestCompleted", On_HTTPRequestCompleted);
yhost.SetEventHandler("HTTPRequestError", On_HTTPRequestError);

yhost.LocalReady();
// gets RSS feed for Sunnyvale, CA
var xmlhttp = yhost.SendHTTPRequest(1,
"http://xml.weather.yahoo.com/forecastrss?p=94089");
```

C++ Syntax

```
[id(0x00000004)]
        HRESULT HTTPRequestError(
                        [in] VARIANT sRequestID,
                        [in] VARIANT sError,
                        [in] VARIANT sHeaders);
```

_IEventsSecondaryWindow

The _IEventsSecondaryWindow interface is implemented by the following objects.

- PluginHost

Events

Name	Description
MainPluginWindowMessage	Fired when Tab plug-in main window sends a message to the secondary window.

MainPluginWindowMessage

Fired when the Tab plug-in main window sends a message to the secondary window.

Parameters

Parameter	Description
`msg`	**`String`**. Message sent to secondary window.

JavaScript Syntax

```
yhost.MainPluginWindowMessage(msg);
```

JavaScript Example

Creates an event handler for when the main window of the Tab plug-in main window sends a message to its secondary window.

```
yhost.SetEventHandler ('MainPluginWindowMessage',
on_MainPluginWindowMessage);
function on_MainPluginWindowMessage(msg)
{
  if (cb_IEventsSecondaryWindow.checked)
    append_log_msg ("MainPluginWindowMessage: '" + msg + "'");
}
```

_IEventsSecondaryWindow
MainPluginWindowMessage

C++ Syntax

```
[id(0x00000001)]
        HRESULT MainPluginWindowMessage([in] VARIANT mes);
```

_IEventsToast

The _IEventsToast interface is implemented by the following objects.

- PluginHost

Events

Name	**Description**
ToastLinkClicked	Fired when the user clicks a link in the toast window.

ToastLinkClicked

Fired when user clicks the link with the toast popup window.

JavaScript Syntax

```
myHost.SetEventHandler(‘ToastLinkClicked’, On_ToastLinkClicked);
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT ToastLinkClicked(
                        [in] VARIANT ToastID,
                        [in] VARIANT LinkHRef);
```

_IEventsWindow

The **`_IEventsWindow`** interface provides the ability to get event notifications when the user interacts with the content tab window, the conversation window, or a text message window in some way. The **`_IEventsWindow`** interface is implemented by the following objects.

- MainWindow
- ConversationWindow

Events

Name	Description
Moved	Fired when window is moved.
Resized	Fired when window is resized.
Maximized	Fired when window is maximized.
Minimized	Fired when window is minimized.
Activated	Fired when the window loses or gains input focus.
Closed	Fired when window is closed.

Moved

The **`Moved`** event is fired when the window is moved to a new location.

Parameters

Parameter	Description
X	X. Number. Top left corner horizontal coordinate of the window.
Y	Y. Number. Top left corner vertical coordinate of the window.

JavaScript Syntax

```
myMessenger.MainWindow.SetEventHandler("Moved", OnWindowMoved)

function OnWindowMoved(xnum, ynum) {
```

```
...
}
```

JavaScript Example

The following example sets an event handler for the **`Moved`** event.

```
myMessenger.MainWindow.SetEventHandler("Moved", OnWindowMoved)

function OnWindowMoved(xnum, ynum){
  document.eventsForm.windowmain.value += "\n" + "Main Window Move Event
    Occurred" + "\n";
  document.eventsForm.windowmain.value += "Top Left Horizontal Position: "
    + xnum + "\n";
  document.eventsForm.windowmain.value += "Top Left Vertical Position: " +
    ynum + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT Moved(
                        [in] VARIANT x,
                        [in] VARIANT y);
```

Resized

The **`Resized`** event is fired when the window is resized.

Parameters

Parameter	Description
Width	Numeric. The new width of the window.
Height	Numeric. The New height of the window.

JavaScript Syntax

```
myMessenger.MainWindow.SetEventHandler("Resized", OnWindowResized)

function OnWindowResized(width, height) {
```

```
...
}
```

JavaScript Example

The following example sets an event handler for the **`Resized`** event.

```
myMessenger.MainWindow.SetEventHandler("Resized", OnWindowResized)

function OnWindowResized(rewidth, reheight){
  document.eventsForm.windowmain.value += "\n" + "Main Window Move Event
    Occurred, Window Resized" + "\n";
  document.eventsForm.windowmain.value += "Window Width: " + rewidth +
    "\n";
  document.eventsForm.windowmain.value += "Window Height: " + reheight +
    "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000002)]
        HRESULT Resized(
                        [in] VARIANT width,
                        [in] VARIANT height);
```

Minimized

The **`Minimized`** event is fired when the window is minimized.

JavaScript Syntax

```
myMessenger.MainWindow.SetEventHandler("Minimized", OnWindowMinimized)

function OnWindowMinimized() {
...
}
```

JavaScript Example

The following example creates an event handler for the **`Minimized`** event.

```
Messenger.MainWindow.SetEventHandler("Minimized", OnWindowMinimized)
```

```
function OnWindowMinimized() {
  document.eventsForm.windowmain.value += "\n" + "Main Window Move Event
    Occurred, Window Minimized" + "\n";
  document.eventsForm.windowmain.value += "Main Window Minimized" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000003)]
        HRESULT Minimized();
```

Maximized

The **`Maximized`** event is fired when the window is maximized.

JavaScript Syntax

```
myMessenger.MainWindow.SetEventHandler("Maximized", OnWindowMaximized)

function OnWindowMaximized() {
...
}
```

JavaScript Example

The following example creates an event handler for the **`Maximized`** event.

```
myMessenger.MainWindow.SetEventHandler("Maximized", OnWindowMaximized)

function OnWindowMaximized(){
  document.eventsForm.windowmain.value += "\n" + "Main Window Move Event
    Occurred, Window Maximized" + "\n";
  document.eventsForm.windowmain.value += "Main Window Maximized" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000005)]
        HRESULT Maximized();
```

Activated

The **`Activated`** event is fired when the window is activated.

JavaScript Syntax

```
myMessenger.MainWindow.SetEventHandler("Activated", OnWindowActivated)

function OnWindowActivated() {
...
}
```

JavaScript Example

The following example creates an event handler for the **`Activated`** event.

```
myMessenger.MainWindow.SetEventHandler("Activated",   OnWindowActivated)

function OnWindowActivated(inFocus){
  document.eventsForm.windowmain.value += "\n" + "Main Window Event
    Occurred, WIndow Activated" + "\n";
  document.eventsForm.windowmain.value += "Main Window Activated? " + "If
    true focus = yes " + inFocus + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000006)]
        HRESULT Activated([in] VARIANT focus);
```

Closed

The **`Closed`** event is fired when the window is closed.

JavaScript Syntax

```
myMessenger.MainWindow.SetEventHandler("Closed", OnWindowClosed)

function OnWindowClosed() {
```

```
...
}
```

JavaScript Example

The following example creates an event handler for the **`Closed`** event.

```
Messenger.MainWindow.SetEventHandler("Closed", OnWindowClosed)

function OnWindowClosed(){
  document.eventsForm.windowmain.value += "\n" + "Main Window Event
  Occurred, Window Closed" + "\n";
  document.eventsForm.triggered.value + "\n" + "\n";
}
```

C++ Syntax

```
[id(0x00000007)]
        HRESULT Closed();
```

_IEventsWindowCreation

The **`_IEventsWindowCreation`** interface is implemented by the following objects:

- Messenger

Events

Name	Description
WindowCreated	Fired when a new window is created

WindowCreated

Create an event handler for WindowCreated to be notified when new window is created.

Parameters

Parameter	Description
`Type`	**`String`**. Window type. Possible values are: “Conversation” “PSTN”, “Conference”.
`win`	If type is “Conversation”: Conversation window object. For other types: **TBD**.

JavaScript Syntax

```
myMessenger.SetEventHandler("WindowCreated", OnWindowCreated)
function OnWindowCreated(wintype, winobj)
```

JavaScript Example

```
Messenger.SetEventHandler("WindowCreated", OnWindowCreated)

function OnWindowCreated(wintype, winobj)
{
  document.eventsForm.triggered.value += "\n" + "Window Creation Event" +
    "\n" + "Window Type: " + wintype.WindowCreated + "\n";
  if (wintype = "Conversation")
```

```
{
  document.eventsForm.windowmain.value += "Window Objects: " + "\n";
  document.eventsForm.windowmain.value += "Opened or Closed: " +
    winobj.WinOpen + "Window Visible? " + winobj.WinVisible + "\n";
  document.eventsForm.windowmain.value += "Top Left Horizontal
    Position: " + winobj.WinX + "Top Left Vertical Position: " +
    winobj.WinY + "\n";
  document.eventsForm.windowmain.value += "Window Width: " +
    winobj.WinWidth + "Window Height: " + winobj.WinHeight + "\n";
  document.eventsForm.windowmain.value += "Window Handler " +
    winobj.WinHWND + "Window Title: " + winobj.WinTitle + "\n";
  document.eventsForm.windowmain.value += "Window In Foreground?" +
    winobj.WinForeground + "Window Active? " + winobj.WinActive + "\n";
  document.eventsForm.windowmain.value += "Window Minimized?" +
    winobj.WinMinimized + "Window Maximized? " + winobj.WinMaximized +
    "\n"
  document.eventsForm.triggered.value + "\n" + "\n";}
```

C++ Syntax

```
[id(0x00000001)]
        HRESULT WindowCreated(
                        [in] VARIANT type,
                        [in] VARIANT win);
```

Appendix

- IM Tags

IM Tags

The following tags can be embedded in the IM messages to modify text attributes such as font, font size, color, style, etc. In the table below, `\033` stands for single octal character `33`.

Tag Name	**Syntax**	**Description**
Bold On	`\033[1m OR <b>`	Characters after this tag are shown bold
Bold Off	`\033[x1m OR <b>`	Bolding ends.
Italics On	`\033[2m OR <i>`	Characters after this tag are shown italic.
Italics Off	**`\033[x2m OR </i>`**	Italic ends.
Underline On	**`\033[3m OR <u>`**	Characters after this tag are shown underlined.
Underline Off	**`\033[x3m OR <\u>`**	Underlining ends.

Color tag 1	**`\033[3?m where ? can be:`**
	`0-black,`
	`1-blue,`
	`2-cyan,`
	`3-gray,`
	`4-green,`
	`5-pink,`
	`6-magenta,`
	`7-orange,`
	`8-red,`
	`9-yellow`
		Characters after this tag are colored.

Tag Name	Syntax	Description
Color tag 2	**`\033[3#RRGGBBm where RRGGBB are hex values for red, green, blue components of the color.`**	Characters after this tag are colored.
Color off	**`\033[3xm`**	Colored text ends.
Font face	**`<font face="Arial">`**	Characters after this tag will be in arial font.
Font size	**`<font size="32">`**	Characters after this tag will be font size 32.
Font face and size	**`<font face="Tahoma" size="32">`**	Characters after this tag will be Tahoma font size 32.
Font Off	**`</font>`**	Characters after this tag will be in the default font.
Faded colored text	**`<fade #rrggbb, #rrggbb> xxxxxx </fade>`**	The text 'xxxxxx' will appear faded from color '1' to color 'n' whose values are represented in rrggbb values in the **`fade`** tag. There can be two or more colors in this tag.
Alternate colored text	**`<alt #rrggbb, #rrggbb> xxxxxx </alt>`**	Each consecutive character in text 'xxxxxx' will appear in different colors (color '1' to color 'n' whose values are represented in rrggbb values in the **`alt`** tag). There can be two or more colors in this tag.