

Live Connect Preliminary Documentation

[This documentation is preliminary and is subject to change.]

This document supports a preliminary release of a software product that may be changed substantially prior to final commercial release. This document is provided for informational purposes only and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2011 Microsoft. All rights reserved.

Microsoft, Hotmail, SkyDrive, Windows, and Windows Live are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Live Connect

[This documentation is preliminary and is subject to change.]

Live Connect provides a set of APIs that enable your apps to work with info in Hotmail, Windows Live Messenger, SkyDrive, and other services that are compatible with Live Connect. The info in these sections shows you how.

- [Getting started](#)

What's new in this release and how to get the settings your apps need to begin calling the Live Connect APIs.

- [Core concepts](#)

How to call the APIs, sign in a user, obtain a user's okay for you to create and work with their info, and get a user's basic info.

- [Identity \(profiles\)](#)

How to call the APIs to greet a user by name, streamline registering a user from your app, contact a user, and determine a user's age.

- [Hotmail \(contacts and calendars\)](#)

How to call the APIs to work with contacts and calendars and how to find contacts who a user knows on a website.

- [SkyDrive \(files and photos\)](#)

How to call the APIs to work with folders, files, albums, photos, videos, tags, and comments.

- [Messenger \(instant messaging\)](#)

How to call the APIs to interact with Messenger users and their buddies by using Extensible Messaging and Presence Protocol (XMPP), an open standard for real-time communication.

© 2011 Microsoft Corporation. All rights reserved.

Getting started

[This documentation is preliminary and is subject to change.]

Live Connect provides a set of APIs that enable your apps to work with info in Hotmail, Windows Live Messenger, SkyDrive, and other services that are compatible with Live Connect. Here's what you can do with these APIs, how to get the settings your apps need to call the APIs, and how to begin writing code to call the APIs.

- [What your apps can do with the APIs](#)
- [Learn more](#)

What your apps can do with the APIs

Identity

- **Profiles.** Get a user's *profile* info. A profile includes info that a user provides about himself or herself, such as a first name, last name, gender, and a birth date.
- **Personalization.** Enable a user to sign in to a website, personalize the signed-in user's experience on that website, and more easily register a user on that website if needed.

To learn more, see [Identity \(profiles\)](#).

Hotmail

- **Contacts.** Read and create contacts. You can also enable a website to determine which of a user's contacts is also a user of that website.
- **Calendars.** Work with calendars and events.

To learn more, see [Hotmail \(contacts and calendars\)](#).

SkyDrive

- **Folders, files, and more.** Work with folders, files, albums, photos, videos, tags, and comments.

To learn more, see [SkyDrive \(files and photos\)](#).

Messenger

- **Instant messaging.** Exchange instant messages between Messenger users.
- **Users and buddies.** Get a user's info and his or her buddies' info—including things like their friendly names, presence info, and status messages.
- **Status updates.** Update a user's *status message*, a personal message that

the user chooses to share with others (for example, participating in a web-based game.)

To learn more, see [Messenger \(instant messaging\)](#).

[Top](#)

Learn more

To get going, start here.

- [What's new](#)

What's new in the latest releases.

- [Getting a client ID and configuring your app](#)

How to get the settings your apps need to begin calling the APIs.

- [Metro style apps](#)

How Metro style apps can call the APIs.

- [Windows Phone apps](#)

How Windows Phone apps can call the APIs.

- [Web apps](#)

How client-side and server-side websites and scripts can call the APIs.

- [Working with the code examples](#)

How to work with many of the smaller code examples throughout this documentation in the context of larger reference code samples.

- [Mobile and Windows desktop apps](#)

How mobile apps and Windows desktop apps can call the APIs.

- [Community resources](#)

Links to helpful tools, code samples, blogs, and more.

[Top](#)

What's new

[This documentation is preliminary and is subject to change.]

Here are the features and functionality that were added, removed, and changed for each release of Live Connect.

- [November 2011 release](#)
- [May 2011 release](#)

November 2011 release

This release of Live Connect, currently scheduled for November 2011, adds these features and functionality to the May 2011 release.

- **Contacts:** Apps can create contacts. (This is in addition to reading contact info and friend info in the first release). For details, see [Contacts](#).
- **Calendar:** Apps can create, read, update, and delete a Hotmail user's calendars and calendar events. Apps can also subscribe to publicly-available calendars like holidays. For details, see [Calendars](#).
- **SkyDrive:** Apps can create, read, update, and delete a SkyDrive user's folders, files, albums, photos, and videos. Apps can also read comments and tags. For details, see [SkyDrive \(files and photos\)](#).
- **Windows Live Messenger:** Apps can exchange instant messages and otherwise interact with a Messenger user and his or her buddies by using Extensible Messaging and Presence Protocol (XMPP), an open standard for real-time communication. For details, see [Messenger \(instant messaging\)](#).
- **Settings management:** You can use the [Live Connect app management site for Metro style apps](#) to create client IDs for Metro style apps. For details, see [Getting a client ID and configuring your app](#).
- **Support for additional app types.** The Live Connect APIs now support Metro style apps and Windows Phone apps. We continue to support client-side websites and scripts with JavaScript. We also provide a general-purpose Representational State Transfer (REST) implementation. For details, see [Metro style apps](#), [Windows Phone apps](#), [Web apps](#), [Mobile and Windows desktop apps](#), [Referencing the APIs](#), [Managed API](#), [JavaScript API](#), and [REST API](#).

[Top](#)

May 2011 release

The May 2011 release of Live Connect (formerly known as Windows Live Messenger Connect, Version 5.0) included these features and functionality:

- **OAuth 2.0:** We support this open web authorization standard. This makes it easier for your apps to work with Hotmail, Messenger, SkyDrive and other

services that are compatible with Live Connect by using an authentication protocol that is used by other web platforms like Facebook and Google. For details, see [OAuth 2.0](#).

- **Single sign-on:** When a user has already signed in with their Microsoft account from Hotmail, SkyDrive, MSN, or elsewhere, they don't have to sign in again when they come to your website. For details, see [Signing users in](#) and [Single-sign-on and OAuth for apps](#).
- **Improved user experience:** Compared to past releases, we're more explicit about each permission that the user grants to your app, which puts users in more control of their info. We also provide multiple versions of the user interface where users give their okay for apps to access their info. These user interface options include a touch-friendly version. For details, see [Obtaining user consent](#).
- **Profile:** Apps can read a user's profile info. For details, see [Identity \(profiles\)](#).
- **Contacts:** Apps can read a Hotmail user's contact and friend info. For details, see [Contacts](#). Apps can also determine which of a Hotmail user's friends are also users of a website that both the user and the user's friends are members of. For details, see [Friend finder](#).
- **Calendar:** Apps can create events in a Hotmail user's default calendar. For details, see [Calendars](#).
- **Status updates:** Apps can update a user's status message. For details, see [Sharing a user's status](#).
- **SkyDrive:** Apps can read a SkyDrive user's folder, album, photo, video, comment, and tag info. For details, see [SkyDrive \(files and photos\)](#).
- **App management:** Compared to past releases, we simplified our [Live Connect app management site](#). For details, see [Getting a client ID and configuring your app](#).
- **Easier coding and improved info transfer.** Compared to past releases, we simplified our REST API by focusing on the JavaScript Object Notation (JSON) data format. At the same time, we decreased the average size of the info that you get back from the Live Connect APIs to speed up performance and decrease network traffic. We also redesigned our JavaScript implementation to require fewer lines of code to do common tasks. For details, see [Referencing the APIs](#), [JavaScript API](#), and [REST API](#).

[Top](#)

Getting a client ID and configuring your app

[This documentation is preliminary and is subject to change.]

Before your app can begin to call the Live Connect APIs, you must configure your app with a unique identifier, which we call a *client ID*. For some app types, you must also configure your app with an associated password, which we call a *client secret*, and a *redirect domain*, a domain that the APIs use to exchange tokens, data, and messages with your app.

To get a client ID—and to get a client secret and specify a redirect domain if needed for your app type—here are your options.

- **Metro style app:** You only need a client ID.
- **Windows Phone app, and a client-side website or script that uses JavaScript:** You need a client ID, and you need to specify a redirect domain.
- **All other app types:** You need a client ID and a client secret, and you need to specify a redirect domain.

You get a client ID and a client secret, and you can specify a redirect domain if needed, at the [Live Connect app management site](#) (or the [Live Connect app management site for Metro style apps](#)).

Once you get a client ID, you may want to do one or more of these.

- **All app types:** Specify settings for the *consent dialog*, which is a user interface component that a user must use to okay your app to work with the user's info.
- **All other app types:** Change a client secret.

Follow these instructions to do these tasks.

- [Get a client ID \(Metro style apps only\)](#)
- [Get a client ID \(all app types except for Metro style apps\)](#)
- [Specify a redirect domain](#)
- [Get a client secret](#)
- [Change a client secret](#)
- [Specify consent dialog settings](#)

Get a client ID (Metro style apps only)

1. With your app open in Microsoft Visual Studio 11 for Windows Developer Preview, in **Solution Explorer**, double-click the package.appxmanifest file to open its properties page.
2. On the **Packaging** tab, note the contents of the **Package Display Name** and **Publisher** boxes.
3. Go to the Live Connect app management site for Metro style apps (<https://manage.dev.live.com/build>).
Note Do not go to the Live Connect app management site at <https://manage.dev.live.com>. That site is not designed to create client IDs for Metro style apps.
4. If prompted, sign in with your Microsoft account credentials.
5. In the page's **Package display name** and **Publisher** boxes, type the values from the preceding step 2, and then click **I accept**.
6. On the next page, note the contents of the **Package name** box.
7. In Visual Studio, go back to your project's package.appxmanifest file's properties page. On the **Packaging** tab, overwrite the current value, if any, of the **Package Name** box with

- the value from the preceding step 6.
8. Save the package.appxmanifest file.

[Top](#)

Get a client ID (all app types except for Metro style apps)

Caution If your app is a Metro style app, don't follow these instructions, or else your client ID won't work. You must follow the instructions in the previous section instead.

1. Go to the [Live Connect app management site](#).
2. If prompted, sign in with your Microsoft account credentials.
3. If you are not immediately prompted to provide the app's display name and primary language, click **Create application**.
4. Type the app's display name and select the app's primary language.
5. Read the Live Connect Terms of Use and the privacy statement, and then click **I accept**. Live Connect creates and then displays the client ID. It should look something like this: 00000000603E0BFE.

[Top](#)

Specify a redirect domain

1. Go to the [Live Connect app management site](#).
2. If prompted, sign in with your Microsoft account credentials.
3. Click the target client ID.
4. Click **Edit settings**.
5. In the **Settings** area, click **API Settings**.
6. In the **Redirect Domain** box, type the domain that the Live Connect APIs will use to exchange tokens, data, and messages with your app (for example, <http://www.contoso.com>).
7. Click **Save**.

[Top](#)

Get a client secret

1. Go to the [Live Connect app management site](#).
2. If prompted, sign in with your Microsoft account credentials.
3. Click the target client ID.
4. Click **Edit settings**.
5. On the application summary page, the client secret displays. It should look something like this: qXipuPomaauItsIsmwtKZ2YacGZtCyXD.

[Top](#)

Change a client secret

You can swap your current client secret for a new client secret if you think your current client secret is compromised. Or, your organization may require you to periodically change your client secret key for security reasons.

1. Go to the [Live Connect app management site](#).
2. If prompted, sign in with your Microsoft account credentials.
3. Click the target client ID.
4. Click **Edit settings**.

5. In the **Settings** area, click **API Settings**.
6. Click **Create a new client secret**.
7. Click **OK**. A new client secret is created and displays.

Note Both the old and new client secrets will continue to work until you follow the next set of steps.

When you're ready to stop using the old client secret, do this:

1. Make sure that all copies of your app are using the new client secret.
2. Go to the [Live Connect app management site](#).
3. If prompted, sign in with your Microsoft account credentials.
4. Click the target client ID.
5. In the **Settings** area, click **API Settings**.
6. Click **Activate**.
7. Click **OK**.

Caution Once you click **Activate**, the old client secret won't work anymore. This means that all copies of your app that rely on the old client secret won't work anymore, either.

[Top](#)

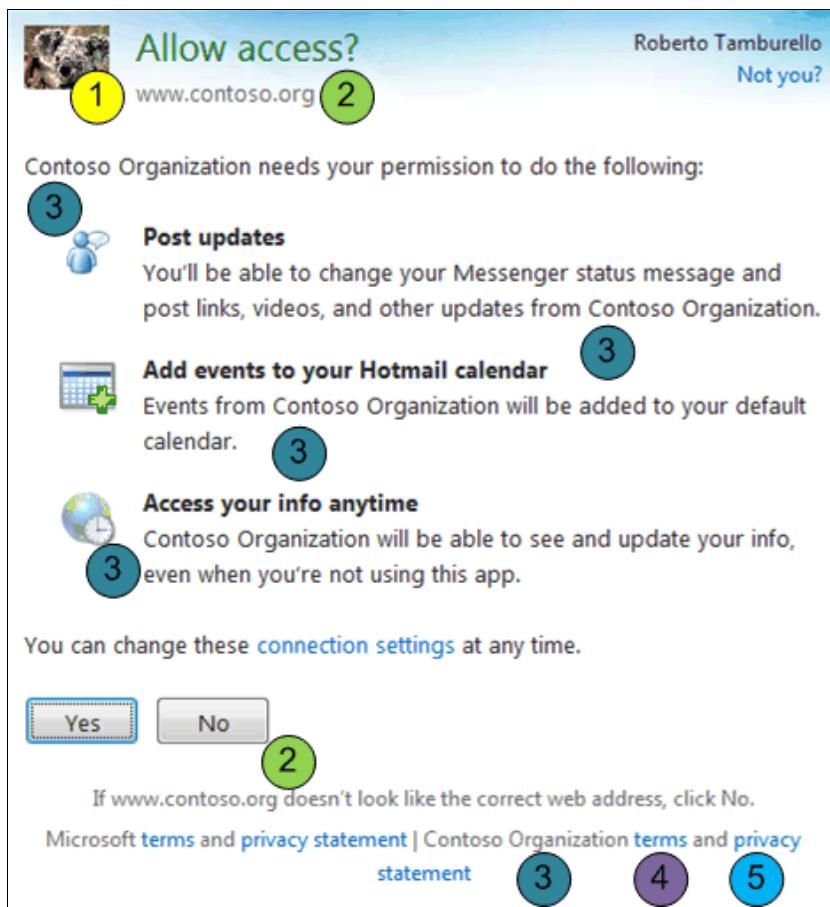
Specify consent dialog settings

After users sign in from your app to a service that is compatible with Live Connect, Live Connect displays the **Allow access** dialog (also called the *consent dialog*), which explains to the user the types of info that your app wants to access. (For more info about the consent dialog, see [Obtaining user consent](#).) To customize the items in the consent dialog, you specify these settings for your app in the [Live Connect app management site](#) (or the [Live Connect app management site for Metro style apps](#)).

Consent dialog	Live Connect app management site setting
1. Your app's logo	Basic Information page, Application logo image
2. Your app's domain name, which appears in several places in the consent dialog	API Settings page, Redirect Domain box
3. Your app's name, which appears in several places in the consent dialog	Basic Information page, Application name box
4. Your app's terms hyperlink	Basic Information page, Terms of service URL box
5. Your app's privacy statement hyperlink	Basic Information page, Privacy URL box

Note If the **Terms of service URL** and **Privacy URL** settings are not specified, the text | APPLICATION_NAME **terms and privacy statement** does not appear in the consent dialog.

The numbered items in the preceding table correspond to the items in the consent dialog as shown here. (This dialog may look a bit different depending on your app type.)



The numbered items also correspond to the items in the Live Connect app management site as shown here.

Settings

Basic Information Text & Logos Localization

API Settings Application name*: Contoso Organization 3

Localization Language: English 4

Application logo:  1

URLs

Terms of service URL: http://www.contoso.org/terms_of_service.html 4

Privacy URL: <http://www.contoso.org/privacy.html> 5

Settings

Basic Information Client ID: 00000004C045607

API Settings Client secret: BweYLkeUNi9AyEFLnVlpdabFw6s2ONAE
[Create a new key](#)

Localization Redirect Domain: www.contoso.org 2

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Metro style apps

[This documentation is preliminary and is subject to change.]

The Live Connect APIs enable Metro style apps to work with info in Hotmail, Windows Live Messenger, SkyDrive, and other services that are compatible with Live Connect.

Note If you're working on a Windows desktop app that is not a Metro style app, this is not the topic you need. Go to [Mobile and Windows desktop apps](#).

Before you can start calling the APIs, you'll need a client ID. For instructions, see [Getting a client ID and configuring your app](#).

- [Referencing the APIs with JavaScript](#)
- [Referencing the APIs with C#](#)
- [Code samples](#)
- [Next steps](#)

Referencing the APIs with JavaScript

To reference the Live Connect APIs from a Metro style app using JavaScript project in Microsoft Visual Studio 11 for Windows Developer Preview, do this:

1. Install the [Live Software Development Kit \(SDK\)](#), if you have not already done so.
2. In your project, in **Solution Explorer**, right-click **References** > **Add Reference**.
3. Click **Windows** > **Extension SDKs** > **Live SDK for Metro style HTML Applications**.
4. Click **Add**, and then click **Close**.
5. If you want to enable Microsoft IntelliSense in the default.html file, add this <script> element within the <head> element.

```
<script src="ms-wwa:///LiveSDKHTML.5.0/js/wl.js"></script>
```

6. If you want to enable IntelliSense in any other JavaScript (.js) file in your project, add this comment to the top of that .js file.

```
/// <reference path="ms-wwa:///LiveSDKHTML.5.0/js/wl.js" />
```

[Top](#)

Referencing the APIs with C#

To reference the Live Connect APIs from a Metro style app project using C# in Visual Studio 11 for Windows Developer Preview, do this:

1. Install the [Live SDK](#), if you have not already done so.
2. In your project, in **Solution Explorer**, right-click **References** > **Add Reference**.
3. Click **Windows** > **Extension SDKs** > **Live SDK for Metro style XAML Applications** > **Add** > **Close**.
4. Add the corresponding **using** statements to your code, like this.

```
using Microsoft.Live;
using Microsoft.Live.Controls;
```

To use the Live Connect sign-in control, you add the corresponding Extensible Application Markup Language (XAML) code to your target XAML file, for instance MainPage.xaml. Then you refer to the sign-in control from your C# code, for instance MainPage.xaml.cs.

1. In the XAML file's <UserControl> opening tag, add this code. It represents the Live Connect sign-in control's XAML namespace.

```
<UserControl
    ...
    xmlns:live="using:Microsoft.Live.Controls"
    ...
    >
```
2. In the body of the <UserControl> tag, for instance inside of a <Grid> tag, add code similar to this. It represents an instance of the sign-in control.

```
<Grid x:Name="LayoutRoot" Background="#FF0C0C0C">
    ...
    <live:SignInButton x:Name="btnSignin" Scopes="wl.signin wl.basic" />
    ...
</Grid>
</UserControl>
```

Important After you add this XAML code, the **Designer** pane may display this error message: **Invalid Markup: Check the Error List for more information**. To continue using the **Designer** pane in design mode, comment out the <live:SignInButton> tag, like this: `<!-- <live:SignInButton x:Name="btnSignin" Scopes="wl.signin wl.basic" /> -->`. Before you run the app, be sure to remove the `<!--` and `-->` comment markers from the <live:SignInButton> tag, or else the sign-in button

will not display when you run the app.

3. In your C# code, you can now refer to the sign-in control by using the value of the sign-in control's **Name** property. In this case, the sign-in control's name is `btnSignin`.

```
...
btnSignin.SessionChanged +=  
    new EventHandler<LiveConnectSessionChangedEventArgs>(btnSignin_SessionChanged);  
...
```

Note Visual Studio 11 for Windows Developer Preview currently doesn't support adding the sign-in control to the **Toolbox**.

[Top](#)

Code samples

To get you started, here's some basic code that shows how to reference the APIs, invite a user to sign in, get the signed-in user's permission to get his or her basic profile info, and display a greeting that includes the user's first and last names.

[JavaScript]

default.html

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8" />  
    <title>Metro style JavaScript Code Sample</title>  
    <!-- WinJS references -->  
    <link rel="stylesheet" href="/winjs/css/ui-dark.css" />  
    <script src="/winjs/js/base.js"></script>  
    <script src="/winjs/js/wwaapp.js"></script>  
    <!-- Metro style JavaScript Code Sample references -->  
    <script src="/js/default.js"></script>  
    <!-- Developer-defined references -->  
    <script src="ms-wwa:///LiveSDKHTML.5.0/js/wl.js"></script>  
</head>  
<body data-design-activate="defaultPage.activated">  
    <h1>Metro style JavaScript Code Sample</h1>  
    <div id="signin"></div>  
    <label id="info"></label>  
    <script>  
        WL.Event.subscribe("auth.login", onLoginComplete);  
        WL.init();  
        var scopes = ["wl.signin", "wl.basic"];  
        WL.ui({  
            name: "signin",  
            element: "signin",  
            scope: scopes  
        });  
        function onLoginComplete(session) {  
            if (!session.error) {  
                WL.api({  
                    path: "me",  
                    method: "GET",  
                }, onAPICalled);  
            }  
            else {  
                document.getElementById("info").innerText =  
                    "Error signing in: " + session.error;  
            }  
        }  
        function onAPICalled(result) {  
            if (!result.error) {  
                document.getElementById("info").innerText =  
                    "Hello, " + result.first_name + " " + result.last_name + "!";  
            }  
            else {  
                document.getElementById("info").innerText =  
                    "Error getting info: " + result.error;  
            }  
        }  
    </script>  
</body>  
</html>
```

[C#]

MainPage.xaml (default page-level attributes omitted for brevity)

```
<UserControl x:Class="MetroStyleCodeSample.MainPage"  
    ...  
    xmlns:live="using:Microsoft.Live.Controls">  
    <Grid x:Name="LayoutRoot" Background="#FF0C0C0C">  
        <live:SignInButton x:Name="btnSignin" Scopes="wl.signin wl.basic" />  
        <TextBlock Height="32" Foreground="White" HorizontalAlignment="Left" Margin="8,76,0,0" Name="infoTextBlock" VerticalAlignment="Top" />  
    </Grid>  
</UserControl>  
  
 MainPage.xaml.cs  
  
using System;  
using Microsoft.Live;  
using Microsoft.Live.Controls;  
  
namespace MetroStyleCodeSample  
{
```

```

partial class MainPage
{
    private LiveConnectClient client;

    public MainPage()
    {
        InitializeComponent();
        this.btnAddSession.SessionChanged += new EventHandler<LiveConnectSessionChangedEventArgs>(btnAddSession_SessionChanged);
    }

    private void btnAddSession_SessionChanged(object sender, LiveConnectSessionChangedEventArgs e)
    {
        if (e.Session != null && e.Session.Status == LiveConnectSessionStatus.Connected)
        {
            client = new LiveConnectClient(e.Session);
            infoTextBlock.Text = "Signed in.";
            client.GetAsync("me", null);
        }
        else
        {
            infoTextBlock.Text = "Not signed in.";
        }
    }

    private void btnAddSession_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
    {
        if (e.Error == null)
        {
            dynamic meResult = e.Result;
            if (meResult.first_name != null && meResult.last_name != null)
            {
                infoTextBlock.Text = "Hello " +
                    meResult.first_name + " " +
                    meResult.last_name + "!";
            }
            else
            {
                infoTextBlock.Text = "Hello, signed-in user!";
            }
        }
        else
        {
            this.infoTextBlock.Text = "Error calling API: " +
                e.Error.ToString();
        }
    }
}

```

Tip

The **GetCompleted** event handler in the preceding example uses the **dynamic** keyword to display results. The **dynamic** keyword is supported here only for Metro style apps using C#. In [Windows Phone apps](#), you'll notice a different approach as a Windows Phone-friendly alternative:

```

private void btnAddSession_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        if (e.Result.ContainsKey("first_name") && e.Result.ContainsKey("last_name"))
        {
            if (e.Result["first_name"] != null && e.Result["last_name"] != null)
            {
                this.infoTextBlock.Text =
                    "Hello, " +
                    e.Result["first_name"].ToString() + " " +
                    e.Result["last_name"].ToString() + "!";
            }
        }
        else
        {
            infoTextBlock.Text = "Hello, signed-in user!";
        }
    }
    else
    {
        infoTextBlock.Text = "Error calling API: " +
            e.Error.ToString();
    }
}

```

To learn what this code does, see [Core concepts](#) and [Identity \(profiles\)](#).

To learn how to use many of the code examples throughout this documentation in the context of a larger reference code sample, see [Working with the code examples](#).

[Top](#)

Next steps

From here, we suggest that you continue your learning with the info in [Core concepts](#), [Identity \(profiles\)](#), [Hotmail \(contacts and calendars\)](#), [SkyDrive \(files and photos\)](#), and [Messenger \(instant messaging\)](#).

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Windows Phone apps

[This documentation is preliminary and is subject to change.]

The Live Connect APIs support enabling Windows Phone 7.0 and 7.5 apps to work with info in Hotmail, Windows Live Messenger, SkyDrive, and other services that are compatible with Live Connect.

Note If you're working on a mobile app that doesn't run on Windows Phone, this is not the topic you need. Go to [Mobile and Windows desktop apps](#).

Tip Before you can start coding Windows Phone apps, you'll need the right tools. For details, see [App Hub - start coding for Windows Phone](#).

Before you can start calling the Live Connect APIs, you'll need a client ID and a redirect domain. For instructions, see [Getting a client ID and configuring your app](#).

- [Referencing the APIs with C#](#)
- [Code sample](#)
- [Next steps](#)

Referencing the APIs with C#

To reference the Live Connect APIs from a Windows Phone project in Microsoft Visual Studio 2010, do this.

1. Install the [Live Software Development Kit \(SDK\)](#), if you have not already done so.
2. In your project, in **Solution Explorer**, right-click **References** > **Add Reference**.
3. On the **.NET** tab, click **Microsoft.Live.Controls**, press and hold the CTRL key, and click **Microsoft.Live**.
4. Click **OK**.
5. Add the corresponding **using** statements to your code, like this.

```
using Microsoft.Live;
using Microsoft.Live.Controls;
```

To display the Live Connect sign-in control in the Microsoft Visual Studio **Toolbox**, if the control is not already visible, do this.

1. In your project, if the **Toolbox** is not already visible, with the Extensible Application Markup Language (XAML) designer open, click **View** > **Toolbox**.
 2. Right-click a blank area of the **Toolbox**, and click **Choose Items**.
 3. Click **Browse**.
 4. Go to the Live SDK install folder. From there, go to the `\Windows Phone\References\` folder, and click **Microsoft.Live.Controls.dll** > **Open** > **OK**.
- Note** By default, the Live SDK is installed in the `\Live\v5.0\` folder in `\Program Files\Microsoft SDKs\` (on 32-bit computers) or `\Program Files (x86)\Microsoft SDKs\` (on 64-bit computers).

[Top](#)

Code sample

To get you started, here's the most basic of code that shows how to reference the APIs, invite a user to sign in, get the signed-in user's permission to get their basic profile info, and display a greeting with the signed-in user's first and last names.

`MainPage.xaml.cs` (default page-level attributes omitted for brevity; replace `CLIENT_ID` with your app's client ID and `REDIRECT_URL` with your app's redirect URL)

```
<phone:PhoneApplicationPage
  ...
  xmlns:my="clr-namespace:Microsoft.Live.Controls;assembly=Microsoft.Live.Controls">
  <Grid x:Name="LayoutRoot" Background="Transparent">
    <my:SignInButton Name="btnSignin" ClientId="CLIENT_ID" Scopes="wl.signin wl.basic" RedirectUri="REDIRECT_URL" Branding="Windows"
      <TextBlock Height="32" HorizontalAlignment="Left" Margin="12,78,0,0" Name="infoTextBlock" Text="" VerticalAlignment="Top" Width="150" />
  </Grid>
</phone:PhoneApplicationPage>

MainPage.xaml.cs

using System;
using System.Windows;
using System.Collections.Generic;
using Microsoft.Phone.Controls;
using Microsoft.Live;
using Microsoft.Live.Controls;

namespace WindowsPhoneCodeSample
{
    public partial class MainPage : PhoneApplicationPage
    {
        private LiveConnectClient client;

        public MainPage()
        {
            InitializeComponent();
        }

        private void btnSignin_SessionChanged(object sender, LiveConnectSessionChangedEventArgs e)
        {
            if (e.Session != null &&
```

```

        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        client = new LiveConnectClient(e.Session);
        infoTextBlock.Text = "Signed in.";
        client.GetCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(btnSignin_GetCompleted);
        client.GetAsync("me", null);
    }
    else
    {
        infoTextBlock.Text = "Not signed in.";
        client = null;
    }
}

void btnSignin_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        if (e.Result.ContainsKey("first_name") &&
            e.Result.ContainsKey("last_name"))
        {
            if (e.Result["first_name"] != null &&
                e.Result["last_name"] != null)
            {
                infoTextBlock.Text =
                    "Hello, " +
                    e.Result["first_name"].ToString() + " " +
                    e.Result["last_name"].ToString() + "!";
            }
            else
            {
                infoTextBlock.Text = "Hello, signed-in user!";
            }
        }
        else
        {
            infoTextBlock.Text = "Error calling API: " +
                e.Error.ToString();
        }
    }
}
}

```

To learn what this code does, see [Core concepts](#) and [Identity \(profiles\)](#).

To learn how to use many of the code examples throughout this documentation in the context of a larger reference code sample, see [Working with the code examples](#).

[Top](#)

Next steps

From here, we suggest continuing your learning with the info in [Core concepts](#), [Identity \(profiles\)](#), [Hotmail \(contacts and calendars\)](#), [SkyDrive \(files and photos\)](#), and [Messenger \(instant messaging\)](#).

[Top](#)

Web apps

[This documentation is preliminary and is subject to change.]

The Live Connect APIs support enabling client-side and server-side websites and scripts to work with info in Hotmail, Windows Live Messenger, SkyDrive, and other services that are compatible with Live Connect. An example of a client-side or server-side website or script is a web browser or a web browser control that enables a user to interact with a website.

Note If you're working on a Metro style app, a Windows Phone app, or a mobile app (even if those apps include website implementations), this is not the topic you need. Go to [Metro style apps](#), [Windows Phone apps](#), or [Mobile and Windows desktop apps](#), respectively.

Before you can start calling the Live Connect APIs, you'll need a client ID, a redirect domain, and a client secret. For instructions, see [Getting a client ID and configuring your app](#).

- [Choosing an API implementation](#)
- [Code samples](#)
- [Testing web apps locally](#)
- [Next steps](#)

Choosing an API implementation

In general, if you're working on a client-side website or script that uses JavaScript, you use our JavaScript implementation. And in general, if you're working on a server-side website or script, or if you're working on a client-side website or script that can't use JavaScript, you use our Representational State Transfer (REST) implementation. For details, see [JavaScript API](#) and [REST API](#).

[Top](#)

Code samples

If you're working on a client-side website or script that uses JavaScript, here's the most basic of code that shows how to reference the Live Connect APIs, invite a user to sign in, get the signed-in user's permission to get their basic profile info, and display a greeting with the signed-in user's first and last names.

default.htm

```
<!DOCTYPE html>
<html>
    <head>
        <title>Client-Side JavaScript Code Sample</title>
        <script src="constants.js"></script>
        <script src="//js.live.net/v5.0/wl.js"></script>
    </head>
    <body>
        <div id="signin"></div>
        <label id="info"></label>
        <script>
```

```

WL.Event.subscribe("auth.login", onLoginComplete);
WL.init({
    client_id: APP_CLIENT_ID,
    redirect_uri: REDIRECT_URL,
    response_type: "token"
});
var scopes = ["wl.signin", "wl.basic"];
WL.ui({
    name: "signin",
    element: "signin",
    scope: scopes
});
function onLoginComplete(session) {
    if (!session.error) {
        WL.api({
            path: "me",
            method: "GET"
        }, onAPICalled);
    }
    else {
        document.getElementById("info").innerText =
            "Error signing in: " + session.error;
    }
}
function onAPICalled(result) {
    if (!result.error) {
        document.getElementById("info").innerText =
            "Hello, " + result.first_name + " " + result.last_name +
        "!";
    }
    else {
        document.getElementById("info").innerText =
            "Error getting info: " + result.error;
    }
}
</script>
</body>
</html>

```

callback.htm

```

<!DOCTYPE html>

<html>
    <head>
        <title></title>
    </head>
    <body>
        <script src="//js.live.net/v5.0/wl.js"></script>
    </body>
</html>

```

constants.js (replace *YOUR_CLIENT_ID* with your app's client ID and *YOUR_REDIRECT_URL* with your app's redirect URL)

```

var APP_CLIENT_ID = "YOUR_CLIENT_ID";
var REDIRECT_URL = "YOUR_REDIRECT_URL";

```

To learn what this code does, see [Core concepts](#) and [Identity \(profile\)](#).

To learn how to use many of the code examples throughout this documentation in the context of a larger reference code sample, see [Working with the code examples](#).

If you're working on a server-side website or script, or if you're working on a client-side website or script that can't use JavaScript, see the REST code samples in [Server-side scenarios](#).

[Top](#)

Testing web apps locally

When you are developing a web app, it is often convenient to test the app on a local computer before deploying it to a web server. However, the Live Connect APIs requires the redirect URL to use the same domain at which the app's client ID was configured. Since your local computer's web server likely uses a default address like **http://localhost**, you must map your domain to the Internet Protocol (IP) address of your local web server. To do this, you add a mapping to the **hosts** file on your computer. (The job of the **hosts** file is to map host names to IP addresses.) On Windows computers, the hosts file is located at **C:\Windows\System32\drivers\etc\hosts**. On Mac OSX and most Unix computers it is located at **/private/etc/hosts**. The format is the IP address, followed by the domain to map, like this.

```
127.0.0.1      MyDomain.com  
127.0.0.1      AnotherDomain.com
```

The preceding example demonstrates mapping two arbitrary domain names to the IP address for **localhost**. With these mappings in place, every time you navigate to **MyDomain.com** or **MyOtherDomain.com**, you will be redirected to 127.0.0.1 (**http://localhost**). The mappings in the **hosts** file apply to all web navigation including redirect URLs returned by the Live Connect APIs. This works because the Live Connect APIs do not check the actual IP address that is in use, it only verifies the domain name. So if your client ID was configured with a redirect URL of **http://MyTestingDomain.com/MyApp/callback.htm**, you would update your **hosts** file with the following: **127.0.0.1 MyTestingDomain.com** Now if you enter "MyTestingDomain.com" into your browser, it will resolve to **http://localhost**, enabling you to use your local web server for testing.

Important

Elevated (or root) privileges are required to edit the **hosts** file, regardless of platform.

Tip

You can map an externally-available domain to your local server IP address, but you must comment out the mapping when you want to view the actual website. This is only applicable to the computer on which the hosts file was modified. This is a handy trick, since it makes it easy to switch between your externally-available website and your local test website.

[Top](#)

Next steps

From here, we suggest starting with the info in [Core concepts](#), [Identity \(profiles\)](#), [Hotmail \(contacts and calendars\)](#), [SkyDrive \(files and photos\)](#), and [Messenger \(instant messaging\)](#). And if you're writing code for a server-side website or script,

be sure to see [Server-side scenarios](#), too.

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Working with the code examples

[This documentation is preliminary and is subject to change.]

To fully leverage the smaller code examples in the [Core concepts](#), [Identity \(profiles\)](#), [Hotmail \(contacts and calendars\)](#), and [SkyDrive \(files and photos\)](#) sections in your apps, you can use our code examples in the context of larger code reference samples that we provide here.

- [JavaScript code examples](#)
- [C# code examples](#)
- [REST code examples](#)
- [Additional code examples](#)

JavaScript code examples

To use the smaller JavaScript code examples, we've written the following larger code reference sample that you can leverage for your own apps.

default.htm (for Metro style apps using JavaScript only)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Metro style JavaScript Code Sample</title>
    <!-- WinJS references -->
    <link rel="stylesheet" href="/winjs/css/ui-dark.css" />
    <script src="/winjs/js/base.js"></script>
    <script src="/winjs/js/waapp.js"></script>
    <!-- Metro style JavaScript Code Sample references -->
    <script src="/js/default.js"></script>
    <!-- Developer-defined references -->
    <script src="ms-wwa:///LiveSDKHTML.5.0/js/wl.js"></script>
    <script src="/js/gather_info.js"></script>
</head>
<body data-design-activate="defaultPage.activated">
    <div id="signin"></div>
    <br />
    <label id="infoLabel"></label>
    <script>
        WL.Event.subscribe("auth.login", onLoginComplete);
        WL.init();
        var scopes = ["wl.signin"];
        WL.ui({
            name: "signin",
            element: "signin",
            scope: scopes
        });
        function onLoginComplete(session) {
            var session = WL.getSession();
            if (session.error) {
                document.getElementById("infoLabel").innerText =
                    "Error signing in: " + session.error;
            }
            else {
                document.getElementById("infoLabel").innerText =
                    "Signed in. Scopes = " + session.scope;
            }
        }
    </script>
</body>
</html>
```

Note The preceding code sample prompts the user to consent to the **wl.signin** scope only. To write code to prompt the user to consent to additional scopes later, see the code examples in the [Obtaining user consent](#) topic's "Requesting additional scopes" section.

default.htm (for client-side websites and scripts using JavaScript only)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Client-Side JavaScript Dashboard</title>
        <script src="//js.live.net/v5.0/wl.js"></script>
        <script src="gather_info.js"></script>
        <script src="constants.js"></script>
    </head>
    <body>
        <div id="signin"></div>
        <br />
        <label id="infoLabel"></label>
        <script>
            WL.Event.subscribe("auth.login", onLoginComplete);
            WL.init({
                client_id: CLIENT_ID,
                redirect_uri: REDIRECT_URL,
                response_type: "token"
            });
            var scopes = ["wl.signin"];
            WL.ui({
                name: "signin",
                element: "signin",
                scope: scopes
            });
            function onLoginComplete() {
                var session = WL.getSession();
            }
        </script>
    </body>
</html>
```

```

        if (session.error) {
            document.getElementById("infoLabel").innerText =
                "Error signing in: " + session.error;
        }
        else {
            document.getElementById("infoLabel").innerText =
                "Signed in.";
        }
    }
</script>
</body>
</html>

```

Note The preceding code sample prompts the user to consent to the **wl.signin** scope only. To write code to prompt the user to consent to additional scopes later, see the code examples in the [Obtaining user consent](#) topic's "Requesting additional scopes" section.

constants.js (for client-side websites and scripts using JavaScript only; replace *YOUR_CLIENT_ID* with your app's client ID and *YOUR_REDIRECT_URL* with your app's redirect URL)

```

var CLIENT_ID = "YOUR_CLIENT_ID";
var REDIRECT_URL = "YOUR_REDIRECT_URL";

callback.htm (for client-side websites and scripts using JavaScript only)

<!DOCTYPE html>

<html>
    <head>
        <title></title>
    </head>
    <body>
        <script src="//js.live.net/v5.0/wl.js"></script>
    </body>
</html>

```

To make your code easier to manage, in a separate file, named *gather_info.js* in this code sample, you could put code that runs when users interact with the app, like clicking buttons.

[Top](#)

C# code examples

To use the smaller C# code examples, we've written the following larger code reference sample that you can leverage for your own apps.

MainPage.xaml (for Metro style apps using C# only; default page-level attributes and other extraneous Extensible Application Markup Language (XAML) code omitted for brevity)

```

<UserControl x:Class="MetroStyleCodeSample.MainPage"
    ...
    xmlns:live="using:Microsoft.Live.Controls"
    ...
    <Grid x:Name="LayoutRoot" Background="#FF0C0C0C">
        ...
        <live:SignInButton x:Name="btnSignin" Scopes="wl.signin wl.basic" />
        ...
        <TextBlock Height="32" Foreground="White" HorizontalAlignment="Left" Margin="8,76,0,0" Name="infoTextBlock" VerticalAlignment="Top" Width="150" />
        ...
    </Grid>
    ...
</UserControl>

```

Note The preceding code sample prompts the user to consent to the **wl.signin** and **wl.basic** scopes only. To write code to prompt the user to consent to additional scopes later, see the code examples in the [Obtaining user consent](#) topic's "Requesting additional scopes" section.

MainPage.xaml (for Windows Phone apps using C# only; default page-level attributes and other extraneous XAML code omitted for brevity; replace *CLIENT_ID* with your app's client ID and *REDIRECT_URL* with your app's redirect URL)

```

<phone:PhoneApplicationPage
    ...
    xmlns:my="clr-namespace:Microsoft.Live.Controls;assembly=Microsoft.Live.Controls">
    ...
    <Grid x:Name="LayoutRoot" Background="Transparent">
        ...
        <my:SignInButton Name="btnSignin" ClientId="CLIENT_ID" Scopes="wl.signin wl.basic" RedirectUri="REDIRECT_URL" Branding="Windows" />
        ...
        <TextBlock Height="32" HorizontalAlignment="Left" Margin="12,78,0,0" Name="infoTextBlock" Text="" VerticalAlignment="Top" Width="150" />
        ...
    </Grid>
    ...
</phone:PhoneApplicationPage>

```

Note The preceding code sample prompts the user to consent to the **wl.signin** and **wl.basic** scopes only. To write code to prompt the user to consent to additional scopes later, see the code examples in the [Obtaining user consent](#) topic's "Requesting additional scopes" section.

MainPage.xaml.cs (for Metro style apps using C# only)

```

using System;
using System.IO;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Microsoft.Live;
using Microsoft.Live.Controls;

namespace MetroStyleCodeSample
{

```

```

partial class MainPage
{
    private LiveAuthClient auth;
    private LiveConnectClient client;
    private LiveConnectSession session;

    public MainPage()
    {
        InitializeComponent();
        btnSignin.SessionChanged += new EventHandler<LiveConnectSessionChangedEventArgs>(btnSignin_SessionChanged);
    }

    void btnSignin_SessionChanged(object sender, LiveConnectSessionChangedEventArgs e)
    {
        if (e.Session != null && e.Session.Status == LiveConnectSessionStatus.Connected)
        {
            session = e.Session;
            client = new LiveConnectClient(session);
            infoTextBlock.Text = "Signed in.";
        }
        else
        {
            infoTextBlock.Text = "Not signed in.";
            client = null;
        }
    }
}

```

MainPage.xaml.cs (for Windows Phone apps using C# only)

```

using System;
using System.Collections.Generic;
using System.Windows;
using Microsoft.Phone.Controls;
using Microsoft.Live;
using Microsoft.Live.Controls;

namespace WindowsPhoneCodeSample
{
    public partial class MainPage : PhoneApplicationPage
    {
        private LiveAuthClient auth;
        private LiveConnectClient client;
        private LiveConnectSession session;

        public MainPage()
        {
            InitializeComponent();
        }

        private void btnSignin_SessionChanged(object sender, LiveConnectSessionChangedEventArgs e)
        {
            if (e.Session != null && e.Session.Status == LiveConnectSessionStatus.Connected)
            {
                session = e.Session;
                client = new LiveConnectClient(session);
                infoTextBlock.Text = "Signed in.";
            }
            else
            {
                infoTextBlock.Text = "Not signed in.";
                client = null;
            }
        }
    }
}

```

Constants.cs (for Windows Phone apps using C# only; replace *YOUR_CLIENT_ID* with your app's client ID and *YOUR_REDIRECT_URL* with your app's redirect URL)

```

using Microsoft.Phone.Controls;

namespace WindowsPhoneCodeSample
{
    public partial class MainPage : PhoneApplicationPage
    {
        public const string CLIENT_ID = "YOUR_CLIENT_ID";
        public const string REDIRECT_URL = "YOUR_REDIRECT_URL";
    }
}

```

Note For Windows Phone apps, the Constants.cs file is needed to provide a client ID and a redirect URL. These are needed for the app to request additional scopes after a user has initially signed in to the app and consented to an initial set of scopes.

[Top](#)

REST code examples

To use the Representational State Transfer (REST) code examples, we're following this format.

```

METHOD https://apis.live.net/v5.0/REST_PATH?access_token=ACCESS_TOKEN
HEADER_FIELD: HEADER_VALUE
BODY

```

In this format, the uppercased characters represent the following:

- *METHOD* represents an HTTP request method (or *verb*) that the Live Connect REST API supports, like GET, POST, UPDATE, and DELETE.
- *REST_PATH* represents a path that our REST implementation supports, like me, me/albums, and me/photos.
- *ACCESS_TOKEN* represents a valid access token that Live Connect issues. This access token enables a certain client ID to work with a certain set of user info for a certain time period.
- *HEADER_FIELD* represents an HTTP header field that our REST implementation supports for the corresponding HTTP verb, like Content-Type.
- *HEADER_VALUE* represents an HTTP header value that our REST implementation supports for the corresponding HTTP header field, like application/json for Content-Type.
- *BODY* represents a body value that our REST implementation supports for the associated HTTP verb, like this for creating an **Event** object.

```
{  
    name: "Global Project Risk Management Meeting",  
    description: "Generate and assess risks for the project",  
    start_time: "2011-04-20T01:00:00-07:00",  
    end_time: "2011-04-20T02:00:00-07:00",  
    location: "Building 81, Room 9981, 123 Anywhere St., Redmond WA 19599",  
    is_all_day_event: false,  
    availability: "busy",  
    visibility: "public"  
}
```

Putting this all together, here are the input parameters for a sample REST call that creates an **Event** object.

```
POST https://apis.live.net/v5.0/me/events?access_token=EwBQ... .AA==
```

```
Content-Type: application/json
```

```
{  
    name: "Global Project Risk Management Meeting",  
    description: "Generate and assess risks for the project",  
    start_time: "2011-04-20T01:00:00-07:00",  
    end_time: "2011-04-20T02:00:00-07:00",  
    location: "Building 81, Room 9981, 123 Anywhere St., Redmond WA 19599",  
    is_all_day_event: false,  
    availability: "busy",  
    visibility: "public"  
}
```

You can apply the patterns of these REST input parameters to your app's specific code syntax for making REST calls.

[Top](#)

Additional code examples

We provide lots of additional code examples outside of our documentation, including these.

- The [Live Software Development Kit \(SDK\)](#) contains several code examples. In the Live SDK install folder, go to one or more of these folders.
 - For Metro style apps using JavaScript: \Samples\MetroHTMLSamples\
 - For Metro style apps using C#: \Samples\MetroXAMLSamples\
 - For Windows Phone apps: \Samples\WindowsPhoneSamples\
 - For additional samples: in \Samples\, open the Live Connect Samples shortcut.

Note By default, the Live SDK is installed in the \Live\v5.0\ folder in \Program Files\Microsoft SDKs\ (on 32-bit computers) or \Program Files (x86)\Microsoft SDKs\ (on 64-bit computers).

- [MSDN Samples - SkyDrive Photo API sample for WP7](#)
- [GitHub - liveservices](#)—code examples for ASP.NET websites, Windows desktop apps with C#, Windows Phone apps, PHP scripts, and iOS apps.

[Top](#)

Mobile and Windows desktop apps

[This documentation is preliminary and is subject to change.]

Live Connect does not explicitly support user sign-in and consent in Windows desktop apps, or in mobile apps not written for Windows Phone. However, you can accomplish the same result by using a modified version of the client-side (web) implementation. This walkthrough demonstrates creating a mobile app, or a Windows desktop app, that can facilitate signing in to services that are compatible with Live Connect and accessing user data. The app in this example is a Windows Presentation Foundation (WPF) app written using C# and the .NET Framework version 4. You can build and run it using either Microsoft Visual Studio 2010 or Microsoft Visual Studio 2008.

Important This walkthrough describes development patterns that can be used for Windows desktop apps and for mobile apps that are not written for Windows Phone. In contrast, Metro style apps and Windows Phone apps have built-in support for user sign-in and consent. For more info, see [Metro style apps](#) and [Windows Phone apps](#).

- [Creating a new app](#)
- [App overview](#)
- [Building an app](#)

Creating a new app

When you get a client ID from the Live Connect app management site, leave the **Redirect Domain** field blank, and select **Yes** for the **Mobile client app** option.

Doing so configures your app to use a special redirect URL (<https://oauth.live.com/desktop>), which allows the app to handle redirects after the user has given authorization, and facilitates refreshing the access token. For more info, see [Getting a client ID and configuring your app](#).

Note

[Top](#)

App overview

To implement the client-side authentication flow, desktop apps must use a web browser control. Most development languages include such a control. In this example, our app uses the [System.Windows.Forms.WebBrowser](#) class. After sign-in is complete, all subsequent Live Connect API calls can be accomplished by using the [System.Net.WebRequest](#) class. Use the web browser control to start the sign-in, passing a URL similar to this one.

```
https://oauth.live.com/authorize?client_id=YOUR_CLIENT_ID&scope=YOUR_SCOPE_STRING&response_type=code&redirect_uri=https://oauth.live.com/desktop
```

This URL displays the standard Live Connect sign-in page. Note that the *redirect_uri* parameter specifies the special Live Connect APIs URL designed for desktop apps: <https://oauth.live.com/desktop>. Use this URL whenever you need to supply a redirect Uniform Resource Identifier (URI). The *response_type* and *scope*

parameters are also required. The value of the `response_type` parameter must be set to **code**.

After the user grants consent, the Live Connect APIs return a URI that contains the authorization code. It looks similar to the URI in the following example.

```
https://oauth.live.com/desktop#authorization_code=AUTHORIZATION_CODE
```

Tip If the `wl.offline_access` scope was specified, a refresh token is also provided.

Your app can read the authentication code from the returned URL, and use it to retrieve an access token by making a call, passing a URL similar to this one.

```
https://oauth.live.com/token?client_id=YOUR_CLIENT_ID&redirect_uri=https://oauth.live.com/desktop&code=AUTHORIZATION_CODE&grant_type=authorization_code
```

Finally, your app can read the token data from the returned URL and use that data to make further requests from the Live Connect APIs by using the [System.Net.WebRequest](#) class.

If the `wl.offline_access` was specified in the initial request, your app can refresh the access token by making a call, using a URL like the one shown here.

```
https://oauth.live.com/token?client_id=YOUR_CLIENT_ID&redirect_uri=https://oauth.live.com/desktop&grant_type=refresh_token&refresh_token=REFRESH_TOKEN
```

[Top](#)

Building an app

In this walkthrough you will build a simple WPF app that enables signing in to Live Connect and that retrieves some user data after sign-in is complete. The app displays token information in a logging window. Before beginning this exercise you must visit the [Live Connect app management site](#) and get a client ID and a client secret, if you don't have one already available. Be sure to leave the **Redirect Domain** field blank, and make a note of your client ID and client secret.

► Create a new project

1. Start Visual Studio 2010.
2. Click **File > New Project**.
3. In **Installed Templates**, click **Visual C# > WPF Application**.
4. In the **Name** field, type "DesktopDemo".
5. Click **OK** to create the project.

► Add the browser window

1. In **Solution Explorer**, right-click the project name, select **Add**, then select **New Item**.
2. In the **Add New Item** dialog, select **Window (WPF)**.
3. In the **Name** field, enter **BrowserWindow.xaml**.
4. Click **Add** to add the window to the project.
5. Open **BrowserWindow.xaml** and paste in the following Extensible

Application Markup Language (XAML) code.

```
<Window x:Class="DesktopDemo.BrowserWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Sign In" Height="460" Width="400">
    <Grid>
        <WebBrowser Height="420" HorizontalAlignment="Left" Name="webView"
            VerticalAlignment="Top" Width="380" LoadCompleted="webView_LoadCompleted" />
    </Grid>
</Window>
```

BrowserWindow.xaml contains a *WebBrowser* control, which is used to display the Live Connect consent dialog.

6. Open **BrowserWindow.xaml.cs** and paste in the following code. Be sure to substitute your own client ID for the value of *client_id*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Text.RegularExpressions;

namespace DesktopDemo
{
    public partial class BrowserWindow : Window
    {
        static string scope = "wl.basic";
        static string client_id = "[YOUR_CLIENT_ID]";
        private static Uri signInUrl = new Uri(String.Format(@"https://oauth.live.com/authorize?client_id={0}&redirect_uri=https://oauth.live.com/desktop&response_type=token&scope={1}", client_id, scope));
        MainWindow mainWindow = new MainWindow();

        public BrowserWindow()
        {
```

```

        InitializeComponent();
        webBrowser.Navigate(signInUrl);
    }

    private void webBrowser_LoadCompleted(object sender, System.Windows.Navigation.NavigationEventArgs e)
    {
        if (e.Uri.Fragment.Contains("access_token"))
        {
            if (App.Current.Properties.Contains("responseData"))
            {
                App.Current.Properties.Clear();
            }
            App.Current.Properties.Add("responseData", 1);
            string[] responseAll = Regex.Split(e.Uri.Fragment.Remove(0, 1), "&");

            for (int i = 0; i < responseAll.Count(); i++)
            {
                string[] nvPair = Regex.Split(responseAll[i], "=");
                App.Current.Properties.Add(nvPair[0], responseAll[i]);
            }
            this.Close();
        }
    }
}

```

The code in `BrowserWindow` handles two basic tasks. When the window opens, `BrowserWindow` directs the `WebBrowser` control to the Live Connect sign-in page using the `Uri` value defined in the `signInUrl` variable. It then monitors the **`webBrowser_LoadCompleted`** event handler. After the user signs in, it parses the Representational State Transfer (REST) response and adds it to **`App.Current.Properties`**. Note that **`App.Current.Properties`** was used to keep this example simple. In an actual app you would persist user data by using a different mechanism, such as a database.

7. Save the project.

▶ Add the main window controls

1. In **Solution Explorer**, right-click **References**.
2. Under **Assemblies**, select **System.Web.Extensions**.

3. Click **Add** to add the reference.
4. Open MainWindow.xaml to view the default form.
5. Paste in the following XAML code.

```

<Window x:Class="DesktopDemo.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Desktop App Demo" Height="500" Width="700" Unloaded="Window_Unloaded">
    <Grid>
        <Image Height="110" HorizontalAlignment="Left" Margin="12,12,0,0" Name="imgUser" Stretch="Fill" VerticalAlignment="Top" Width="110" />
        <TextBlock Height="40" HorizontalAlignment="Left" Margin="12,128,0,0" Name="txtBlock_Name" VerticalAlignment="Top" Width="360" FontFamily="Trebuchet MS" FontSize="32" />
        <Button Content="Click to Sign In" Height="23" HorizontalAlignment="Center" Margin="262,204,259,0" Name="btnSignIn" VerticalAlignment="Top" Width="157" Click="btnSignIn_Click" />
        <TextBox Height="172" HorizontalAlignment="Left" Margin="12,244,0,0" Name="txtTokens" VerticalAlignment="Top" Width="654" Visibility="Collapsed" TextWrapping="Wrap" VerticalScrollBarVisibility="Auto" />
        <Button Content="Clear App Settings" Height="23" HorizontalAlignment="Left" Margin="12,426,0,0" Name="btnClear" VerticalAlignment="Top" Width="162" Click="btnClear_Click" />
    </Grid>
</Window>

```

6. Save and close MainWindow.xaml.

► Add initialization and request code

1. In **Solution Explorer**, open MainWindow.xaml.cs.
2. Add the following **using** statements after the default **using** statements.

```

using System.Web.Script.Serialization;
using System.Net;
using System.IO;

```

3. At the top of the main class, add the following variables.

```

private static string requestUrl = @"https://apis.live.net/v5.0/";
public Dictionary<string, string> userData = new Dictionary<string, string>();

```

The *requestUrl* variable stores the base request URL, which we will use to build request URLs later. *userData* is a **Dictionary** value that we will use

to store user data that is returned by the request.

4. Add the following line of code immediately following the **InitializeComponent** call in the default function.

```
setUserUI();
```

This is a call to the **setUserUI** function, which requests the user's data and populates the app's UI.

5. Add the following functions immediately after the **MainWindow** function.

```
private void setUserUI()
{
    if (App.Current.Properties.Contains("responseData"))
    {
        makeRequest(requestUrl + "me?" + App.Current.Properties["access_token"]);
    }
}

private void makeRequest(string requestUrl)
{
    WebClient wc = new WebClient();
    wc.DownloadStringCompleted += new DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
    wc.DownloadStringAsync(new Uri(requestUrl));
}

void client_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
{
    userData = deserializeJson(e.Result);
    changeView();
}

private Dictionary<string, string> deserializeJson(string json)
{
    var jss = new JavaScriptSerializer();
    var d = jss.Deserialize<Dictionary<string, string>>(json);
    return d;
}
```

The **setUserUI** function first checks to see whether the app properties contain any response data. If there is data, it calls **makeRequest**, which uses a **WebClient** object to make the request. The **client_DownloadStringCompleted** event handler is triggered upon

completion of the **WebClient** request. It calls **deserializeJson**, which uses **JavaScriptSerializer** to deserialize the results and store them in the *userData* dictionary as name-value pairs.

6. Save MainWindow.xaml.cs.

► Add code to update the UI

1. Paste the following code after the **client_DownloadStringCompleted** event handler.

```
private void changeView()
{
    btnSignIn.Visibility = Visibility.Collapsed;
    txtTokens.Visibility = Visibility.Visible;
    if (userData != null)
    {
        txtBlock_Name.Text = userData["name"];
        string imgUrl = requestUrl + "me/picture?" + App.Current.Properties["access_token"];
        imgUser.Source = new BitmapImage(new Uri(imgUrl, UriKind.RelativeOrAbsolute));
        txtTokens.Text += App.Current.Properties["access_token"] + "\r\n\r\n";
        txtTokens.Text += App.Current.Properties["authentication_token"];
    }
}
```

The **changeView** function updates the user interface to display the user data that is returned by the request.

2. Paste the code for the following two functions after the **changeView** function.

```
private void btnSignIn_Click(object sender, RoutedEventArgs e)
{
    BrowserWindow browser = new BrowserWindow();
    browser.Closed += new EventHandler(browser_Closed);
    browser.Show();
}

private void btnClear_Click(object sender, RoutedEventArgs e)
{
    App.Current.Properties.Clear();
    btnSignIn.Visibility = Visibility.Visible;
    txtTokens.Visibility = Visibility.Collapsed;
    txtTokens.Text = "";
}
```

```
    imgUser.Source = null;
    txtBlock_Name.Text = "";
}
```

These are the event handlers for each of the buttons. The **btnSignIn_Click** handler launches the **BrowserWindow** window, and the **btnClear_Click** handler deletes all session data and refreshes the UI. (**btnClear_Click** is intended as a convenience for when you are experimenting with the code).

3. Last, paste the following code after **btnClear_Click**.

```
void browser_Closed(object sender, EventArgs e)
{
    setUserUI();
}

private void Window_Unloaded(object sender, RoutedEventArgs e)
{
    Application.Current.Shutdown();
}
```

The **browser_Closed** event handler calls **setUserUI**, which calls **makeRequest** to make the request for user data.

4. Now it's time to build and test the project. Press F5 to build and run the project. You should see the Live Connect sign-in page displayed in the *WebBrowser* control, and the sign-in URL in the browser window. Sign in using a Microsoft account, and the app displays the user's name and photo, and also the `access_token` and `authentication_token` values.

[Top](#)

Community resources

[This documentation is preliminary and is subject to change.]

If you have questions about calling the Live Connect APIs from your code, or if you want to learn more beyond this documentation, see the following resources:

- [Live Connect website](#)—Provides links to helpful tools, code samples, blogs, and additional learning content.
- [Live Connect Forum](#)—Offers opportunities to ask technical questions, get answers from subject-matter experts and enthusiasts, and read and search for questions and answers from others.

© 2011 Microsoft Corporation. All rights reserved.

Core concepts

[This documentation is preliminary and is subject to change.]

Before you call the Live Connect APIs, you should feel comfortable with a set of core concepts that apply to all app types. These core concepts include how to reference the Live Connect APIs; how to sign in a user; how to get the signed-in user's permission for your app to work with his or her info in Hotmail, SkyDrive, Windows Live Messenger, and other services that are compatible with Live Connect; and how to begin working with the signed-in user's info.

- [Referencing the APIs](#)

Begin coding with the Live Connect APIs.

- [Signing users in](#)

How your app can enable a user to sign in.

- [Obtaining user consent](#)

How your app can get a signed-in user's permission, or *consent*, to work with his or her info.

- [Getting user data](#)

How your app can begin working with a signed-in user's info.

- [Single-sign-on and OAuth for apps](#)

Ways to avoid requiring a user to sign in again as he or she switches between multiple apps.

These core concepts build on the info in [Getting started](#). After you learn these concepts, we encourage to begin working with the code examples in areas such as [Identity \(profiles\)](#), [Hotmail \(contacts and calendars\)](#), [SkyDrive \(files and photos\)](#), and [Messenger \(instant messaging\)](#). You can also go straight to the [Live Connect reference](#) to get info to further customize your apps.

Referencing the APIs

[This documentation is preliminary and is subject to change.]

The Live Connect APIs includes implementations for JavaScript, managed languages like C#, and Representational State Transfer (REST). Here's how to reference the APIs.

- [Referencing the API with JavaScript](#)
- [Referencing the API with C#](#)
- [Referencing the API with REST](#)

Referencing the APIs with JavaScript

Instructions for Metro style apps using JavaScript

To reference the APIs in a project in Microsoft Visual Studio 11 for Windows Developer Preview, see the [Metro style apps](#) topic's "Referencing the APIs with JavaScript" section.

Instructions for client-side websites and scripts that use JavaScript

For client-side websites and scripts that use JavaScript, to reference the APIs, set a reference to a web-based version of the wl.js file from your code, like this.

```
<script src="//js.live.net/v5.0/wl.js"></script>
```

A web-based debugging version of wl.js, called wl.debug.js, is also available. It is an uncompressed version of wl.js. To call it, substitute wl.debug.js for wl.js, like this:

```
<script src="//js.live.net/v5.0/wl.debug.js"></script>
```

Web-based localized versions of the APIs are also available for JavaScript. You simply add a URL segment that specifies the culture-name string for the localized version that you want to use. For example, to load the web-based Japanese version, your app uses a URL like this:

```
<script src="//js.live.net/v5.0/ja/wl.js"></script>
```

For a complete list of supported localized versions, see [Supported locales](#).

[Top](#)

Referencing the APIs with C#

For Metro style apps with C# and for Windows Phone apps with C#, to reference the APIs, see the [Metro style apps](#) topic's "Referencing the APIs with C#" section or the [Windows Phone apps](#) topic's "Referencing the APIs with C#" section, respectively.

[Top](#)

Referencing the APIs with REST

Use the info in [Server-side scenarios](#) to learn about available REST calls and their syntax. To learn how to make REST calls for your specific platform and programming language combination, see your documentation.

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Signing users in

[This documentation is preliminary and is subject to change.]

Before you can work with a user's info, the user must be signed in to Live Connect. A standard way to do this is by adding sign-in functionality to your app. This functionality—typically a button or a hyperlink—enables a user to provide their Microsoft account credentials. After the Live Connect APIs verify these credentials, it returns an *access token* to your app. This access token confirms that the user successfully signed in, and it specifies which parts of the user's info your app can work with and for how long.

The APIs support several different ways to help you add sign-in functionality to your app.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Signing in a user with JavaScript](#)
- [Signing in a user with C#](#)
- [Signing in a user with REST](#)
- [Next steps](#)

Signing in a user with JavaScript

The Live Connect APIs includes two methods that you can use to add sign-in functionality to your app with JavaScript: the **WL.ui** method, which you can use to create a sign-in button; and the **WL.login** method, which allows you to add sign-in functionality to any HTML element.

First, you start by initializing the APIs. You can define your own initialization code by using the **WL.Event.subscribe** method to subscribe to the **auth.login** event (which is called during sign-in) and then using **WL.init** to handle the initialization itself. The APIs call **WL.init** when it loads. Here's how to use **WL.init**.

```
WL.Event.subscribe("auth.login", onLoginComplete);

WL.init({
  client_id: CLIENT_ID,
  redirect_uri: REDIRECT_URL,
  response_type: "token"
});
```

Note that Metro style apps using JavaScript don't have to specify the *client_id*, *redirect_uri*, and *response_type* parameters in the call to **WL.init**. Simply call **WL.init()**.

Using WL.ui

After you initialize the APIs, if you want to follow a more controls-based approach to signing users in, we recommend using the **WL.ui** method to create a Live Connect sign-in control. With this control, you define the type of UI element you want to create—in this case, a sign-in button.

To add a sign-in control, you first need to add a container element, such as a **<div>** tag, like this.

```
<div id="signin"></div>
```

Now you can use **WL.ui** method to create the sign-in control. When this code runs, it adds a sign-in button to the **<div>** tag that has an ID of *signin*. After the user clicks the button and successfully signs in, Live Connect asks the user to *okay*, or *consent*, to the **wl.signin** permission level, or *scope*.

```
var scopes = ["wl.signin"];
WL.ui({
  name: "signin",
  element: "signin",
  scope: scopes
});
```

In general, we recommend that you include the sign-in control as part of your initialization functions.

Using WL.login

The APIs support an alternate way to sign in a user with JavaScript: the **WL.login** method. This method is ideal for situations in which the sign-in control does not meet the needs of your app, or in which the user must provide an additional *okay*, or *consent*, for your app to work with additional info for him or her.

To use **WL.login**, assuming that you've already initialized the APIs in the previous section, you provide the desired permission levels, or *scopes*. Here's how a custom *signInUser* function calls **WL.login** to sign in the user with the **wl.signin** scope. (The results of the sign in this case are handled by a custom *onLoginComplete* function, which is provided later in this topic.) The custom *signInUser* function is called when the user clicks the accompanying HTML button control.

```
function signInUser() {
  var scopes = ["wl.signin"];
  WL.login({
    scope: scopes
  }, onLoginComplete);
}

document.write("<button onclick='signInUser()'>" +
  "Sign In</button>");
```

Handling events

Many of the methods in the APIs support an optional callback parameter that your app can use in JavaScript to control what happens after a call finishes. You can also use an event handler to detect changes in user status. To handle events with the APIs, you use the **WL.Event.subscribe** method. For example, here's how to use this method to subscribe to the **auth.login** event, which happens when

the user signs in.

```
WL.Event.subscribe("auth.login", onLoginComplete);
```

Another common event to subscribe to is **auth.sessionChange**. This event happens when some aspect of the sign-in session changes, such as when the user consents to a new scope, like this.

```
WL.Event.subscribe("auth.sessionChange", onSessionChange);
```

To handle a session change event, use the **WL.getSession** method to get a session object for the user. This object contains information such as the user ID, access token, and the scopes the user consented to. You can then use the object to run code to handle the event.

Here's how to handle the **auth.login** and **auth.onSessionChange** events.

```
function onLoginComplete() {
    var session = WL.getSession();
    if (session.error) {
        document.getElementById("infoLabel").innerText =
            "Error signing in: " + session.error;
    }
    else {
        document.getElementById("infoLabel").innerText =
            "Signed in.";
    }
}

function onSessionChange() {
    var session = WL.getSession();
    if (session) {
        document.getElementById("infoLabel").innerText =
            "Something about the session changed.";
    }
    else {
        document.getElementById("infoLabel").innerText =
            "Signed out or session error.";
    }
}
```

[Top](#)

Signing in a user with C#

One of the ways to prompt a user to sign in is to use the Live Connect sign-in control. You do this by setting the sign-in control's properties in Extensible Application Markup Language (XAML) code, for instance in a MainPage.xaml file. At a minimum, you must set the **Scopes** property. For Windows Phone apps, you must also set the **SessionChanged**, **ClientId**, and **RedirectUri** properties. Of the sign-in control's other properties, two are worth mentioning here: if you don't set the **Branding** and **TextType** properties, by default the sign-in button will display the Windows brand logo with the text "Sign in".

Here's what the XAML code might look like for a Metro style app using C# (default page-level attributes are omitted here for brevity).

```
<UserControl x:Class="WindowsTailoredCodeSample.MainPage"
    ...
    xmlns:live="using:Microsoft.Live.Controls">
    <Grid x:Name="LayoutRoot" Background="#FF00C0C0">
        <live:SignInButton x:Name="btnLogin" Scopes="wl.signin wl.basic" />
        <TextBlock Height="32" Foreground="White" HorizontalAlignment="Left" Margin="8,76,0,0" Name="infoTextBlock" VerticalAlignment="Top" Width="150" />
    </Grid>
</UserControl>
```

Here's what the XAML code might look like for a Windows Phone app (default page-level attributes are omitted here for brevity; replace **CLIENT_ID** with your app's client ID and **REDIRECT_URL** with your app's redirect domain).

```
<phone:PhoneApplicationPage
    ...
    xmlns:my="clr-namespace:Microsoft.Live.Controls;assembly=Microsoft.Live.Controls">
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <my:SignInButton Name="btnLogin" ClientId="CLIENT_ID" Scopes="wl.signin wl.basic" RedirectUri="REDIRECT_URL" Branding="Windows" />
        <TextBlock Height="32" HorizontalAlignment="Left" Margin="12,78,0,0" Name="infoTextBlock" Text="" VerticalAlignment="Top" Width="150" />
    </Grid>
</phone:PhoneApplicationPage>
```

After you set these properties, simply call the **InitializeComponent** method, like this.

```
public MainPage()
{
    InitializeComponent();
}
```

Note For Windows Phone apps, you set the **SessionChanged** event handler method in XAML code as shown. For Metro style apps using C#, you set the **SessionChanged** event handler after the call to **InitializeComponent**, like this.

```
...
InitializeComponent();
btnSignin.SessionChanged +=
    new EventHandler<LiveConnectSessionChangedEventArgs>(btnSignin_SessionChanged);
...
```

In the custom method that handles the sign-in control's **SessionChanged** event handler, do this.

```
private void btnSignin_SessionChanged(object sender, LiveConnectSessionChangedEventArgs e)
{
    if (e.Session != null
        && e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
    }
}
```

```

        }
    else
    {
        infoTextBlock.Text = "Not signed in.";
        client = null;
    }
}

```

Here's what the code does.

1. Checks to see if the **LiveConnectSessionChangedEventArgs** instance's (represented here by the variable `e`) session exists and is successfully connected.
2. If so, sets the **LiveConnectSession** instance (represented here by the global variable `session`, which was declared earlier) to the connect client's session, so that it can be reused later in code if needed. Also, initializes an instance of the **LiveConnectClient** class, representing the connected client's session (represented here by the global variable `client`, which was declared earlier), so that it also can be reused later in code if needed.
3. If the session can't be successfully established and connected, display an error message and set an instance of the **LiveConnectClient** class to **null**.

Alternatively, you can prompt a user to sign in without using the Live Connect sign-in control. Or, after an initial sign-in, you can prompt the signed-in user to okay additional scopes that your app needs. One way to do this is to create a button control and then call this code in the button control's click event handler, like this.

```

private void moreScopes_Click(object sender, RoutedEventArgs e)
{
    string[] allScopes = { "wl.signin", "wl.basic" };
    var scopes = new List<string>(allScopes);
    auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
    auth.LoginCompleted +=
        new EventHandler<LoginCompletedEventArgs>(MoreScopes_LoginCompleted);
    auth.LoginAsync(scopes);
}

void MoreScopes_LoginCompleted(object sender, LoginCompletedEventArgs e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}

```

Here's what the code does.

1. Specifies one or more scopes, expressed as an array of strings.
2. Initializes an instance of the **LiveAuthClient** class (represented here by the global variable `auth`, which was declared earlier) with the app's client ID and redirect URL. (For Metro style apps using C#, you can omit providing the client ID and redirect URL. Simply call `auth = new LiveAuthClient();`.)
3. Specifies an event handler method for the **LiveAuthClient** instance's **LoginCompleted** event (in this case, `MoreScopes_LoginCompleted`).
4. Calls the **LiveAuthClient** instance's **LoginAsync** method, specifying the requested scopes.
5. In the event handler method, verifies whether the **LoginCompletedEventArgs** instance's (represented here by the variable `e`) session, representing the connected client's session, is established and connected.
6. If so, sets the **LiveConnectSession** instance (represented here by the global variable `session`, which was declared earlier) to the connect client's session, so that it can be reused later in code if needed. Also, initializes an instance of the **LiveConnectClient** class, representing the connected client's session (represented here by the global variable `client`, which was declared earlier), so that it also can be reused later in code if needed.

[Top](#)

Signing users in with REST

If your app can't use JavaScript or a managed language like C#, we provide a Representational State Transfer (REST) implementation. To use it, your app must first initiate the sign-in process by contacting the Live Connect authorization web service. To do this, make this call from a web browser or a web browser control.

```
GET https://oauth.live.com/authorize?client_id=CLIENT_ID&scope=SCOPES&response_type=RESPONSE_TYPE&redirect_uri=REDIRECT_URL
```

In this code, replace these:

- Replace `CLIENT_ID` with your app's client ID.
- Replace `SCOPES` with the list of scopes to be requested. To specify more than one scope, separate each scope with a space, for example, `wl.signin+wl.basic`. (For more info about scopes, see [Obtaining user consent](#)).
- Replace `RESPONSE_TYPE` with the desired type of response. If you're using the implicit grant flow, to specify an access token to be returned, use `token`. If you're using the authorization code grant flow, to specify an authorization code to be returned, use `code`.

- If you're using the implicit grant flow, replace *REDIRECT_URL* with <http://oauth.live.com/desktop>. If you're using the authorization code grant flow, replace *REDIRECT_URL* with the URL to your callback webpage. If you specified a redirect domain when you created your client ID (like <http://www.contoso.com>), the redirect domain portion of the URL must be the same as the one that you specified when your client ID was created. In any case, the URL must use URL escape codes in this REST call, such as %20 for spaces, %3A for colons, and %2F for forward slashes.

Note For details on the implicit grant flow and the authorization code grant flow, see [OAuth 2.0](#).

When Live Connect receives this call, it checks to see if the user is already signed in. If not, the user is prompted to provide his or her Microsoft account credentials.

After the user successfully signs in and consents to the requested scopes, the Live Connect authorization web service uses the redirect domain to return the user to your app. If you're using the implicit grant flow, you'll see this:

```
https://oauth.live.com/desktop#access_token=ACCESS_TOKEN&token_type=bearer&authentication_token=AUTHENTICATION_TOKEN&expires_in=3600&scopes=SCOPE
```

You use this access token (represented here as *ACCESS_TOKEN*) to work with a user's info. To see how, skip down to the end of this section.

If, however, you're using the authorization code grant flow, different info is returned, and you'll need to make one more call. Here's what's returned.

```
http://www.contoso.com/Callback.htm?code=AUTHORIZATION_CODE
```

If you're using the authorization code grant flow, you use this authorization code (represented here as *AUTHORIZATION_CODE*) to make another call to obtain an access token.

```
POST https://oauth.live.com/token
Content-Type: application/x-www.form-urlencoded
client_id=CLIENT_ID&redirect_uri=REDIRECT_URL&client_secret=CLIENT_SECRET&code=AUTHORIZATION_CODE&grant_type=authorization_code
```

In the body, replace these:

- Replace *CLIENT_ID* with your app's client ID.
- Replace *REDIRECT_URL* with the URL to your callback webpage. This URL must be the same as the URL that you specified when you requested an authorization code. The URL must use URL escape codes, such as %20 for spaces, %3A for colons, and %2F for forward slashes.
- Replace *CLIENT_SECRET* with the client secret that you received when your client ID was created.
- Replace *AUTHORIZATION_CODE* with the authorization code that was returned on the initial call.

If this second call is successful, the response from the Live Connect authorization web service contains a JavaScript Object Notation (JSON) formatted object that includes the access token. The access token is contained in a JSON-formatted object that looks like this, where *ACCESS_TOKEN* represents the actual access token, and *SCOPES* represents the actual scopes:

```
{
  "access_token": "ACCESS_TOKEN",
  "expires_in": 3600,
  "scope": "SCOPES",
  "token_type": "bearer"
}
```

At this point, you can use REST calls to work with the signed-in user's info. For each request, you must append an *access_token* parameter that contains the access token. Note that the access token is valid for only the number of seconds that are specified in the *expires_in* value. Here, *REST_PATH* represents the actual target REST path (like `me`), and *ACCESS_TOKEN* represents the actual access token:

```
https://apis.live.net/v5.0/REST_PATH?access_token=ACCESS_TOKEN
```

For info about using this implementation in a server-side website or script, see [Server-side scenarios](#).

[Top](#)

Next steps

Now that you understand how to sign users in by using the Live Connect APIs, the next steps are to understand user consent and how to get user info. For details about what info is available to your application, see [Scopes and permissions](#). To learn more about consent and how to work with user info, see [Obtaining user consent](#) and [Getting user data](#).

[Top](#)

Obtaining user consent

[This documentation is preliminary and is subject to change.]

Users can store a great deal of info in Hotmail, SkyDrive, Windows Live Messenger, and other services that are compatible with Live Connect. Most of this info is considered the users' private property. For your app to access this info, it must first get the signed-in user's okay for permission, or *consent*. The Live Connect APIs divide these okays for permission into several levels or categories, called *scopes*.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Common scopes](#)
- [Requesting initial scopes](#)
- [User experience](#)
- [Requesting additional scopes](#)
- [Getting a list of consented-to permissions](#)
- [Next steps](#)

Common scopes

Some common scopes that you might want to request are:

- **wl.basic**—Allows access to a user's basic info, such as his or her Hotmail contact list.
- **wl.emails**—Allows you to access a user's email addresses.
- **wl.photos**—Allows you to access a SkyDrive user's photos.

For a complete list of scopes, see [Scopes and permissions](#).

[Top](#)

Requesting initial scopes

In most cases, you request a user's consent to access his or her info by adding one or more scopes to the sign-in functionality on your site. For example, if you're using JavaScript together with the Live Connect sign-in control, you add a **scope** property, like this.

```
var scopes = ["wl.signin"];
WL.ui({
    name: "signin",
    element: "signin",
    scope: scopes
});
```

If you're using C# together with the Live Connect sign-in control, you set the initial scopes in XAML code, and then you call the **InitializeComponent** method. To see

how to do this, see the [Signing users in](#) topic's "Signing in a user with C#" section.

For Representational State Transfer (REST), you specify scopes in the initial call to the Live Connect authorization web service, where *CLIENT_ID* is the actual client ID, and *REDIRECT_URI* is either a custom redirect URL or <http://oauth.live.com/desktop>, depending on your target OAuth grant flow. If you do not specify *REDIRECT_URI*, the value that was set in the redirect domain is used. For example, if in the app configuration the redirect domain is set to <http://www.contoso.com>, then if you do not specify *REDIRECT_URI*, consent redirects to http://www.contoso.com#access_token=ACCESS_TOKEN.

```
https://oauth.live-int.com/authorize?client_id=CLIENT_ID&scope=wl.signin&response_type=RESPONSE_TYPE&redirect_uri=REDIRECT_URL
```

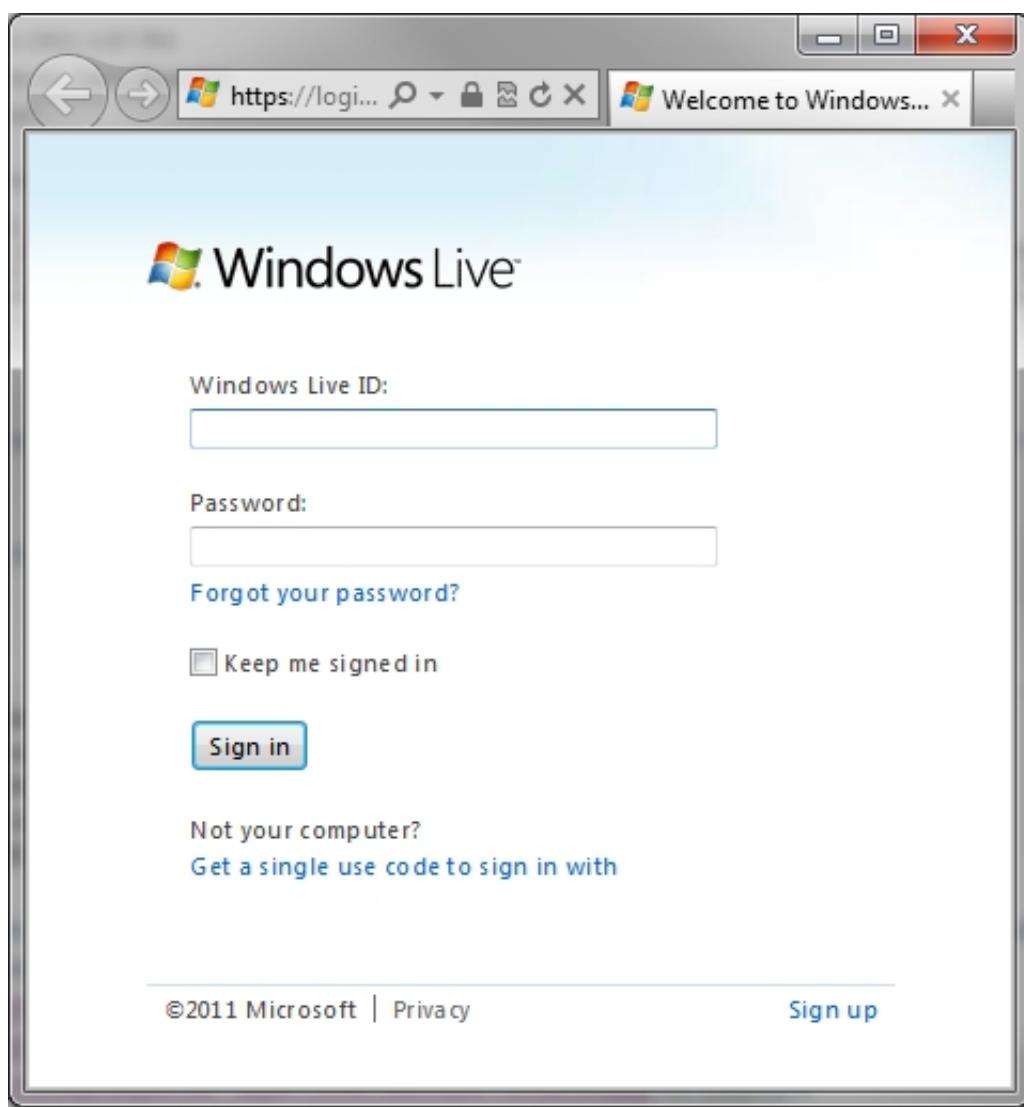
You can request multiple scopes at a time. Each scope must be separated by a space, for example "wl.signin wl.basic". (For REST, separate each scope with the escape character %20.)

To learn more about scopes, see [Scopes and permissions](#), which explains all the different scopes that you can use with your app. We recommend that you limit the number of scopes you request at any given time to the smallest number possible. This helps your users better understand what info they are sharing with your app.

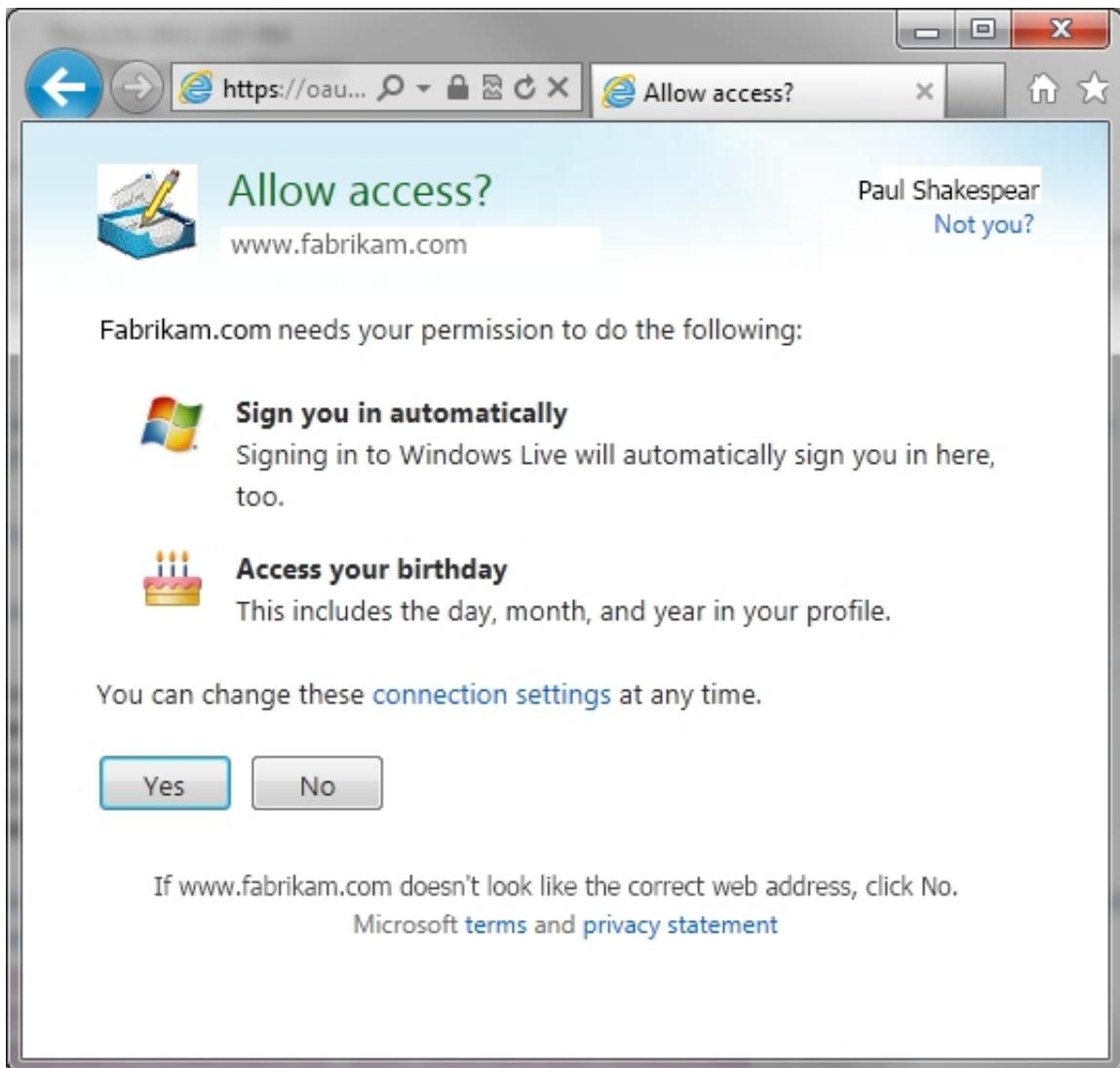
[Top](#)

User experience

The Live Connect APIs use OAuth 2.0 to sign in users and enable them to provide consent to your app. When a user signs in to your app, he or she is redirected to a window that is hosted by the Live Connect authorization web service. (This window may look a bit different depending on your app type.)



In this window, a user types his or her Microsoft account credentials. If the user successfully signs in, the user is then directed to the *consent dialog*, which explains to them what info your app wants to work with. (The consent dialog may look a bit different depending on your app type.)



If the user clicks **Yes**, the **auth.login** event (for JavaScript) or the **SessionChanged** or **LoginCompleted** event (for C#) is raised. Your app subscribes to this event so that when it fires, the app can obtain the session object from the event handler method specified by the **WL.Event.subscribe** method (for JavaScript) or the **SessionChanged** or **LoginCompleted** event handler method (in C#). Your app can then begin working with the user's information.

Once a user provides consent, Live Connect gives your app a special code, or *access token*, that lets your app work with that portion of the user's info that he or she consented to. This access token is good for typically about one hour only. After this hour is up, your app won't be able to work with the user's info anymore—it must ask the user to go through the sign-in and consent process once again. To get around this, you can ask the user to consent to the **wl.offline** scope. This gives your app an additional code, called a *refresh token*, that your app can use to get a new access token whenever it needs to—even after the user signs out—typically for up to a year. However, the user can revoke your app's access at any time. If a user chooses to revoke consent to your app, no corresponding access tokens nor refresh tokens will work—your app must ask the user to go through the sign-in and consent process once again.

Tip You can specify settings for the consent dialog, such as an app logo and links to terms-of-service and privacy pages for your app. For more info about these settings, see [Getting a client ID and configuring your app](#).

[Top](#)

Requesting additional scopes

If a user chooses to use a feature of your app that requires scopes that are different from the ones that the user originally okayed, you must request the user's further consent, like this.

[JavaScript]

```
function moreScopes_onClick() {
    var moreScopes = ["wl.signin wl.basic"];
    WL.login({
        scope: moreScopes
    }, onLoginComplete);
}

function onLoginComplete() {
    var session = WL.getSession();
    if (session.error) {
        document.getElementById("infoLabel").innerText =
            "Error signing in: " + session.error;
    }
    else {
        document.getElementById("infoLabel").innerText =
            "Signed in.";
    }
}
```

[C#]

```
private void moreScopes_Click(object sender, RoutedEventArgs e)
{
    string[] allScopes = { "wl.signin", "wl.basic" };
    var scopes = new List<string>(allScopes);
    auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
    auth.LoginCompleted +=
        new EventHandler<LoginCompletedEventArgs>(MoreScopes_LoginCompleted);
    auth.LoginAsync(scopes);
}

void MoreScopes_LoginCompleted(object sender, LoginCompletedEventArgs e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}
```

```
}
```

Note For Metro style apps using C#, you can omit providing the client ID and redirect URL here and elsewhere. Simply call `auth = new LiveAuthClient();`.

[REST]

For REST, you must get a new access token that covers the additional scopes in addition to any previously-consented-to scopes. You do this by specifying the list of scopes in a new call to the Live Connect authorization web service, like this:

```
https://oauth.live-int.com/authorize?client_id=CLIENT_ID&scope=w1.signin%20w1.basic&response_type=RESPONSE_TYPE&redirect_uri=REDIRECT_URL
```

[Top](#)

Getting a list of consented-to permissions

You can get a list of permissions that the signed-in user has already okayed for your app, like this.

[JavaScript]

```
function checkPermissions_onClick() {
    var scopes = ["wl.signin wl.basic"];
    WL.login({
        scope: scopes
    }, onCheckPermissionsLoginComplete);
}

function onCheckPermissionsLoginComplete() {
    var session = WL.getSession();
    if (session.error) {
        document.getElementById("infoLabel").innerText =
            "Error signing in: " + session.error;
    }
    else {
        WL.api({
            path: "me/permissions",
            method: "GET"
        }, onCheckPermissionsAPICalled);
    }
}

function onCheckPermissionsAPICalled() {
    if (result.error) {
        document.getElementById("infoLabel").innerText =
            "Error calling API: " + result.error.message;
    }
    else {
        var message = "";
        for (property in response.data[0]) {
            if (response.data[0].hasOwnProperty(property))
                message += "<br />" + property;
        }
        document.getElementById("infoLabel").innerHTML = result;
    }
}
```

[C#]

```
private void checkPermissions_Click(object sender, RoutedEventArgs e)
```

```

    {
        string[] allScopes = { "wl.signin", "wl.basic" };
        var scopes = new List<string>(allScopes);
        auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
        auth.LoginCompleted +=
            new EventHandler<LoginCompletedEventArgs>(CheckPermissions_LoginCompleted);
        auth.LoginAsync(scopes);
    }

void CheckPermissions_LoginCompleted(object sender, LoginCompletedEventArgs e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(e.Session);
        infoTextBlock.Text = "Signed in.";
        client.GetCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(CheckPermissions_GetCompleted);
        client.GetAsync("me/permissions");
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}

void CheckPermissions_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        infoTextBlock.Text = e.RawResult;
    }
    else
    {
        infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
    }
}

```

[REST]

GET https://apis.live.net/v5.0/me/permissions?access_token=ACCESS_TOKEN

The returned list of permissions will look similar to this. A "1" here represents a scope that the user has already consented to. (If a scope is not listed, the user is currently not consenting to it for the app.)

```
{
    "data": [
        {
            "wl.basic": 1,
            "wl.signin": 1
        }
    ]
}
```

[Top](#)

Next steps

At this point, you have learned how to sign in users by using the Live Connect APIs, and how scopes and the consent process work. Next, see [Getting user data](#) to learn how to work with the info that the APIs return. From there, you can start looking at other scenarios that demonstrate the capabilities of the APIs, like the ones in [Getting user data](#) and [Identity \(profiles\)](#).

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Getting user data

[This documentation is preliminary and is subject to change.]

To access a user's info from Hotmail, SkyDrive, Windows Live Messenger, and other services that are compatible with Live Connect, your app must first check for two things. First, it must confirm that the user has successfully signed in. Second, it must request and receive the user's okay for permission, or *consent*, to work with his or her info. For details, see [Signing users in](#) and [Obtaining user consent](#). The pattern for accessing user data consists of making a request and then handling the results of that request.

This topic assumes that your app has already successfully initialized the APIs, the user has successfully signed in, and he or she has consented to the **wl.basic** scope (to get basic info about the user and his or her contacts). For details about this, see [Signing users in](#) and [Obtaining user consent](#).

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Requesting info](#)
- [Working with info](#)
- [Next steps](#)

Requesting info

To work with a user's info, you have two options. The first uses the user ID as part of a call, like this, where the user ID is represented as *USER_ID*.

[JavaScript]

```
function readContact() {
    WL.api({
        path: "USER_ID",
        method: "GET"
    }, onReadContactCallback);
}

function onReadContactCallback(response) {
    if (!response.error) {
        var name = response.first_name;
        // Display user's first name.
    }
}
```

[C#]

```
private void readContact(object sender, RoutedEventArgs e)
{
    LiveConnectClient readContact = new LiveConnectClient(session);
    readContact.GetCompleted += ReadContactProperties_GetCompleted;
    readContact.GetAsync("USER_ID");
}

void ReadContactProperties_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string name = e.Result["first_name"].ToString();
        // Display user's first name.
    }
}
```

```
    }  
}
```

[REST]

```
GET http://apis.live.net/v5.0/USER_ID?access_token=ACCESS_TOKEN
```

The second makes use of a Live Connect APIs shortcut called **me**. You can use it to get info that relates to the user who is currently signed in, like this:

[JavaScript]

```
...  
WL.api({  
    path: "me",  
    method: "GET"  
}, onApiCalled);  
...
```

[C#]

```
...  
readContact.GetAsync("me");  
...
```

[REST]

```
GET http://apis.live.net/v5.0/me?access_token=ACCESS_TOKEN
```

Using **me** is often easier and simpler than supplying the user ID, because you do not have to verify that the user ID that you are using is the signed-in user's user ID.

Here's how to use **me** to get a list of the signed-in user's friends.

[JavaScript]

```
...  
WL.api({  
    path: "me/friends",  
    method: "GET"  
}, onApiCalled);  
...
```

[C#]

```
...  
readContact.GetAsync("me/friends");  
...
```

[REST]

```
GET http://apis.live.net/v5.0/me/friends?access_token=ACCESS_TOKEN
```

Assuming that your app has permission to access data that relates to the user's friends, you can get that info by using the user ID of a particular friend, like this.

[JavaScript]

```
...  
WL.api({  
    path: "cc988147fd2a1111",  
    method: "GET"  
}, onApiCalled);  
...
```

[C#]

```
...
readContact.GetAsync("cc988147fd2a1111");
...
```

[REST]

```
GET http://apis.live.net/v5.0/cc988147fd2a1111?access_token=ACCESS_TOKEN
```

To make additional calls to work with info, in JavaScript use [WL.api](#). In C#, use the **LiveConnectClient** class's methods like **GetAsync**, **PostAsync**, **PutAsync**, and **DeleteAsync** for GET, POST, PUT, and DELETE calls, respectively. For example, here's how to make a GET call to get the first name of a user's friend.

[JavaScript]

```
function getFriendName_onClick() {
    WL.api({
        path: "cc988147fd2a1111",
        method: "GET"
    }, function (response) {
        if (!response.error) {
            alert("Friend's name is: " + response.first_name);
        }
    });
}
```

[C#]

```
private void getFriendName_Click(object sender, RoutedEventArgs e)
{
    if (client == null)
    {
        MessageBox.Show("Must sign in first.");
    }
    else
    {
        client.GetCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(getFriendName_GetCompleted);
        client.GetAsync("cc988147fd2a1111");
    }
}

void getFriendName_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Friend's name is: " + e.Result["first_name"]);
    }
    else
    {
        infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
    }
}
```

[REST]

In REST, once you make the appropriate GET call, the user's friend's first name is inside of the response. Here's how to make a call using the friend's ID.

```
GET https://apis.live.net/v5.0/cc988147fd2a1111?access_token=ACCESS_TOKEN
```

[Top](#)

Working with info

The APIs return the response as info formatted in JavaScript Object Notation (JSON). Here's a JSON-formatted object that contains a user's publicly available info.

```
{  
  "id" : "cc988147fd2a1111",  
  "first_name" : "Roberto",  
  "last_name" : "Tamburello",  
  "name" : "Roberto Tamburello",  
  "gender" : "male",  
  "locale" : "en_US"  
}
```

The [REST API](#) topic provides info about the JSON-formatted object that is returned in response to each type of request.

[Top](#)

Next steps

Now you know how to get info about a user by using the Live Connect APIs. From here, for details about how to get specific sets of user info, see [Identity \(profiles\)](#), [Hotmail \(contacts and calendars\)](#), and [SkyDrive \(files and photos\)](#).

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Identity (profiles)

[This documentation is preliminary and is subject to change.]

The more info you have about a user, the better you can customize the user's experience. By identifying the user and personalizing his or her experience, you can influence users to return to your app regularly.

Here you can learn how to use the Live Connect APIs to identify users and to use what you know about them to create a personalized app experience.

- [Single-sign-on and OAuth for apps](#)

Provide a single-sign-on experience for your users.

- [Greeting the user](#)

Learn how to get a user's basic info, and use that info to welcome the user by displaying his or her first name and profile picture.

- [Streamlining account registration](#)

Improve any account-registration process for your app by making use of info that a user already has in Live Connect.

- [Contacting users](#)

Access a user's phone number, email address, or postal address to do things like contact the user about an order's status or to share new info about your app with him or her.

- [Determining a user's age](#)

Get a user's date of birth and use that info to change the app experience depending on his or her age.

© 2011 Microsoft Corporation. All rights reserved.

Single-sign-on and OAuth for apps

[This documentation is preliminary and is subject to change.]

As websites and apps become more personalized, the need to simplify the authentication process is more important than ever. You can set up your app or website to provide a single-sign-on experience for your users, so that if a user is already signed in to Live Connect, they will not need to sign in again when they use your app or website.

Because Windows Developer Preview enables users to sign in to their device by using a Microsoft account, application developers can provide a single-sign-on experience across websites and Metro style apps. For example, a website can integrate the Live Connect APIs so that once a user has signed in, they can also access services that are compatible with Live Connect like Hotmail and SkyDrive. Or, if a user is signed in to their Windows Developer Preview computer, they will already be signed in to the app's website.

Overview

The key to enabling single-sign-on is the authentication token, which contains an app-specific user ID and is encoded as a JavaScript Object Notation (JSON) Web Token (JWT). The basic flow for working with the authentication token is described here.

- [Get the authentication token](#)
- Decode the token
- Save the data from each segment of the token
- Use the user ID value for comparison when a user signs in

Get the authentication token

You may configure an application to obtain an authentication token in one of the following ways. Note that the way you choose will be constrained by the type of application you are working on.

- It is one of the values returned after the hash fragment as part of the response to a successful Representational State Transfer (REST) authorization request. For example:

```
http://www.contoso.com/callback.htm#access_token=ACCESS_TOKEN&authentication_token=AUTHENTICATION_TOKEN
```

- In JavaScript, use the **authentication_token** property of the **session** object.
- In C#, use the **AuthenticationToken** property of the **LiveConnectSession** object.
- In Windows Developer Preview, you can use the appropriate classes in the **Windows.Security.Authentication** namespace.

Authentication token format

The authentication token data is encoded as a JSON Web Token (JWT), which is constructed by concatenating an encoded version of the JSON payload with a signed and encoded version of the JSON payload. For more information, see [JSON Web Token \(JWT\)](#). The authentication token can be decoded and verified by using code similar to the example in the next section. Once decoded, the application-specific user ID can then be stored and used to identify the user the next time they sign in with a valid authentication token.

Authentication token data

The authentication token is broken up into two sets of data: the header which contains metadata about the token, and the body of the token which contains the actual token data. The following table describes the properties of the authentication token.

Property	Name	Info
Algorithm	alg	The cryptographic algorithm used to sign the token. Only HMAC SHA-256 is supported.
Type	typ	The type of token. Only JWT (JSON Web Token) is supported.
KID	kid	Key version field of the application. It can be found in the application settings tab on the application management page.
Version	ver	Version identifier for the token.
Issuer	iss	Identifies the principal who issued the token.
Expiration	exp	Identifies the expiration time on or after which the token must not be accepted for processing. The unit is given in seconds since midnight of 1/1/1970 in coordinated universal time (UTC).
Audience	aud	Identifies the JWT audience that the JWT is intended for. In this case, it is the domain name for the application.
User Identifier	uid	An identifier for the user which is unique to the application.
Package	urn:microsoft:appurl	The Windows client identifier of the

SID

application, if there is one.

The following example shows an authentication token and the corresponding JSON objects that can be obtained by decoding it.

```
eyJhbGciOiJIUzI1NiI...eyJ2ZXIiOjEsICJpc3MiOiJ1cm46d2luZG93czpsaXZlaWQilCAiZXhwIjozMzA2Mjk2ODY2LCAiYXVkJoi...tIiwgInVpZCI6IjcxNmEyZGY5OWE2MWQ4NzlhMGQ2ZmMyNWM4Yz...HY2thZ2VzaWQoIiifQ.gg5I38qJILse6-ELZ3p3cu4qJotcbx6aJ-Cs8T...xvM5s
```

```
Header: {"alg": "HS256", "typ": "JWT", "kid": "0"}  
Token: {"ver": 1, "iss": "urn:windows:liveid", "exp": 1306296866, "aud": "oauth.unit...test.com", "uid": "716a2df99a61d879a0d6fc25c8c3733f", "urn:microsoft:appurl": ""}
```

Example helper class code

The following C# code demonstrates decoding the authentication token into the corresponding JSON strings, and also shows how to use the application's client secret to verify that the token has not been tampered with.

```
namespace Build.Samples.Authentication  
{  
    // Reference: http://tools.ietf.org/search/draft-jones-json-web-token-00  
    //  
    // JWT is made up of 3 parts: Envelope, Claims, Signature.  
    // - Envelope - specifies the token type and signature algorithm used to produce  
    //               signature segment. This is in JSON format.  
    // - Claims - specifies claims made by the token. This is in JSON format.  
    // - Signature - Cryptographic signature use to maintain data integrity.  
    //  
    // To produce a JWT token:  
    // 1. Create Envelope segment in JSON format.  
    // 2. Create Claims segment in JSON format.  
    // 3. Create signature.  
    // 4. Base64url encode each part and append together separated by ".."  
  
    using System;  
    using System.Collections.Generic;  
    using System.Security.Cryptography;  
    using System.Text.RegularExpressions;  
    using System.Runtime.Serialization;  
    using System.Runtime.Serialization.Json;  
    using System.Text;  
    using System.IO;  
  
    public class JsonWebToken  
    {  
        #region Helper Classes  
        [DataContract]  
        public class JsonWebTokenClaims  
        {  
            [DataMember(Name = "exp")]  
            private int expUnixTime  
            {  
                get;  
                set;  
            }  
        }  
    }
```

```

private DateTime? expiration = null;
public DateTime Expiration
{
    get
    {
        if (this.expiration == null)
        {
            this.expiration = new DateTime(1970, 1, 1, 0, 0, 0).AddSeconds(expUnixTime);
        }

        return (DateTime)this.expiration;
    }
}

[DataMember(Name = "iss")]
public string Issuer
{
    get;
    private set;
}

[DataMember(Name = "aud")]
public string Audience
{
    get;
    private set;
}

[DataMember(Name = "uid")]
public string UserId
{
    get;
    private set;
}

[DataMember(Name = "ver")]
public int Version
{
    get;
    private set;
}

[DataMember(Name = "urn:microsoft:appurl")]
public string ClientIdentifier
{
    get;
    private set;
}

[DataContract]
public class JsonWebTokenEnvelope
{
    [DataMember(Name = "typ")]
    public string Type
    {
        get;
        private set;
    }

    [DataMember(Name = "alg")]
}

```

```

        public string Algorithm
        {
            get;
            private set;
        }

        [DataMember(Name = "kid")]
        public int KeyId
        {
            get;
            private set;
        }
    }
#endregion

#region Properties

    private static readonly DataContractJsonSerializer ClaimsJsonSerializer
= new DataContractJsonSerializer(typeof(JsonWebTokenClaims));
    private static readonly DataContractJsonSerializer EnvelopeJsonSerializer
= new DataContractJsonSerializer(typeof(JsonWebTokenEnvelope));
    private static readonly UTF8Encoding UTF8Encoder = new UTF8Encoding(true,
, true);
    private static readonly SHA256Managed SHA256Provider = new SHA256Managed
();

    private string claimsTokenSegment;
    public JsonWebTokenClaims Claims
    {
        get;
        private set;
    }

    private string envelopeTokenSegment;
    public JsonWebTokenEnvelope Envelope
    {
        get;
        private set;
    }

    public string Signature
    {
        get;
        private set;
    }

    public bool IsExpired
    {
        get
        {
            return this.Claims.Expiration < DateTime.Now;
        }
    }
}

#endregion

#region Constructors
public JsonWebToken(string token, Dictionary<int, string> keyIdsKeys)
{
    // Get the token segments and perform validation.
    string[] tokenSegments = this.SplitToken(token);

    // Decode and deserialize the claims.
}

```

```

        this.claimsTokenSegment = tokenSegments[1];
        this.Claims = this.GetClaimsFromTokenSegment(this.claimsTokenSegment
    );

        // Decode and deserialize the envelope.
        this.envelopeTokenSegment = tokenSegments[0];
        this.Envelope = this.GetEnvelopeFromTokenSegment(this.envelopeTokenS
egment);

        // Get the signature.
        this.Signature = tokenSegments[2];

        // Ensure that the tokens KeyId exists in the secret keys list.
        if (!keyIdsKeys.ContainsKey(this.Envelope.KeyId))
        {
            throw new Exception(string.Format("Could not find key with id {0
}.", this.Envelope.KeyId));
        }

        // Validation
        this.ValidateEnvelope(this.Envelope);
        this.ValidateSignature(keyIdsKeys[this.Envelope.KeyId]);
    }

    private JsonWebToken()
    {
    }
}

#endregion

#region Parsing Methods

    private JsonWebTokenClaims GetClaimsFromTokenSegment(string claimsTokenS
egment)
    {
        byte[] claimsData = this.Base64UrlDecode(claimsTokenSegment);
        using (MemoryStream memoryStream = new MemoryStream(claimsData))
        {
            return ClaimsJsonSerializer.ReadObject(memoryStream) as JsonWebT
okenClaims;
        }
    }

    private JsonWebTokenEnvelope GetEnvelopeFromTokenSegment(string envelope
TokenSegment)
    {
        byte[] envelopeData = this.Base64UrlDecode(envelopeTokenSegment);
        using (MemoryStream memoryStream = new MemoryStream(envelopeData))
        {
            return EnvelopeJsonSerializer.ReadObject(memoryStream) as JsonWe
bTokenEnvelope;
        }
    }

    private string[] SplitToken(string token)
    {
        // Expected token format: Envelope.Claims.Signature

        if (string.IsNullOrEmpty(token))
        {
            throw new Exception("Token is empty or null.");
        }

        string[] segments = token.Split('.');
    }
}

```

```

        if (segments.Length != 3)
        {
            throw new Exception("Invalid token format. Expected Envelope.Claims.Signature.");
        }

        if (string.IsNullOrEmpty(segments[0]))
        {
            throw new Exception("Invalid token format. Envelope must not be empty.");
        }

        if (string.IsNullOrEmpty(segments[1]))
        {
            throw new Exception("Invalid token format. Claims must not be empty.");
        }

        if (string.IsNullOrEmpty(segments[2]))
        {
            throw new Exception("Invalid token format. Signature must not be empty.");
        }

        return segments;
    }

#endregion

#region Validation Methods

private void ValidateEnvelope(JsonWebTokenEnvelope envelope)
{
    if (envelope.Type != "JWT")
    {
        throw new Exception("Unsupported token type.");
    }

    if (envelope.Algorithm != "HS256")
    {
        throw new Exception("Unsupported crypto algorithm.");
    }
}

private void ValidateSignature(string key)
{
    // Derive signing key, Signing key = SHA256(secret + "JWTsig")
    byte[] bytes = UTF8Encoder.GetBytes(key + "JWTsig");
    byte[] signingKey = SHA256Provider.ComputeHash(bytes);

    // To Validate:
    //
    // 1. Take the bytes of the UTF-8 representation of the JWT Claim Segment and calculate an HMAC SHA-256 MAC on them using the shared key.
    //
    // 2. Base64url encode the previously generated HMAC as defined in this document.
    //
    // 3. If the JWT Crypto Segment and the previously calculated value exactly match in a character by character, case sensitive
}

```

```

//      comparison, then one has confirmation that the key was used to
//      generate the HMAC on the JWT and that the contents of the JWT
//      Claim Segment have not been tampered with.
//
// 4. If the validation fails, the token must be rejected.

// UTF-8 representation of the JWT envelope.claim segment.
byte[] input = UTF8Encoder.GetBytes(this.envelopeTokenSegment + "."
+ this.claimsTokenSegment);

// Calculate an HMAC SHA-256 MAC.
using (HMACSHA256 hashProvider = new HMACSHA256(signingKey))
{
    byte[] myHashValue = hashProvider.ComputeHash(input);

    // Base64 url encode the hash.
    string base64urlEncodedHash = this.Base64UrlEncode(myHashValue);

    // Now compare the two hash values.
    if (base64urlEncodedHash != this.Signature)
    {
        throw new Exception("Signature does not match.");
    }
}

#endregion

#region Base64 Encode / Decode Functions
// Reference: http://tools.ietf.org/search/draft-jones-json-web-token-00

public byte[] Base64UrlDecode(string encodedSegment)
{
    string s = encodedSegment;
    s = s.Replace('-', '+'); // 62nd char of encoding.
    s = s.Replace('_', '/'); // 63rd char of encoding.
    switch (s.Length % 4) // Pad with trailing '='s.
    {
        case 0: break; // No pad chars in this case.
        case 2: s += "=="; break; // Two pad chars.
        case 3: s += "="; break; // One pad char.
        default: throw new System.Exception("Illegal base64url string");
    }
    return Convert.FromBase64String(s); // Standard base64 decoder.
}

public string Base64UrlEncode(byte[] arg)
{
    string s = Convert.ToBase64String(arg); // Standard base64 encoder.
    s = s.Split('=')[0]; // Remove any trailing '='s.
    s = s.Replace('+', '-'); // 62nd char of encoding.
    s = s.Replace('/', '_'); // 63rd char of encoding.
    return s;
}

#endregion
}

```


Greeting the user

[This documentation is preliminary and is subject to change.]

Greeting a user by name when he or she visits your app is a simple but effective way of building a stronger connection between the user and your app. This connection can help you influence users to return to your app. This topic describes and demonstrates how to use the Live Connect APIs to greet a user by name and how to display the user's profile picture.

This topic assumes that your app has already successfully initialized the APIs, the user has successfully signed in, and he or she has consented to the **wl.basic** scope (to get the user's name). For details about this, see [Signing users in](#) and [Obtaining user consent](#).

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Getting the user's name and profile image](#)
- [Next steps](#)

Getting the user's name and profile picture

Here's how to display a greeting message containing the user's name.

[JavaScript]

```
function greetUser_onClick() {
    WL.api({
        path: "me",
        method: "GET"
    }, function (response) {
        if (!response.error) {
            document.getElementById("infoArea").innerText =
                "Hello, " + response.first_name + " " +
                response.last_name + "!";
        }
    });
}
```

The code calls the **WL.api** function. Assuming the call is successful, and assuming a **textarea** control exists with the name `infoArea`, the code changes the control's text to display a message with the signed-in user's first and last name. The code gets this from `response.first_name` and `response.last_name`.

[C#]

```
private void greetUser_Click(object sender, RoutedEventArgs e)
{
    if (client == null)
    {
        MessageBox.Show("Must sign in first.");
    }
    else
    {
        client.GetCompleted +=
```

```

        new EventHandler<LiveOperationCompletedEventArgs>(GreetUser_GetCompl
eted);
        client.GetAsync("me");
    }
}

void GreetUser_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Hello, " + e.Result["first_name"] + " " +
            e.Result["last_name"]);
    }
    else
    {
        this.infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
    }
}

```

In the custom `greetUser_Click` event handler method, the code calls the **LiveConnectClient** instance's **GetAsync** method. In the custom `greetUser_GetCompleted` event handler method, assuming the call is successful, the code displays a message containing the signed-in user's first name and last name. The code gets the signed-in user's first name and last name from the **LiveOperationCompletedEventArgs** instance's **Result** property. The **Result** property returns a **IDictionary<string, object>** object. This object is a representation of the underlying JavaScript Object Notation (JSON)-formatted info.

Tip For Metro style apps using C#, you can shorten calls like `e.Result["first_name"]` to something easier like `meResult.first_name`, and write less code overall, by using the **dynamic** keyword. For details, see the [Metro style apps](#) topic's "Code samples" section in the `MainPage.xaml.cs` code example.

[REST]

In Representational State Transfer (REST), once you make the appropriate GET call, the user's first and last names are inside of the response. Here's how to make a GET call where `me` represents the signed-in user.

```
GET https://apis.live.net/v5.0/me?access_token=ACCESS_TOKEN
```

Here's code that shows how to get a user's profile picture.

[JavaScript]

```

function getImage_onClick() {
    var scopes = ["wl.basic"];
    WL.login({
        scope: scopes
    }, function (response) {
        if (!response.error) {
            document.getElementById("userImage").setAttribute("src",
                "https://apis.live.net/v5.0/me/picture?access_token=" +
                escape(response.session.access_token));
        }
    });
}

```

The code uses the **WL.login** function to attempt to sign in the user with the **wl.basic** scope. If the sign-in succeeds, and assuming an **img** control exists with

the name `userImage`, the code sets the `userImage` control's `src` attribute to an appropriately-constructed GET call to the signed-in user's profile picture. The code gets the needed access token from `response.session.access_token`.

[C#]

```
private void getUserImage_Click(object sender, RoutedEventArgs e)
{
    string[] allScopes = { "wl.basic" };
    var scopes = new List<string>(allScopes);
    auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
    auth.LoginCompleted +=
        new EventHandler<LoginCompletedEventArgs>(GetUserImage_LoginCompleted);
    auth.LoginAsync(scopes);
}

void GetUserImage_LoginCompleted(object sender, LoginCompletedEventArgs e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
        var memoryStream = new System.IO.MemoryStream();
        client.DownloadCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(GetProfilePicture_
DownloadCompleted);
        client.DownloadAsync("me/picture", memoryStream, memoryStream);
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}

void GetProfilePicture_DownloadCompleted(object sender, LiveOperationCompletedEv
entArgs e)
{
    if (e.Error == null)
    {
        var memoryStream = e.UserState as System.IO.MemoryStream;
        if (memoryStream != null)
        {
            imageFrame.Visibility = Visibility.Visible;
            BitmapImage imgSource = new BitmapImage();
            imgSource.SetSource(memoryStream);
            imageFrame.Source = imgSource;
            memoryStream.Close();
        }
    }
    else
    {
        infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
    }
}
```

The code does this.

1. In the custom `getUserImage_Click` event handler method, uses the **LoginAsync** method to attempt to sign in the user with the **wl.basic** scope.
2. In the custom `GetUserImage_LoginCompleted` event handler method, if the sign-in succeeds, the code calls the **LiveConnectClient** instance's **DownloadAsync** method to get the signed-in user's profile picture.
3. In the custom `GetProfilePicture_DownloadCompleted` event handler method, if the code successfully gets the profile picture, and assuming a **Picture** control exists named `imageFrame`, the `imageFrame` Control's **Source** property is set to the profile picture.

[REST]

In REST, you can get the signed-in user's profile picture by making an appropriately-constructed GET call. Here's how, where `me` represents the signed-in user.

```
GET https://apis.live.net/v5.0/me/picture?access_token=ACCESS_TOKEN
```

[Top](#)

Next steps

You now have an app that can greet a user by name and display his or her profile picture. From here, you can learn how to work with additional user info by going to [Streamlining account registration](#), [Contacting users](#), and [Determining a user's age](#).

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Streamlining account registration

[This documentation is preliminary and is subject to change.]

Most users dislike entering their personal info in multiple apps, especially when it comes to creating an app-specific account. If your app needs to create an account for a user, you can use the Live Connect APIs to streamline your account registration process. You can do this by using info that the user may have already stored with Live Connect, like his or her email address. This can make creating an app-specific account as simple for the user as clicking a link and typing his or her Microsoft account credentials.

This topic describes how to get a user's okay for permission, or *consent*, for your app to access his or her info—specifically, the user's first name, last name, email address, gender, and birthday.

This topic assumes that your app has already successfully initialized the APIs, the user has successfully signed in, and he or she has consented to the scopes **wl.basic** (to get the user's name, and gender if it exists), **wl.emails** (to get the user's email address, if it exists), and **wl.birthday** (to get the user's birthday info, if it exists). For details about this, see [Signing users in](#) and [Obtaining user consent](#).

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Getting and working with user info](#)
- [Next steps](#)

Getting and working with user info

To get info about a user, use the [WL.api](#) method in JavaScript or the [LiveConnectClient.GetAsync](#) method in C#. Here's how.

[JavaScript]

```
function onLoginAccountReg() {
    WL.api({
        path: "me",
        method: "GET"
    }, function (response) {
        if (!response.error) {
            fillRegistrationForm(response);
        }
    });
}
```

[C#]

```
private void streamlineAccountReg_Click(object sender, RoutedEventArgs e)
{
    string[] allScopes = { "wl.signin", "wl.basic", "wl.birthday", "wl.emails" };
    var scopes = new List<string>(allScopes);
    auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
    auth.LoginCompleted +=
```

```

        new EventHandler<LoginCompletedEventArgs>(StreamlineAccountReg_LoginComp
leted);
        auth.LoginAsync(scopes);
    }

void StreamlineAccountReg_LoginCompleted(object sender, LoginCompletedEventArgs
e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
        client.GetCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(CompleteAccountReg
istration_GetCompleted);
        client.GetAsync("me");
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}

```

[REST]

In REST, once you make the appropriate GET call, the user's info is inside of the response. Here's how to make a GET call where the signed-in user is represented as `me` and where `ACCESS_TOKEN` is the actual access token.

```
GET https://apis.live.net/v5.0/me?access_token=ACCESS_TOKEN
```

The custom `onLoginAccountReg` function in JavaScript and the custom `streamlineAccountReg_LoginCompleted` method in C# is really a wrapper around the **WL.api** function in JavaScript and the **LiveAuthClient.GetAsync** method in C#, respectively. These make calls by using the Live Connect Representational State Transfer (REST) syntax behind the scenes. In this case, the request is for `me`, which returns the signed-in user's profile info. When the Live Connect APIs return the data to your app, the **WL.api** method in JavaScript and **LiveAuthClient.GetAsync** method in C#, respectively, can call a custom `fillRegistrationForm` function in JavaScript or a custom `completeAccountRegistration_GetCompleted` event handler method in C#, respectively, which inserts the user's info into user interface controls. Here's how.

[JavaScript]

```

function fillRegistrationForm(user) {
    document.getElementById("first_name").innerText = user.first_name;
    document.getElementById("last_name").innerText = user.last_name;
    document.getElementById("email").innerText = user.emails.preferred;
    document.getElementById("gender").innerText = user.gender;
    document.getElementById("birthday").innerText = user.birth_month + " " + use
r.birth_day + " " + user.birth_year;
}

```

[C#]

```
void CompleteAccountRegistration_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        IDictionary<string, object> emails = null;
        if (e.Result.ContainsKey("first_name"))
        {
            if (e.Result["first_name"] != null)
            {
                first_nameTextBox.Text = e.Result["first_name"].ToString();
            }
        }
        if (e.Result.ContainsKey("last_name"))
        {
            if (e.Result["last_name"] != null)
            {
                last_nameTextBox.Text = e.Result["last_name"].ToString();
            }
        }
        if (e.Result.ContainsKey("emails"))
        {
            if (e.Result["emails"] != null)
            {
                emails = (IDictionary<string, object>)e.Result["emails"];
                if (emails.ContainsKey("preferred"))
                {
                    if (emails["preferred"] != null)
                    {
                        emailTextBox.Text = emails["preferred"].ToString();
                    }
                }
            }
        }
        if (e.Result.ContainsKey("gender"))
        {
            if (e.Result["gender"] != null)
            {
                genderTextBox.Text = e.Result["gender"].ToString();
            }
        }
        if (e.Result.ContainsKey("birth_month") &&
            e.Result.ContainsKey("birth_day") &&
            e.Result.ContainsKey("birth_year"))
        {
            if (e.Result["birth_month"] != null &&
                e.Result["birth_day"] != null &&
                e.Result["birth_month"] != null)
            {
                birthdayTextBox.Text =
                    e.Result["birth_month"].ToString() + " " +
                    e.Result["birth_day"].ToString() + " " +
                    e.Result["birth_year"].ToString();
            }
        }
    }
    else
    {
        this.infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
    }
}
```

In this code, `user.structure_name` in JavaScript or `user.Result["structure_name"].ToString()` in C# is used, respectively, to get the info in the corresponding structure, such as **first_name**. Note that in C#, you must first check to make sure that the structure name exists and that if it exists, its value is not **null**. Also, note that in JavaScript, you can write `user.emails.preferred` to get at the `preferred` structure's info, but in C#, it's a little more involved.

Tip For Metro style apps using C#, you can shorten calls like `e.Result["first_name"]` to something easier like `meResult.first_name`, and write less code overall, by using the **dynamic** keyword. For details, see the [Metro style apps](#) topic's "Code samples" section in the `MainPage.xaml.cs` code example.

[REST]

In REST, once the info is returned in JavaScript Object Notation (JSON) format, you can parse through the results to get at the individual structures within.

The Live Connect APIs return info to your app in the form of a JSON object, with structure that correspond to the scopes that you requested for your app. Here's a simplified example of an object that is returned.

```
{  
    first_name : "Luka",  
    last_name : "Abrus",  
    emails : {  
        preferred: "luka.abrus@fabrikam.com"  
    },  
    birth_month : 6,  
    birth_date : 26,  
    birth_year : 1973  
}
```

With the form filled in, the user can then click a "submit" button, which calls whatever code you have for creating an app-specific account. (Info about how to write this code falls outside the scope of this topic, but you may want to consider visiting the [Live Connect Forum](#) to ask for help.)

Note that this is just one way of working with the user's info. For example, you could bypass the form altogether and send the user's info directly to some server-side script for processing.

[Top](#)

Next steps

This topic shows how to streamline app-specific account registration using the Live Connect APIs. It shows how to request specific scopes that your app needs to work with certain portions of user info. From here, you can learn how to work with additional user info by going to [Greeting the user](#), [Contacting users](#), and [Determining a user's age](#).

[Top](#)

Contacting users

[This documentation is preliminary and is subject to change.]

Users can store a lot of their contact info in Live Connect. With the Live Connect APIs, if the user provides the right info, you can get this info to contact a user by email, by postal mail, or by phone.

This topic describes and demonstrates how to acquire a user's available contact info. After the code gets the user's contact information, it displays it in a set of user interface controls.

This topic assumes that your app has already successfully initialized the APIs, the user has successfully signed in, and he or she has consented to the scopes **wl.basic** (to get the user's name), **wl.emails** (to get the user's email address, if it exists), **wl.postal_addresses** (to get the user's postal address info, if it exists), and **wl.phone_numbers** (to get the user's phone number info, if it exists). For details about this, see [Signing users in](#) and [Obtaining user consent](#).

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Getting and working with user info](#)
- [Next steps](#)

Getting and working with user info

To get a user's contact info, use the [WL.api](#) method in JavaScript or the [LiveConnectClient.GetAsync](#) method in C#. Here's how.

[JavaScript]

```
function onLoginContactUser() {
    WL.api({
        path: "me",
        method: "GET"
    }, function (response) {
        if (!response.error) {
            fillContactForm(response);
        }
    });
}

function fillContactForm(user) {
    document.getElementById("first_name").innerText = user.first_name;
    document.getElementById("last_name").innerText = user.last_name;
    document.getElementById("email").innerText = user.emails.preferred;
    document.getElementById("address").innerText = user.addresses.personal.street +
        " " +
        user.addresses.personal.city + " " +
        user.addresses.personal.state + " " +
        user.addresses.personal.postal_code + " " +
        user.addresses.personal.region
    document.getElementById("phone").innerText = user.phones.personal;
}
```

[C#]

```
private void contactingUsers_Click(object sender, RoutedEventArgs e)
{
    string[] allScopes = { "wl.signin", "wl.basic", "wl.postal_addresses", "wl.phone_numbers", "wl.emails" };
    var scopes = new List<string>(allScopes);
    auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
    auth.LoginCompleted +=
        new EventHandler<LoginCompletedEventArgs>(ContactingUsers_LoginCompleted);
    auth.LoginAsync(scopes);
}

void ContactingUsers_LoginCompleted(object sender, LoginCompletedEventArgs e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
        client.GetCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(FillInContactForm_GetCompleted);
        client.GetAsync("me");
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}

void FillInContactForm_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        IDictionary<string, object> emails = null;
        IDictionary<string, object> addresses = null;
        IDictionary<string, object> personal = null;
        IDictionary<string, object> phones = null;
        if (e.Result.ContainsKey("first_name"))
        {
            if (e.Result["first_name"] != null)
            {
                first_nameTextBox.Text = e.Result["first_name"].ToString();
            }
        }
        if (e.Result.ContainsKey("last_name"))
        {
            if (e.Result["last_name"] != null)
            {
                last_nameTextBox.Text = e.Result["last_name"].ToString();
            }
        }
        if (e.Result.ContainsKey("emails"))
        {
            if (e.Result["emails"] != null)
```

```

        {
            emails = (IDictionary<string, object>)e.Result["emails"];
            if (emails.ContainsKey("preferred"))
            {
                if (emails["preferred"] != null)
                {
                    emailTextBox.Text = emails["preferred"].ToString();
                }
            }
        }
    if (e.Result.ContainsKey("addresses"))
    {
        if (e.Result["addresses"] != null)
        {
            addresses = (IDictionary<string, object>)e.Result["addresses"];
            if (addresses["personal"] != null)
            {
                personal = (IDictionary<string, object>)addresses["personal"];
            }
            if (personal.ContainsKey("street") &&
                personal.ContainsKey("city") &&
                personal.ContainsKey("state") &&
                personal.ContainsKey("postal_code") &&
                personal.ContainsKey("region"))
            {
                if (personal["street"] != null &&
                    personal["city"] != null &&
                    personal["state"] != null &&
                    personal["postal_code"] != null &&
                    personal["region"] != null)
                {
                    addressTextBox.Text =
                        personal["street"].ToString() + " " +
                        personal["city"].ToString() + " " +
                        personal["state"].ToString() + " " +
                        personal["postal_code"].ToString() + " " +
                        personal["region"].ToString();
                }
            }
        }
    }
    if (e.Result.ContainsKey("phones"))
    {
        if (e.Result["phones"] != null)
        {
            phones = (IDictionary<string, object>)e.Result["phones"];
            if (phones.ContainsKey("personal"))
            {
                if (phones["personal"] != null)
                {
                    phoneTextBox.Text = phones["personal"].ToString();
                }
            }
        }
    }
}
else
{
    this.infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
}

```

```
}
```

The custom `onLoginContactUser` function in JavaScript and the custom `contactingUsers_LoginCompleted` method in C# is really a wrapper around the **WL.api** function in JavaScript and the **LiveAuthClient.GetAsync** method in C#, respectively. In this case, the request is for **me**, which returns the signed-in user's profile info.

When the Live Connect APIs return the data to your app, the **WL.api** method in JavaScript and **LiveAuthClient.GetAsync** method in C# calls a custom `fillContactForm` function in JavaScript or a custom `FillInContactForm_GetCompleted` event handler method in C#, respectively, which inserts the user's info into user interface controls.

In this code, `user.structure_name` in JavaScript or `e.Result["structure_name"].ToString()` in C# is used, respectively, to get the info in the corresponding structure, such as **first_name**. Note that in C#, you must first check to make sure that the structure name exists and that if it exists, its value is not **null**. Also, note that in JavaScript, you can write `user.addresses.personal.postal_code` to get at the `postal_code` structure's info, but in C#, it's a little more involved.

Tip For Metro style apps using C#, you can shorten calls like `e.Result["first_name"]` to something easier like `meResult.first_name`, and write less code overall, by using the **dynamic** keyword. For details, see the [Metro style apps](#) topic's "Code samples" section in the `MainPage.xaml.cs` code example.

[REST]

In Representational State Transfer (REST), once you make the appropriate GET call, the user's info is inside of the response. The response is returned in JavaScript Object Notation (JSON) format, which you can parse through to get at the individual structures within. Here's how to make a GET call where the signed-in user is represented as `me` and where `ACCESS_TOKEN` is the actual access token.

```
GET https://apis.live.net/v5.0/me?access_token=ACCESS_TOKEN
```

Objects such as **postal_addresses**, **emails**, and **phone_numbers** contain structures themselves. In addition, the **emails** object contains multiple structures that correspond to different email addresses that a user might have in his or her Live Connect profile info: **preferred**, **personal**, and **business**. Here are some email addresses, postal addresses, and phone numbers that the Live Connect APIs returns for an example user, in JSON format.

```
{
  ...
  "emails": {
    "preferred": "lukas.abrus@fabrikam.com",
    "account": "lukas.abrus.account@fabrikam.com",
    "personal": "lukas.abrus.personal@fabrikam.com",
    "business": "lukas.abrus.business@fabrikam.com"
  },
  "addresses": {
    "personal": {
      "street": "1235 Main St",
      "street_2": null,
      "city": "Buffalo",
    }
  }
}
```

```
        "postal_code": "98074",
        "state": "NY",
        "region": "USA"
    },
    "business": {
        "street": "One Fabrikam Place",
        "street_2": "Unit 5",
        "city": "Buffalo",
        "postal_code": "98074",
        "state": "WA",
        "region": "USA"
    }
},
"phones": {
    "personal": "(425)555-0101",
    "business": "(425)555-0155",
    "mobile": "(425)555-0167"
}
...
}
```

The [REST API](#) topic offers further details about the info that is contained in the **emails**, **phones**, and **addresses** objects.

The APIs return **null** for any object or structure that does not have a value. We recommend that your code check for null values in situations where your app needs to check for a value, especially if you're using C#.

[Top](#)

Next steps

This topic demonstrates how to use the **wl.postal_addresses**, **wl.email_addresses**, and **wl.phone_numbers** scopes to get a user's contact info. From here, you can learn how to work with additional user info by going to [Greeting the user](#), [Streamlining account registration](#), and [Determining a user's age](#).

[Top](#)

Determining a user's age

[This documentation is preliminary and is subject to change.]

When your apps interact with users, certain pieces of info can become critical to ensuring the right user experience, like knowing a user's age. Knowing how old a user is can be very important—especially if your apps allows actions such as in-app purchases for age-restricted media.

This topic describes and demonstrates how to get a user's age, if he or she chooses to store this info in Live Connect. In the example provided here, a purchase control is disabled until the app can verify that the signed-in user is at least 18 years old.

This topic assumes that your app has already successfully initialized the APIs, the user has successfully signed in, and he or she has consented to the **wl.birthday** scope (to get the user's birthday info, if it exists). For details about this, see [Signing users in](#) and [Obtaining user consent](#).

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Getting and working with user info](#)
- [Next steps](#)

Getting and working with user info

To get a user's age, use the [WL.api](#) method in JavaScript and the [LiveConnectClient.GetAsync](#) method in C#. Here's how.

[JavaScript]

```
function onLoginUserAge() {
    WL.api({
        path: "me",
        method: "GET"
    }, function (response) {
        if (!response.error && response.birth_year) {
            enablePurchaseButton(response);
        }
    });
}

function enablePurchaseButton(user) {
    var date = new Date();
    var year = date.getFullYear();
    var age = year - user.birth_year;
    if (age >= 18) {
        document.getElementById("purchaseButton").disabled = "";
    } else {
        document.getElementById("purchaseButton").disabled = "disabled";
    }
}
```

[C#]

```

private void userAge_Click(object sender, RoutedEventArgs e)
{
    string[] allScopes = { "wl.signin", "wl.birthday" };
    var scopes = new List<string>(allScopes);
    auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
    auth.LoginCompleted +=
        new EventHandler<LoginCompletedEventArgs>(UserAge_LoginCompleted);
    this.auth.LoginAsync(scopes);
}

void UserAge_LoginCompleted(object sender, LoginCompletedEventArgs e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
        client.GetCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(EnablePurchase_Get
Completed);
        client.GetAsync("me");
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}

private void EnablePurchase_GetCompleted(object sender, LiveOperationCompletedEv
entArgs e)
{
    if (e.Error == null && e.Result.ContainsKey("birth_year"))
    {
        if (e.Result["birth_year"] != null)
        {
            DateTime date = DateTime.Now;
            int year = date.Year;
            int age = year - (int)e.Result["birth_year"];
            if (age >= 18)
            {
                purchaseButton.IsEnabled = true;
            }
            else
            {
                purchaseButton.IsEnabled = false;
            }
        }
    }
    else
    {
        this.infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
    }
}

```

The custom `onLoginUserAge` function in JavaScript and the custom `UserAge_LoginCompleted` method in C# is really a wrapper around the **WL.api** function in JavaScript and the **LiveAuthClient.GetAsync** method in C#, respectively. In

this case, the request is for **me**, which returns the signed-in user's profile info.

When the Live Connect APIs return the data to your app, the **WL.api** method in JavaScript and the **LiveAuthClient.GetAsync** method in C# calls a custom `enablePurchaseButton` function in JavaScript or a custom `EnablePurchase_GetCompleted` event handler method in C#, respectively, which attempts to verify the user's age depending on the birthday info that he or she stored in Live Connect.

In this code, `user.birth_year` in JavaScript and `e.Result["birth_year"]` in C# is used, respectively, to try to get the user's year of birth. Note that in C#, you must first check to make sure that the **birth_year** structure name exists and that if it exists, its value is not **null**.

Tip For Metro style apps using C#, you can shorten calls like `e.Result["birth_year"]` to something easier like `meResult.birth_year`, and write less code overall, by using the **dynamic** keyword. For details, see the [Metro style apps](#) topic's "Code samples" section in the `MainPage.xaml.cs` code example.

[REST]

In REST, once you make the appropriate GET call, the user's info is inside of the response. The response is returned in JavaScript Object Notation (JSON) format, which you can parse through to get at the individual structures within. Here's how to make a GET call where the signed-in user is represented as me.

```
GET https://apis.live.net/v5.0/me?access_token=ACCESS_TOKEN
```

Here's what the JSON-formatted info might look like that is returned for an example user.

```
{  
    first_name : "Luka",  
    last_name : "Abrus",  
    birth_month : 6,  
    birth_date : 26,  
    birth_year : 1968  
}
```

[Top](#)

Next steps

This topic demonstrates how to use the **wl.birthday** scope to get a user's birth year and determine whether the user is old enough to make a certain type of purchase through your app. From here, you can learn how to work with additional user info by going to [Greeting the user](#), [Streamlining account registration](#), and [Contacting users](#).

[Top](#)

Hotmail (contacts and calendars)

[This documentation is preliminary and is subject to change.]

Your apps can use the Live Connect APIs to work with a Hotmail user's contacts and calendars in these ways.

- [Contacts](#)

Create and read contacts.

- [Calendars](#)

Create, read, update, and delete calendars and subscribe to public calendars.

- [Friend finder](#)

Find a list of a user's friends who that user already knows on a website.

© 2011 Microsoft Corporation. All rights reserved.

Contacts

[This documentation is preliminary and is subject to change.]

Your apps can use the Live Connect APIs to create and read a Hotmail user's contacts.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#). To request the required scopes, see the code patterns in [Obtaining user consent](#).

- [Reading contacts](#)
- [Creating contacts](#)

Reading contacts

To get info about a contact, use code like this. The **wl.basic** scope is required.

[REST]

```
GET http://apis.live.net/v5.0/contact.de3413e60000000000000000000000?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function readContact_onClick() {
    WL.api({
        path: "contact.d785d3760000000000000000000000",
        method: "GET"
    }, onReadContactCallback);
}

function onReadContactCallback(response) {
    if (!response.error) {
        var id = response.id;
        var name = response.first_name + " " + response.last_name;
        var result = "ID: " + id + "<br/> Name: " + name;
        document.getElementById("resultDiv").innerHTML = result;
    }
}
```

[C#]

```
private void btnReadContact_Click(object sender, RoutedEventArgs e)
{
    LiveConnectClient readContact = new LiveConnectClient(session);
    readContact.GetCompleted += ReadContactProperties_GetCompleted;
    readContact.GetAsync("contact.d8487c0200000000000000000000");
}

void ReadContactProperties_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string name = e.Result["first_name"].ToString() + " " + e.Result["last_name"].ToString();
        MessageBox.Show("Contact: " + name);
    }
}
```

In the preceding code, change the contact ID to this.

- A different contact ID to get info about another contact.
- `me/contacts` to get info about all of the signed-in users' contacts.
- `me/friends` to get info about all of the signed-in users' friends.

Note A *friend* is another Hotmail user whom a user has invited to be a friend and who in turn has accepted the friend request. A friend is a contact that has its **is_friend**

structure set to **true**. (You can't set the **is_friend** structure programmatically.)

- *USER_ID/contacts* to get info about all of the contact entries for the user represented by *USER_ID*.
- *USER_ID/friends* to get info about all of the friend entries for the user represented by *USER_ID*.

You can also do this.

- Get a limited number of contact or friend entries by using the *limit* parameter in the preceding code, specifying the number of entries to get. For instance, to get the first two contact entries for the signed-in user, use `me/contacts?limit=2`.
- Set the first contact or friend entry to start getting by using the *offset* parameter in the preceding code, specifying the index of the first entry to get. For instance, to get two contact entries for the signed-in user starting at the third contact entry, use `me/contacts?limit=2&offset=3`.

Note In the JavaScript Object Notation (JSON)-formatted object that's returned, you can look in the **paging** object for the **previous** and **next** structures, if they apply, to get the *offset* and *limit* parameter values of the previous and next entries, if they exist.

[Top](#)

Creating contacts

To add a new contact to a user's contacts list, use code like this. The **wl.contacts_create** scope is required.

[REST]

```
POST https://apis.live.net/v5.0/me/contacts

Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json

{  
    first_name: "Roberto",  
    last_name: "Tamburello"  
}
```

[JavaScript]

```
function createContact_onClick() {  
    var contact = {  
        first_name: "William",  
        last_name: "Flash"  
    };  
    WL.api({  
        path: "me/contacts",  
        method: "POST",  
        body: contact  
    }, onContactCreatedCallback);  
}  
  
function onContactCreatedCallback(response) {  
    if (!response.error) {  
        var id = response.id;  
        var name = response.first_name + " " + response.last_name;  
        var result = "ID: " + id + "<br/> Name: " + name;  
        document.getElementById("resultDiv").innerHTML = result;  
    }  
}
```

[C#]

```

private void btnCreateContact_Click(object sender, RoutedEventArgs e)
{
    LiveConnectClient createContact = new LiveConnectClient(session);
    var contact = new Dictionary<string, object>();
    contact.Add("first_name", "Roberto");
    contact.Add("last_name", "Tamburello");
    createContact.PostCompleted += CreateContactProperties_PostCompleted;
    createContact.PostAsync("me/contacts", contact);
}

void CreateContactProperties_PostCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string id = e.Result["id"].ToString();
        string name = e.Result["first_name"].ToString() + " " + e.Result["last_name"].ToString();
        MessageBox.Show("Contact" + name + " created with id :" + id);
    }
}

```

In the preceding code, you can change `me` to `USER_ID`, where `USER_ID` represents the signed-in user's user ID.

For details about the minimum required and optional structures that your app must provide when using POST, see the [REST API](#) topic's "Contact object" section.

Note The Live Connect APIs don't enable you to create friend entries. You can, however, create a contact entry and then use the Hotmail UI to invite that contact to become a friend.

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Calendars

[This documentation is preliminary and is subject to change.]

Your apps can use the Live Connect APIs to create, read, update, and delete a Hotmail user's calendars. Your apps can also subscribe to public calendars, such as a list of holidays.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#). To request the required scopes, see the code patterns in [Obtaining user consent](#).

- [Reading calendar properties](#)
- [Deleting calendars](#)
- [Creating calendars](#)
- [Updating calendar properties](#)
- [Subscribing to public calendars](#)
- [Reading calendar events](#)
- [Deleting calendar events](#)
- [Creating calendar events](#)
- [Updating calendar events](#)

Reading calendar properties

To get info about a calendar, use code like this. The **wl.calendars** scope is required.

[REST]

```
GET https://apis.live.net/v5.0/calendar.611afb17fa9448f28cdb8277e8ffeb77?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function onReadCalendar() {
    WL.api({
        path: "calendar.407a520a616e43438ab87875b064c290",
        method: "GET"
    }, onReadCalendarCallback);
}

function onReadCalendarCallback(response) {
    if (!response.error) {
        var id = response.id;
        var name = response.name;
        var result = "ID: " + id + "<br/> Name: " + name;
        document.getElementById("resultDiv").innerHTML = result;
    }
}
```

[C#]

```
private void btnReadCalendar_Click(object sender, RoutedEventArgs e)
{
    LiveConnectClient readCalendar = new LiveConnectClient(session);
    readCalendar.GetCompleted += ReadCalendar_GetCompleted;
    readCalendar.GetAsync("calendar.407a520a616e43438ab87875b064c290");
}

void ReadCalendar_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string calendarName = e.Result["name"].ToString();
        MessageBox.Show("Calendar name : " + calendarName);
    }
}
```

In the preceding code, change the calendar ID to this.

- A different calendar ID to get info about another calendar.
- `me/calendars` to get info about all of the signed-in user's calendars.
- `USER_ID/calendars` to get info about all of the calendars for the signed-in user represented by `USER_ID`.

You can also do this.

- Get a limited number of calendar entries by using the `limit` parameter in the preceding code, specifying the number of entries to get. For instance, to get the first two calendar entries for the signed-in user, use `me/calendars?limit=2`.
- Set the first calendar entry to start getting by using the `offset` parameter in the preceding code, specifying the index of the first entry to get. For instance, to get two calendar entries for the signed-in user starting at the third contact entry, use `me/calendars?limit=2&offset=3`.

Note In the JavaScript Object Notation (JSON)-formatted object that's returned, you can look in the **paging** object for the **previous** and **next** and structures, if they apply, to get the `offset` and `limit` parameter values of the previous and next entries, if they exist.

[Top](#)

Deleting calendars

To remove a calendar, use code like this. The **wl.calendars_update** scope is required.

[REST]

```
DELETE https://apis.live.net/v5.0/calendar.611afb17fa9448f28cdb8277e8ffeb77?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function onDeleteCalendar() {
    WL.api({
        path: "calendar.407a520a616e43438ab87875b064c290",
        method: "DELETE"
    }, onDeleteCalendarCallback);
}

function onDeleteCalendarCallback(response) {
    if (!response.error) {
        var result = "Deleted.";
        document.getElementById("resultDiv").innerHTML = result;
    }
}
```

[C#]

```
private void btnDeleteCalendar_Click(object sender, RoutedEventArgs e)
{
    LiveConnectClient deleteCalendar = new LiveConnectClient(session);
    deleteCalendar.DeleteCompleted += DeleteCalendar_DeleteCompleted;
    deleteCalendar.DeleteAsync("calendar.407a520a616e43438ab87875b064c290");
}

void DeleteCalendar_DeleteCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Deleted.");
    }
}
```

In the preceding code, change the calendar ID to the calendar ID of the calendar you want to remove.

[Top](#)

Creating calendars

To add a new calendar, use code like this. The **wl.calendars_update** scope is required.

[REST]

```
POST https://apis.live.net/v5.0/me/calendars

Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json

{
    name: "My example calendar"
}
```

[JavaScript]

```
function onCreateCalendar() {
    var calendar =
    {
        name: "My example calendar"
    };
    WL.api({
        path: "me/calendars",
        method: "POST",
        body: calendar
    }, onCreateCalendarCallback);
}

function onCreateCalendarCallback(response) {
    if (!response.error) {
        var id = response.id;
        var name = response.name;
        var result = "ID: " + id + "<br/> Name: " + name;
        document.getElementById("resultDiv").innerHTML = result;
    }
}
```

[C#]

```
private void btnCreateCalendar_Click(object sender, RoutedEventArgs e)
{
    LiveConnectClient createCalendar = new LiveConnectClient(session);
    var calendar = new Dictionary<string, object>();
    calendar.Add("name", "My example calendar");
    createCalendar.PostCompleted += CreateCalendar_PostCompleted;
    createCalendar.PostAsync("me/calendars", calendar);
}

void CreateCalendar_PostCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string id = e.Result["id"].ToString();
        string name = e.Result["name"].ToString();
    }
}
```

```
        MessageBox.Show("Calendar " + name + " created with ID " + id);
    }
}
```

In the preceding code, you can change `me` to *USER_ID*, where *USER_ID* represents the signed-in user's user ID.

For details about the minimum required and optional structures that your app must provide when using POST, see the [REST API](#) topic's "Calendar object" section.

[Top](#)

Updating calendar properties

To change info for an existing calendar, use code like this. The **wl.calendars_update** scope is required.

[REST]

```
PUT https://apis.live.net/v5.0/calendar.611afbf17fa9448f28cdb8277e8ffeb77

Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json

{
    name: "My example calendar has changed"
}
```

[JavaScript]

```
function onUpdateCalendar() {
    var calendar =
    {
        name: "My example calendar updated"
    };
    WL.api({
        path: "calendar.407a520a616e43438ab87875b064c290",
        method: "PUT",
        body: calendar
    }, onUpdateCalendarCallback);
}

function onUpdateCalendarCallback(response) {
    if (!response.error) {
        var id = response.id;
        var name = response.name;
        var result = "ID: " + id + "<br/> Name: " + name;
        document.getElementById("resultDiv").innerHTML = result;
    }
}
```

[C#]

```
private void btnUpdateCalendar_Click(object sender, RoutedEventArgs e)
{
    LiveConnectClient updateCalendar = new LiveConnectClient(session);
    var calendar = new Dictionary<string, object>();
    calendar.Add("name", "My example calendar has changed");
    updateCalendar.PutCompleted += UpdateCalendar_PutCompleted;
    updateCalendar.PutAsync("calendar.407a520a616e43438ab87875b064c290", calendar);
}

void UpdateCalendar_PutCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string newCalendarName = e.Result["name"].ToString();
        MessageBox.Show("Calendar name updated: " + newCalendarName);
    }
}
```

In the preceding code, change the calendar ID to the calendar ID of the calendar you want to change.

Note To update a calendar's properties, the calendar's **permissions** structure must have a value of **owner**, **co-owner**, or **read-write**.

For details about the minimum required and optional structures that your app must provide when using PUT, see the [REST API](#) topic's "Calendar object" section.

[Top](#)

Subscribing to public calendars

To subscribe to a public calendar, use code like the following. The subscription URL must use the webcal scheme and the calendar file must use the iCalendar format and the file extension .ics. The **wl.calendars_update** scope is required.

[REST]

```
POST https://apis.live.net/v5.0/me/calendars

Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json

{
    name: "Anime/game character birthdays",
    subscription_location: "webcal://ical.mac.com/fredduck/Anime47Game32Birthdays.ics"
}
```

[JavaScript]

```
function onSubscribeCalendar() {
    var calendar =
    {
        name: "Anime/game character birthdays",
        subscription_url: "webcal://ical.mac.com/fredduck/Anime47Game32aBirthdays.ics"
    };
    WL.api({
        path: "me/calendars",
        method: "POST",
        body: calendar
    }, onSubscribeCalendarCallback);
}

function onSubscribeCalendarCallback(response) {
    if (!response.error) {
        var result = "Subscribed.";
        document.getElementById("resultDiv").innerHTML = result;
    }
}

[C#]
```

```
private void btnSubscribeCalendar_Click(object sender, RoutedEventArgs e)
{
    var calendar = new Dictionary<string, object>();
    calendar.Add("name", "Anime/game character birthdays");
    calendar.Add("subscription_url", "webcal://ical.mac.com/fredduck/Anime47Game32Birthdays.ics");
    LiveConnectClient subscribeCalendar = new LiveConnectClient(session);
    subscribeCalendar.PostCompleted += subscribeCalendar_PostCompleted;
    subscribeCalendar.PostAsync("me/calendars", calendar);
}

void subscribeCalendar_PostCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Subscribed.");
    }
}
```

[Top](#)

Reading calendar events

To get info about an event, use code like this. The **wl.calendars** scope is required.

[REST]

```
GET https://apis.live.net/v5.0/event.c76d2a8c4e674ff9bc96f17dee3c34c9.b987b00b77194723b8c16bbe57a03037?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function onReadEvent() {
    WL.api({
        path: "event.d050fdffbb314d9bb40bc20a3c961d84.57f866fd013f4a14adcf18d9a9dc23bc",
        method: "GET"
    }, onReadEventCallback);
}

function onReadEventCallback(response) {
    if (!response.error) {
        var id = response.id;
        var locations = response.location;
        var result = "ID: " + id + "<br/> Location: " + locations;
        document.getElementById("resultDiv").innerHTML = result
    }
}
```

[C#]

```
private void btnReadEvent_Click(object sender, RoutedEventArgs e)
{
    // You must request permission for wl.calendars.
    LiveConnectClient readEvent = new LiveConnectClient(session);
    readEvent.GetCompleted += readEvent_GetCompleted;
    readEvent.GetAsync("event.d050fdffbb314d9bb40bc20a3c961d84.57f866fd013f4a14adcf18d9a9dc23bc");
}

void readEvent_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string eventLoacation = e.Result["location"].ToString();
        MessageBox.Show("Location: " + eventLoacation);
    }
}
```

In the preceding code, change the event ID to this.

- A different event ID to get info about a different event.
- `me/events` to get info about events in all of the signed-in user's calendars.
- `USER_ID/events` to get info about events in all of the calendars for the signed-in user represented by `USER_ID`.

- `CALENDAR_ID/events` to get info about events in the calendar represented by `CALENDAR_ID`.

You can also do this.

- Get a limited number of events based on their starting and ending times in coordinated universal time (UTC) format by using the `start_time` and `end_time` parameters in the preceding code. For instance, to get all events between and including August 1st and 3rd for a calendar, use `CALENDAR_ID/events?start_time=2011-08-01T00:00:00Z&end_time=2011-08-03T00:00:00Z`. For info about time formatting options here, see the [Creating calendar events](#) section later in this topic.
- Note** If you don't specify the `start_time` and `end_time` parameters, by default you will get events only between the current day and 30 days into the future. If there are no events within this time period, you will get an empty result.

Note You can get info about recurring events. However, you can get only the first occurrence of a recurring event for the specified time period.

[Top](#)

Deleting calendar events

To remove an event, use code like this. The `wl.calendars_update` scope is required.

[REST]

```
DELETE https://apis.live.net/v5.0/event.c76d2a8c4e674ff9bc96f17dee3c34c9.b987b00b77194723b8c16bbe57a03037?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function onDeleteEvent() {
    WL.api({
        path: "event.d050fdffbb314d9bb40bc20a3c961d84.57f866fd013f4a14adcf18d9a9dc23bc",
        method: "DELETE"
    }, onDeleteEventCallback);
}

function onDeleteEventCallback(response) {
    if (!response.error) {
        var result = "Deleted!";
        document.getElementById("resultDiv").innerHTML = result;
    }
}
```

[C#]

```
private void btnDeleteEvent_Click(object sender, RoutedEventArgs e)
{
    LiveConnectClient deleteEvent = new LiveConnectClient(session);
    deleteEvent.DeleteCompleted += deleteEvent_DeleteCompleted;
    deleteEvent.DeleteAsync("event.d050fdffbb314d9bb40bc20a3c961d84.57f866fd013f4a14adcf18d9a9dc23bc");
}

void deleteEvent_DeleteCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Event deleted.");
    }
}
```

In the preceding code, change the event ID to the event ID of the event you want to remove.

Note You can delete recurring events.

[Top](#)

Creating calendar events

To add a new event in the user's default calendar, use code like this. The `wl.events_create` scope is required.

[REST]

```
POST https://apis.live.net/v5.0/me/events
```

```
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    name: "Family Dinner",
    description: "Dinner with Cynthia's family",
    start_time: "2011-03-22T01:30:00-08:00",
    end_time: "2011-03-22T03:00:00-08:00",
    location: "Coho Vineyard and Winery, 123 Main St., Redmond WA 19532",
    is_all_day_event: false,
    availability: "busy",
    visibility: "public"
}
```

[JavaScript]

```
function onCreateEvent() {
    var event =
    {
        name: "Family Dinner",
        description: "Dinner with Cynthia's family",
        start_time: "2011-04-22T01:30:00-08:00",
```

```

        end_time: "2011-04-22T03:00:00-08:00",
        location: "Coho Vineyard and Winery, 123 Main St., Redmond WA 19532",
        is_all_day_event: "false",
        availability: "busy",
        visibility: "public"
    };
    WL.api({
        path: "me/events",
        method: "POST",
        body: event
    }, onCreateEventCallback);
}

function onCreateEventCallback(response) {
    if (!response.error) {
        var id = response.id;
        var name = response.name;
        var result = "ID: " + id + "<br/> Name: " + name;
        document.getElementById("resultDiv").innerHTML = result;
    }
}
[C#]

private void btnCreateEvent_Click(object sender, RoutedEventArgs e)
{
    var calEvent = new Dictionary<string, object>();
    calEvent.Add("name", "Family Dinner");
    calEvent.Add("description", "Dinner with Cynthia's family");
    calEvent.Add("start_time", "2011-03-22T01:30:00-08:00");
    calEvent.Add("end_time", "2011-03-22T03:00:00-08:00");
    calEvent.Add("location", "Coho Vineyard and Winery, 123 Main St., Redmond WA 19532");
    calEvent.Add("is_all_day_event", false);
    calEvent.Add("availability", "busy");
    calEvent.Add("visibility", "public");
    LiveConnectClient createEvent = new LiveConnectClient(session);
    createEvent.PostCompleted += createEvent_PostCompleted;
    createEvent.PostAsync("me/events", calEvent);
}

void createEvent_PostCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string id = e.Result["id"].ToString();
        string name = e.Result["name"].ToString();
        MessageBox.Show("Event " + name + " created with ID " + id);
    }
}

```

In the preceding code, you can change `me` to `USER_ID`, where `USER_ID` represents the signed-in user's user ID.

To add an event to a user's calendar other than his or her default calendar, replace `me` with the desired calendar ID.

Note To create a calendar event, the corresponding calendar's **permissions** structure must have a value of **owner**, **co-owner**, or **read-write**.

The start and end times must be expressed in one of these supported formats.

- Times must begin in one of these formats: `yyyy-mm-ddThh:mm:ss`, `yyyy-mm-ddThh:mm:ss.sss`, or `yyyy-mm-dd hh:mm:ss`. For example, `2011-05-10T14:29:00.000` represents 10 May 2011 at 2:29 P.M.
 - Times must end in one of these formats: `+hhmm`, `-hhmm`, `+hh:mm`, `-hh:mm`, or `Z`, representing the number of hours and minutes as an offset to, or in coordination with, UTC. Thus `-0700` represents seven hours preceding UTC; applied to the first example, the result is `2011-05-10T14:29:00.000-0700`.
- Note** If `Z` is not specified, the time is treated as local time. Also, if `end_time` is not specified, the default end time is 30 minutes after the specified `start_time`.

For details about the minimum required and optional structures that your app must provide when using POST, see the [REST API](#) topic's "Event object" section.

Note You cannot create recurring events programmatically.

[Top](#)

Updating calendar events

To change info for an existing event, use code like this. The `wl.calendars` scope is required.

[REST]

```
PUT https://apis.live.net/v5.0/event.c76d2a8c4e674ff9bc96f17dee3c34c9.b987b00b77194723b8c16bbe57a03037
```

```
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    name: "My example event has changed"
}
```

[JavaScript]

```
function onUpdateEvent() {
    var calendar =
    {
```

```

        name: "My example event has changed"
    };
    WL.api({
        path: "event.d050fdffbb314d9bb40bc20a3c961d84.57f866fd013f4a14adcf18d9a9dc23bc",
        method: "PUT",
        body: calendar
    }, onUpdateEventCallback);
}

function onUpdateEventCallback(response) {
    if (!response.error) {
        var id = response.id;
        var name = response.name;
        var result = "ID: " + id + "<br/> Name: " + name;
        document.getElementById("resultDiv").innerHTML = result;
    }
}
[C#]

private void btnUpdateEvent_Click(object sender, RoutedEventArgs e)
{
    var calEvent = new Dictionary<string, object>();
    calEvent.Add("name", "My example event has changed");
    LiveConnectClient updateEvent = new LiveConnectClient(session);
    updateEvent.PutCompleted += updateEvent_PutCompleted;
    updateEvent.PutAsync("event.d050fdffbb314d9bb40bc20a3c961d84.57f866fd013f4a14adcf18d9a9dc23bc", calEvent);
}

void updateEvent_PutCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string newEventName = e.Result["name"].ToString();
        MessageBox.Show("Event name updated: " + newEventName);
    }
}

```

In the preceding code, change the event ID to the event ID of the event you want to change.

Note To update a calendar event, the corresponding calendar's **permissions** structure must have a value of **owner**, **co-owner**, or **read-write**.

For details about the minimum required and optional structures that your app must provide when using PUT, see the [REST API](#) topic's "Event object" section.

Note You cannot update recurring events programmatically.

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Friend finder

[This documentation is preliminary and is subject to change.]

In this scenario, when a user with a Microsoft account visits your website, you can use a combination of client-side JavaScript code and server-side code to determine whether any of the visiting user's contacts is also a registered user of your website.

For your website to participate in this scenario, you must create a hash of each email address that belongs to each registered user of your website. Each of these hashes must follow a specific format that the Live Connect APIs recognize. After you have created a hash for each of your registered users' email addresses, you can compare your list of hashes with the list of email-address hashes that Live Connect generates for the visiting user's contacts. Each matching hash represents a registered user of your website who is also one of the visiting user's contacts.

To fully leverage the JavaScript and C# code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

- [Generate a set of email-address hashes](#)
- [Get a list of email-address hashes with JavaScript or C#](#)
- [Get a list of email-address hashes with server-side code](#)
- [Server-side code sample](#)
- [Running the server-side code sample](#)
- [Integrating the server-side code sample](#)

Generate a set of email-address hashes

Generate a hash that corresponds to each email address for each of your website's registered users. Each hash must be generated by following these steps in this order:

1. Trim leading and trailing white spaces from the email address (for example, Someone@Example.org).
2. Trim leading and trailing white spaces from the requesting application's client ID (for example, 0000000603DB0F).
3. Concatenate the results of steps 1 and 2 (for example, Someone@Example.org0000000603DB0F).
4. Convert the result of step 3 to lowercase (someone@example.org0000000603db0f).
5. Calculate the SHA-256 value of step 4 and convert it to its hexadecimal equivalent (for example, 6A9F...63A4). (Part of the value is omitted here for brevity.)
6. Convert the results of step 5 to lowercase (for example, 6a9f...63a4).
The result of this step is the email-address hash.

[Top](#)

Get a list of email-address hashes with JavaScript or C#

You can get a list of a user's contacts' email addresses with code like this.

[JavaScript]

```
function friendFinder_onClick() {
    var scopes = ["wl.basic"];
    WL.login({
        scope: scopes
    }, onLoginFriendFinder);
}

function onLoginFriendFinder() {
    WL.api({
        path: "me/contacts",
        method: "GET"
    }, function (result) {
        if (!result.error) {
            var resultData = result.data;
            var emailHashes = new Array();
            for (i = 0; i < resultData.length; i++) {
                for (j = 0; j < resultData[i].email_hashes.length; j++) {
                    emailHashes[emailHashes.length] = resultData[i].email_hashes[j];
                }
            }
            var resultText = "";
            for (k = 0; k < emailHashes.length; k++) {
                resultText += emailHashes[k] + "\r\n";
            }
            document.getElementById("infoArea").setAttribute("rows", emailHashes.length);
            document.getElementById("infoArea").value = resultText;
        } else {
            alert("Error getting contacts: " + result.error.message);
        }
    });
}
```

[C#]

```
private void friendFinder_Click(object sender, RoutedEventArgs e)
{
    string[] allScopes = { "wl.basic" };
    var scopes = new List<string>(allScopes);
    auth = new LiveAuthClient(CLIENT_ID, REDIRECT_URL);
    auth.LoginCompleted
        += new EventHandler<LoginCompletedEventArgs>(FriendFinder_LoginCompleted);
    this.auth.LoginAsync(scopes);
}

void FriendFinder_LoginCompleted(object sender, LoginCompletedEventArgs e)
{
    if (e.Session != null &&
        e.Session.Status == LiveConnectSessionStatus.Connected)
    {
        session = e.Session;
```

```

        client = new LiveConnectClient(session);
        infoTextBlock.Text = "Signed in.";
        client.GetCompleted +=
            new EventHandler<LiveOperationCompletedEventArgs>(FriendFinder_GetCompleted);
        client.GetAsync("me");
    }
    else if (e.Error != null)
    {
        infoTextBlock.Text = "Error signing in: " + e.Error.ToString();
    }
    else
    {
        infoTextBlock.Text = "Error signing in.";
    }
}

private void FriendFinder_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    string message = "";
    if (e.Error == null)
    {
        List<object> data = null;
        IDictionary<string, object> contact = null;
        List<object> emailHashes = null;
        if (e.Result.ContainsKey("data"))
        {
            data = (List<object>)e.Result["data"];
            for (int i = 0; i < data.Count; i++)
            {
                contact = (IDictionary<string, object>)data[i];
                if (contact.ContainsKey("email_hashes"))
                {
                    emailHashes = (List<object>)contact["email_hashes"];
                    for (int j = 0; j < emailHashes.Count; j++)
                    {
                        message += emailHashes[j].ToString() + "\r\n";
                    }
                }
            }
        }
        infoTextBlock.Text = message;
    }
    else
    {
        this.infoTextBlock.Text = "Error calling API: " + e.Error.ToString();
    }
}

```

[Top](#)

Get a list of email-address hashes with server-side code

If a user signs in and consents to a **wl.basic** scope request, and you have an access token that corresponds to this consented request, you can retrieve a list of the user's contacts. For each email address that is available for a given contact, an email-address hash is displayed and you can extract it by using your preferred coding language and platform. To get the list of email-address hashes, make a server-side HTTP **GET** call with REST and the following request parameters.

```
https://apis.live.net/v5.0/me/contacts?access\_token=ACCESS\_TOKEN
```

In the preceding code examples, replace *ACCESS_TOKEN* with your access token string.

The JavaScript Object Notation (JSON)-formatted response collection will look similar to the following (in which ellipses appear in place of information that is omitted for brevity).

```
{  
    "data": [  
        {  
            "id": "contact.b4466224b2ca42798c3d4ea90c75aa56",  
            "first_name": "Henrik",  
            "last_name": "Jensen",  
            ...  
            "email_hashes": [  
                "9ecdb19f4eb8e04304c5d1280368c42e85b6e4fe39f08b0c837ec592b905a620",  
                "fc05492f50da6488aa14dcf221d395bcb29a4e43b43b250d60c68df4f831cad3"  
            ],  
            ...  
        }, {  
            ...  
        }  
    ]  
}
```

For more information about how to make server-side HTTP calls with REST, see [Server-side scenarios](#).

[Top](#)

Server-side code sample

This section provides a code sample (designed and tested with PHP version 5.3.6.0) that demonstrates how to create a set of email-address hashes and compare them with an existing set of email-address hashes in a MySQL server. This code sample consists of the following scripts:

- `hash_email_addresses.php`—Uses the SHA-256 cryptographic algorithm to produce email-address hashes and compares lists of email address hashes.
- `email_hashes_mysql.php`—Retrieves email-address hashes from a MySQL server. This script can also use an email-address hash to get a user's name from the same server.
- `callback_email.php`—Provides a list of email addresses to hash, compares the email-address hashes to an existing list of email addresses, and uses the matches to get the users' corresponding names.

hash_email_addresses.php

This script produces a list of email-address hashes by using the SHA-256 cryptographic algorithm, and it can compare lists of email address hashes to find matches in both lists.

```
<?php  
include_once 'globals.php';
```

```

function hashContactEmailAddresses ($emails, $clientID) {
    $trimmedID = trim($clientID);
    $hashedEmails = array();
    logMessage('Results of email address hashing:');
    foreach($emails as $item) {
        $lowercaseIdAndEmail = strtolower(trim($item) . $trimmedID);
        $lowercaseSHA2ValueToHex = strtolower(hash('sha256', $lowercaseIdAndEmail, false));
        logMessage('email = ' . $item);
        logMessage('lowercaseIdAndEmail = ' . $lowercaseIdAndEmail);
        logMessage('lowercaseSHA2ValueToHex = ' . $lowercaseSHA2ValueToHex);

        array_push($hashedEmails, $lowercaseSHA2ValueToHex);
    }
    logMessage('Contact email address hashes:');
    logMessage(var_dump($hashedEmails));
    return $hashedEmails;
}

function compareEmailAddressHashLists ($wlUserEmailHashes, $websiteEmailHashes)
{
    $matchedEmailHashes = array();
    logMessage('Windows Live user's contact email address hashes:');
    logMessage(var_dump($wlUserEmailHashes));
    logMessage('Email address hashes from MySQL:');
    logMessage(var_dump($websiteEmailHashes));
    foreach($wlUserEmailHashes as $wlItem) {
        if(array_search($wlItem, $websiteEmailHashes, true)) {
            array_push($matchedEmailHashes, $wlItem);
        }
    }
    logMessage('Matching email address hashes:');
    logMessage(var_dump($matchedEmailHashes));
    return $matchedEmailHashes;
}
?>
```

The `hashContactEmailAddresses` function takes a set of email addresses and an application's client ID and returns a set of email-address hashes. It uses the algorithm previously described.

The `compareEmailAddressHashLists` function takes a list of the email-address hashes of a user's contacts and compares it to a list of email-address hashes from a website. The function returns any email address hashes that appear in both lists.

email_hashes_mysql.php

This script retrieves email-address hashes from a MySQL server, and it can use a list of email-address hashes to get users' names from the same server.

```

<?php
include_once 'globals.php';

define('MYSQL_SERVER', '127.0.0.1');
define('MYSQL_USERNAME', 'root');
define('MYSQL_PASSWORD', 'mysql');

/*
These constants assume the following MySQL database and table structure:
```

```

users.users: user_entry_id (AUTO_INCREMENT), user_id, user_name
users.emails: email_entry_id (AUTO_INCREMENT), user_entry_id (matching entry in
users.users), email_address
users.email_hashes: email_hash_entry_id (AUTO_INCREMENT), email_entry_id (matchi
ng entry in users.emails),
    app_entry_id (matching entry in apps.apps), email_hash
apps.apps: app_entry_id (AUTO_INCREMENT), app_id, client_secret, redirect_uri
*/
define('USERS_DB', 'users');
define('APPS_DB', 'apps');
define('USERS_TABLE', USERS_DB . '.users');
define('EMAILS_TABLE', USERS_DB . '.emails');
define('EMAIL_HASHES_TABLE', USERS_DB . '.email_hashes');
define('APPS_TABLE', APPS_DB . '.apps');

$LINK;

function getEmailAddressHashesFromMySQL() {
    if (verifyConnection() == true) {
        $query = 'SELECT * FROM ' . EMAIL_HASHES_TABLE;
        $results = mysql_query($query);
        if (!$results) {
            logMessage('Error using query \'' . $query . '\'': ' . mysql_error());
        ;
            mysql_close($GLOBALS['LINK']);
            return false;
        }
        else {
            $emailHashList = array();
            logMessage('Email address hashes table from MySQL:');
            while($row = mysql_fetch_array($results)) {
                array_push($emailHashList, $row['email_hash']);
                logMessage('email_hash_entry_id = ' . $row['email_hash_entry_id']);
];
                logMessage('email_entry_id = ' . $row['email_entry_id']);
                logMessage('app_entry_id = ' . $row['app_entry_id']);
                logMessage('email_hash = ' . $row['email_hash']);
            }
            logMessage('Email address hashes from MySQL');
            logMessage(var_dump($emailHashList));
            return $emailHashList;
        }
    }
    else {
        logMessage('Error getting email address hashes from MySQL.');
        mysql_close($GLOBALS['LINK']);
        return false;
    }
}

function useEMailAddressHashesToGetUsersFromMySQL($hashList) {
    if (verifyConnection() == true) {
        logMessage('Matching email address hashes:');
        logMessage(var_dump($hashList));
        $users = array();
        logMessage('Matching user names from MySQL:');
        foreach($hashList as $item) {
            $query = 'SELECT user_name FROM ' . USERS_TABLE . ' WHERE user_entry
_id = ' .
                '(SELECT user_entry_id FROM ' . EMAILS_TABLE . ' WHERE email_ent

```

```

ry_id = ' .
        '(SELECT email_entry_id FROM ' . EMAIL_HASHES_TABLE . ' WHERE em
ail_hash = ' .
        '\' . $item . '\')';
$results = mysql_query($query);
if (!$results) {
    logMessage('Error using query \'' . $query . '\': ' . mysql_errno());
    mysql_close($GLOBALS['LINK']);
    return false;
}
else {
    while($row = mysql_fetch_row($results)) {
        array_push($users, $row[0]);
        logMessage('user_name = ' . $row[0]);
    }
}
logMessage('Matching user names:');
logMessage(var_dump($users));
return $users;
}
else {
    logMessage('Error getting user information based on email address hashes
from MySQL.');
    mysql_close($GLOBALS['LINK']);
    return false;
}
}

function verifyConnection() {
    if (!isset($GLOBALS['LINK'])) {
        $GLOBALS['LINK'] = mysql_connect(MYSQL_SERVER, MYSQL_USERNAME, MYSQL_PAS
WORD);
        if (!$GLOBALS['LINK']) {
            logMessage('Error connecting to MySQL server \'' . MYSQL_SERVER . '\
': ' . mysql_error());
            return false;
        }
        return true;
    }
    else {
        return true;
    }
}
?>
```

The `getEmailAddressHashesFromMySQL` function retrieves all of the email-address hashes that are in the MySQL server.

The `useEmailAddressHashesToGetUsersFromMySQL` function retrieves all of the users' names that match the provided list of email-address hashes.

callback_email.php

This script calls the preceding scripts. It provides a list of email addresses to hash; it compares the newly generated list of email-address hashes to an existing list of email address hashes in a MySQL server; and it uses any matches to get the users' corresponding names in the same server.

```

<?php
include_once 'hash_email_addresses.php';
include_once 'email_hashes_mysql.php';
include_once 'globals.php';

$emailArray = array(
    'Roberto.Tamburello@example.com',
    'r_tamburello@contoso.com',
    'Mariusz.Wolodzko@example.com');
$appClientID = '0000000603DB0F';

$hashedContactEmailAddresses = hashContactEmailAddresses ($emailArray, $appClientID);

if (!$emailAddressHashesFromMySQL = getEmailAddressHashesFromMySQL()) {
    logMessage('Error getting email address hashes from MySQL. Code execution has stopped.');
    die();
}
else {
    if (!$emailAddressHashMatches = compareEmailAddressHashLists(
        $hashedContactEmailAddresses, $emailAddressHashesFromMySQL)) {
        logMessage('Error comparing email address hash lists. Code execution has stopped.');
        die();
    }
    else {
        if (!$matchingUsers = useEmailAddressHashesToGetUsersFromMySQL($emailAddressHashMatches)) {
            logMessage('Error using email address hashes to get users from MySQL
.
.
.
        'Code execution has stopped.');
            die();
        }
        else {
            logMessage('Matching user names:');
            logMessage(var_dump($matchingUsers));
        }
    }
}
?>

```

This script creates email-address hashes from a set of hard-coded email addresses and a hard-coded application client ID. It then gets a list of all of the email-address hashes from a MySQL server. For each email-address hash that appears in both lists, the script uses the MySQL server to display the user's name that corresponds to that email-address hash.

[Top](#)

Running the server-side code sample

To run the preceding code sample, you must:

- Include the script `globals.php`, which is available in [Server-side scenarios](#), in the "globals.php" section.
- Have a MySQL server with the following databases, tables, and fields already present.

```

-- apps database
CREATE DATABASE apps;
USE apps;
CREATE TABLE apps (
    app_entry_id mediumint(9) NOT NULL AUTO_INCREMENT,
    app_id text NOT NULL,
    client_secret text,
    redirect_uri text NOT NULL,
    PRIMARY KEY (app_entry_id)
);
-- apps sample data
INSERT INTO apps (app_entry_id, app_id, client_secret, redirect_uri) VALUES
    (NULL, '00000000603DB0FC', 'MLWILLT555GicSriATma5qgyBXebRIKT', 'http%3A%2F%2
Fwww.contoso.com%2Fcallback.php');

-- users database
CREATE DATABASE users;
USE users;
CREATE TABLE users (
    user_entry_id mediumint(9) NOT NULL AUTO_INCREMENT,
    user_id text NOT NULL,
    user_name text NOT NULL,
    PRIMARY KEY (user_entry_id)
);
CREATE TABLE emails (
    email_entry_id mediumint(9) NOT NULL AUTO_INCREMENT,
    user_entry_id mediumint(9) NOT NULL,
    email_address text NOT NULL,
    PRIMARY KEY (email_entry_id)
);
CREATE TABLE email_hashes (
    email_hash_entry_id mediumint(9) NOT NULL AUTO_INCREMENT,
    email_entry_id mediumint(9) NOT NULL,
    app_entry_id mediumint(9) NOT NULL,
    email_hash text,
    PRIMARY KEY (email_hash_entry_id)
);
-- users sample data
INSERT INTO users (user_entry_id, user_id, user_name) VALUES
    (NULL, '00091865', 'Roberto Tamburello'),
    (NULL, '00082974', 'Mariusz Wolodzko'),
    (NULL, '00073185', 'Henrik Jensen');
INSERT INTO emails (email_entry_id, user_entry_id, email_address) VALUES
    (NULL, 1, 'Roberto.Tamburello@example.com'),
    (NULL, 1, 'r_tamburello@contoso.com'),
    (NULL, 2, 'Mariusz.Wolodzko@example.com'),
    (NULL, 3, 'Henrik.Jensen@example.com'),
    (NULL, 3, 'h_jensen@contoso.com');
INSERT INTO email_hashes (email_hash_entry_id, email_entry_id, app_entry_id, ema
il_hash) VALUES
    (NULL, 1, 1, '6a9f1d2067ba3546cdbcf74ee45c49c1d45893628939420e47ffab2376636
a4'),
    (NULL, 2, 1, '2882b6e9f7766326d5d08844a9d51c6f85f719a2a5fdeb5f20fe083d8f59bb
0e'),
    (NULL, 3, 1, '369c8d85752a6f4ffe1461a0c42a2c557026696f3b0456ced2451ab752e7f6
c6'),
    (NULL, 4, 1, 'c236dfb1fe82c4a67a2c01ef8ef43fe12eb1bdd56f52a50979729828c803f9
46'),
    (NULL, 5, 1, '1caf98006d4fc601886804fd1a9fc2dbdd8042d5db06f5938c771a78c5c8b
15');


```

[Top](#)

Integrating the server-side code sample

Integrating the server-side code sample

To integrate the server-side code sample with a code sample such as the one in [Server-side scenarios](#), in the "PHP Code Sample" section:

1. Include the hash_email_addresses.php and email_hashes_mysql.php scripts in the code sample in [Server-side scenarios](#).
2. Modify the globals.php script in [Server-side scenarios](#) and the email_hashes_mysql.php script to match the settings in your experimental environment.
3. Create a list of email-address hashes for a set of email addresses, and insert the hashes into your MySQL server's users.email_hashes table. Make sure that the list of email addresses exists for the user that will be signing in. Also make sure to use the application client ID that matches the one in your MySQL server's apps.apps table and in the globals.php script.
4. Modify the callback.php script in [Server-side scenarios](#) get a list of the email-address hashes for the signed-in user's contacts. Then, use the hash_email_addresses.php and email_hashes_mysql.php scripts to compare the two lists of email-address hashes and retrieve the contacts' names from the MySQL server for the matching email-address hashes.

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

SkyDrive (files and photos)

[This documentation is preliminary and is subject to change.]

Your apps can use the Live Connect APIs to work with a SkyDrive user's folders, files, albums, photos, videos, tags, and comments in these ways.

- [SkyDrive core concepts](#)

Important concepts that apply across most or all SkyDrive objects.

- [Folders and files](#)

Create, read, update, and delete folders and files, as well as get links to files so that others can access those files directly.

- [Albums, photos, videos, and tags](#)

Create, read, update, and delete albums, photos, and videos. Read tags on photos and videos.

- [Comments](#)

Read comments on folders, files, albums, photos, and videos.

© 2011 Microsoft Corporation. All rights reserved.

SkyDrive core concepts

[This documentation is preliminary and is subject to change.]

- Traversing the SkyDrive directory
- Object permissions
- File concepts
- Supported file formats

Traversing the SkyDrive directory

To use the Live Connect APIs to get info about a user's top-level folder in the SkyDrive directory, you make a request to `me/skydrive` or `USER_ID/skydrive`. (In Representational State Transfer (REST) you make this call with GET. In JavaScript you make this call with the `WL.api` function, specifying a *method* parameter of "GET". In C#, you make this call with the `LiveConnectClient.GetAsync` method.) In the JavaScript Object Notation (JSON)-formatted object that's returned, two structures are important to note: the **id** structure, representing the top-level folder's folder ID; and the **upload_location** structure, representing the location for the top-level folder into which you can upload files, photos, and videos.

If you make a request to `me/skydrive/files` or `USER_ID/skydrive/files`, the JSON-formatted object that's returned contains info about all of the child folders, child albums, and files within only the user's top-level folder.

From there, you can traverse a user's entire SkyDrive directory by making a request with `FOLDER_ID/files` or `ALBUM_ID/files`, where `FOLDER_ID` or `ALBUM_ID` represents the **id** structure corresponding to a folder ID or album ID, anywhere in the directory.

As with a user's top-level SkyDrive folder, you can use the **upload_location** structure for any folder or album to upload files to that folder or album. Additionally, you can use the **upload_location** for any file, photo, or video to overwrite the contents of that file, photo, or video.

For example, here's conceptually what a user's SkyDrive directory might look like.

```
GET https://apis.live.net/v5.0/me/skydrive?access_token=ACCESS_TOKEN
---
200 OK
{
  "id": "folder.a6b2a7e8f2515e5e",
  ...
  "upload_location": "https://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e/files",
  ...
  "type": "folder",
  ...
}
---
GET https://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e/files?access_token=ACCESS_TOKEN
---
200 OK
{
  "data": [
    {
      "id": "folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!110",
      ...
      "upload_location": "https://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!110/files/",
      ...
      "type": "folder",
      ...
    },
    {
      "id": "photo.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!131",
      ...
      "upload_location": "https://apis.live.net/v5.0/photo.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!131/content/",
      ...
      "type": "photo",
      ...
    },
    {
      "id": "file.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!119",
      ...
      "upload_location": "https://apis.live.net/v5.0/file.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!119/content/",
      ...
      "type": "file",
      ...
    }
  ]
}
```

[Top](#)

Object permissions

Users can control who can access certain objects in their SkyDrive storage location. This includes objects like folders, files, albums, photos, and videos.

Object permissions in SkyDrive are inherited from their parent objects all the way up through to a user's top-level folder in the SkyDrive directory.

When your code creates or updates an object in SkyDrive, your code can't programmatically set or change the object's permissions. Its permissions are automatically inherited. However, an application can allow sharing with a link to grant users read or read/write access to a file or folder. Of course, the user can change an object's permissions through the SkyDrive user interface.

Your code can't change an object's permissions in a way that would give it more restrictive permissions all the way up through a user's top-level folder in the SkyDrive directory. For example, if a certain folder allows anyone to access its contents, you can't change the permissions on any of the folder's contained objects to restrict who can access them.

[Top](#)

File concepts

To upload a file to SkyDrive, your code can use HTTP operations such as POST or PUT. You can also use MOVE and COPY to avoid downloading then uploading files that you want to move or copy.

By default, when your code uploads a file, and a file with the same name already exists in the same location in SkyDrive, the default behavior is for SkyDrive to overwrite the existing file. You can add an **Overwrite** header with a value of **false** to prevent the existing file from being overwritten.

[Top](#)

Supported file formats

SkyDrive currently supports the following file formats:

- .ARW
- .CR2
- .CRW
- .DOC
- .DOCM
- .DOCX
- .DOT
- .DOTM
- .DOTX
- .ERF
- .GIF
- .JPG
- .MEF
- .MP4
- .MRW
- .NEF
- .NRW
- .ONE
- .ORF
- .PDF
- .PEF
- .PNG
- .POT
- .POTM
- .POTX
- .PPS
- .PPSM
- .PPSX
- .PPT
- .PPTM

- .PPTX
- .RAW
- .RW2
- .RWL
- .SR2
- .TIF
- .TIFF
- .TXT
- .WMV
- .XLS
- .XLSB
- .XLSM
- .XLSX

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Folders and files

[This documentation is preliminary and is subject to change.]

Your apps can use the Live Connect APIs to create, read, update, and delete a SkyDrive user's folders and files.

Note In this context, a *file* represents a file of any supported format, including a photo or a video. A *folder* can contain other child folders as well as files. (An *album* is a special type of folder; for more info, see [Albums, photos, videos, and tags](#).) For a list of supported file formats, see the [SkyDrive core concepts](#) topic's "Supported file formats" section.

Your apps can also use the APIs to get a link to an uploaded file. You can then share this link with others so that they can access the file directly.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#). To request the required scopes, see the code patterns in [Obtaining user consent](#).

- [Reading folder properties](#)
- [Deleting folders](#)
- [Creating folders](#)
- [Updating folder properties](#)
- [Reading file properties](#)
- [Updating file properties](#)
- [Downloading files](#)
- [Uploading files](#)
- [Updating uploaded files](#)
- [Deleting uploaded files](#)
- [Getting links to folders and files](#)
- [Moving and copying folders and files](#)

Reading folder properties

To get info about a folder, use code like this. The **wl.skydrive** scope is required.

[REST]

```
GET http://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!114?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function readFolderProperties_onClick() {
    var scopes = [ "wl.skydrive" ];
    WL.login({ scope: scopes }, onWLLoginComplete)
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            folderid = "folder.22609f034aebe211.22609F034AEBE211!107"
            WL.api({ path: folderid, method: "GET" }, onReadFolderPropertiesComplete)
        }
    };
    function onReadFolderPropertiesComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error reading folder properties: " + result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "Folder properties: name = " + result.name + ", ID = " + result.id;
        }
    }
}
```

```

        }
    };
}

[C#]

private void btnGetFolderProperties_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.GetCompleted += new EventHandler<LiveOperationCompletedEventArgs>(getFolderProperties_Completed);
        client.GetAsync("folder.d6619aad0e87f473.D6619AAD0E87F473!707");
    }
}

void getFolderProperties_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        Dictionary<string, object> folder = (Dictionary<string, object>)e.Result;
        MessageBox.Show("Folder name= " + folder["name"].ToString() + "\n,id= " + folder["id"].ToString());
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the folder ID to this.

- A different folder ID to get info about another folder.
- *FOLDER_ID/files*, where *FOLDER_ID* represents a valid folder ID, to get a list of all items in a folder. Your code could then check to see if any these items has their **type** structure set to **folder**. Then your code could make another call using the desired folder ID to get that folder's info, too.

You can also do this.

- Get info about a user's top-level folder in the SkyDrive directory. To do this, replace the folder ID with *me/skydrive* (or *USER_ID/skydrive*).
- Get only certain types of items by using the *filter* parameter in the preceding code and specifying the item type: *all* (default), *photos*, *videos*, *folders*, or *albums*. For example, to get only photos, use *FOLDER_ID/files?filter=photos*.
- Get a limited number of items by using the *limit* parameter in the preceding code, specifying the number of items to get. For instance, to get the first two items, use *FOLDER_ID/files?limit=2*.
- Set the first item to start getting by using the *offset* parameter in the preceding code, specifying the index of the first item to get. For instance, to get two items starting at the third item, use *FOLDER_ID/files?limit=2&offset=3*.

Note In the JavaScript Object Notation (JSON)-formatted object that's returned, you can look in the **paging** object for the **previous** and **next** and structures, if they apply, to get the *offset* and *limit* parameter values of the previous and next entries, if they exist.

- Set the items' sort criteria by using the *sort_by* parameter in the preceding code and specifying the criteria: *created*, *updated*, *name*, *size*, or *default*. For example, to sort the items by name, use *FOLDER_ID/files?sort_by=name*.
- Set the items' sort order by using the *sort_order* parameter in the preceding code and

specifying the order: `ascending` or `descending`. For example, to sort the items in descending order, use `FOLDER_ID/files?sort_order=descending`.

[Top](#)

Deleting folders

To remove a folder, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```
DELETE http://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!114?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function deleteFolder_onClick() {
    var scopes_needed = ["wl.skydrive_update"];
    WL.login({ scope: scopes_needed }, onWLLoginComplete);
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            folder = "folder.22609f034aebe211.22609F034AEBE211!107"
            WL.api({ path: folder, method: "DELETE" }, onDeleteFolderComplete)
        }
    };
    function onDeleteFolderComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error deleting folder: " + result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "Deleted folder";
        }
    };
}
```

[C#]

```
private void btnDeleteFileOrFolder_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.DeleteCompleted += new EventHandler<LiveOperationCompletedEventArgs>(DeleteFileOrFolder_Completed);
        client.DeleteAsync("folder.22609f034aebe211.22609F034AEBE211!127");
    }
}

void DeleteFileOrFolder_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("File or folder deleted.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}
```

In the preceding code, change the folder ID to the folder ID of the folder you want to remove.

[Top](#)

Creating folders

To add a new folder, use code like this.

[REST]

```
POST https://apis.live.net/v5.0/me/skydrive  
Authorization: Bearer ACCESS_TOKEN  
Content-Type: application/json  
  
{  
    name: "My example folder"  
}
```

[JavaScript]

```
function createFolder_onClick() {  
    var scopes_needed = ["wl.skydrive_update"];  
    WL.login({ scope: scopes_needed }, onWLLoginComplete);  
    function onWLLoginComplete() {  
        var session = WL.getSession();  
        if (session.error) {  
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;  
        }  
        else {  
            var body = {  
                "name": "My new new folder",  
                "description": "A new folder"  
            };  
            parent_folder = "/me/skydrive";  
            WL.api({ path: parent_folder, method: "POST", body: body }, onCreateFolderComplete);  
        }  
    };  
    function onCreateFolderComplete(result) {  
        if (result.error) {  
            document.getElementById("infoLabel").innerText = "Error creating folder: " + result.error.code;  
        }  
        else {  
            document.getElementById("infoLabel").innerText = "Created folder. Name = " + result.name + ", ID = " + result.id;  
        }  
    };  
}
```

[C#]

```
private void btnCreateFolder_Begin(object sender, RoutedEventArgs e)  
{  
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)  
    {  
        MessageBox.Show("You must sign in first.");  
    }  
    else  
    {  
        Dictionary<string, object> folderData = new Dictionary<string, object>();  
        folderData.Add("name", "my new folder");  
        LiveConnectClient client = new LiveConnectClient(session);  
        client.PostCompleted += new EventHandler<LiveOperationCompletedEventArgs>(CreateFolder_Completed);  
        client.PostAsync("me/skydrive", folderData);  
    }  
}  
  
void CreateFolder_Completed(object sender, LiveOperationCompletedEventArgs e)  
{  
    if (e.Error == null)  
    {  
        MessageBox.Show("Folder created.");  
    }  
    else  
    {  
        MessageBox.Show(e.Error.Message);  
    }  
}
```

```
    }
}
```

In the preceding code, you can change `me/skydrive` to the folder ID of the folder into which the new folder will be created.

For details about the minimum required and optional structures that your app must provide when using POST, see the [REST API](#) topic's "Folder object" section.

[Top](#)

Updating folder properties

To change info for an existing folder, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```
PUT https://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!114
```

```
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    name: "My example folder has changed"
}
```

[JavaScript]

```
function updateFolderProperties_onClick() {
    var scopes_needed = ["wl.skydrive_update"];
    WL.login({ scope: scopes_needed }, onWLLoginComplete);
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            var body = {
                name: "My folder name"
            };
            folder = "folder.22609f034aebe211.22609F034AEBE211!107";
            WL.api({ path: folder, method: "PUT", body: body }, onUpdateFolderPropertiesComplete);
        }
    };
    function onUpdateFolderPropertiesComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error updating folder: " + result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "Updated folder. Name = " + result.name + ", ID = " + result.id;
        }
    };
}
```

[C#]

```
private void btnRenameFolder_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        Dictionary<string, object> folderData = new Dictionary<string, object>();
        folderData.Add("name", "my new folder name");
        LiveConnectClient client = new LiveConnectClient(session);
        client.PutCompleted += new EventHandler<LiveOperationCompletedEventArgs>(RenameFolder_Completed);
    }
}
```

```

        client.PutAsync("folder.d6619aad0e87f473.D6619AAD0E87F473!707", folderData);
    }

void RenameFolder_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Folder created.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the folder ID to the folder ID of the folder you want to change.

For details about the minimum required and optional structures that your app must provide when using PUT, see the [REST API](#) topic's "Folder object" section.

[Top](#)

Reading file properties

To get info about a file, photo, or video, use code like this. The **wl.skydrive** scope is required.

[REST]

```
GET http://apis.live.net/v5.0/file.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!126?access_token=ACCESS_TOKEN
```

[JavaScript]

```

function readFileProperties_onClick() {
    var scopes = ["wl.skydrive"];
    WL.login({ scope: scopes }, onWLLoginComplete)
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            file = "file.22609f034aebe211.22609F034AEBE211!105"
            WL.api({ path: file, method: "GET" }, onReadFilePropertiesComplete)
        }
    };
    function onReadFilePropertiesComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error reading file properties: " + result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "File properties: name = " + result.name + ", ID = " + result.id;
        }
    };
}

```

[C#]

```

private void btnGetFileProperties_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.GetCompleted += new EventHandler<LiveOperationCompletedEventArgs>(getFileProperties_Completed);
        client.GetAsync("file.d6619aad0e87f473.D6619AAD0E87F473!707");
    }
}

```

```

}

void getFileProperties_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        Dictionary<string, object> file = (Dictionary<string, object>)e.Result;
        MessageBox.Show("File name= " + file["name"].ToString() + "\n,id= " + file["id"].ToString());
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the file ID to a different file ID, photo ID, or video ID to get info about a different file, photo, or video.

[Top](#)

Updating file properties

To change info about a file, photo, or video, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```
PUT https://apis.live.net/v5.0/file.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!126
```

```
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    name: "MyNewFileName.doc"
}
```

[JavaScript]

```

function updateFileProperties_onClick() {
    var scopes_needed = ["wl.skydrive_update"];
    WL.login({ scope: scopes_needed }, onWLLoginComplete);
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            var body = {
                name: "MyNewFileName"
            };
            file = "file.22609f034aebe211.22609F034AEBE211!105"
            WL.api({ path: file, method: "PUT", body: body }, onUpdateFilePropertiesComplete)
        }
    };
    function onUpdateFilePropertiesComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error updating file properties: " + result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "Updated file properties. Name = " + result.name + ", ID = " + result.id;
        }
    };
}

```

[C#]

```

private void btnRenameFile_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {

```

```

        MessageBox.Show("You must sign in first.");
    }
    else
    {
        Dictionary<string, object> fileData = new Dictionary<string, object>();
        fileData.Add("name", "newfilename.doc");
        LiveConnectClient client = new LiveConnectClient(session);
        client.PutCompleted += new EventHandler<LiveOperationCompletedEventArgs>(RenameFile_Completed);
        client.PutAsync("file.d6619aad0e87f473.D6619AAD0E87F473!707", fileData);
    }
}

void RenameFile_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("File renamed");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the file ID to a different file ID, photo ID, or video ID to change info about a different file, photo, or video.

For details about the minimum required and optional structures that your app must provide when using PUT for a photo or video, see the [REST API](#) topic's "Photo object" or "Video object" section.

For details about the minimum required and optional structures that your app must provide when using PUT for any other file type, see the [REST API](#) topic's "File object" section.

[Top](#)

Downloading files

How to get a file's, photo's, or video's contents depends on your app type. The **wl.skydrive** scope is required.

[REST]

For Representational State Transfer (REST), use the REST code in the preceding [Reading file properties](#) section, but insert `/content` after the file ID, photo ID, or video ID, for example `FILE_ID/content`.

To get a download link to a file's, photo's, or video's contents, use the REST code in the preceding [Reading file properties](#) section, but insert `/content?suppress_redirects=true` after the file ID, photo ID, or video ID, for example `FILE_ID/content?suppress_redirects=true`.

[JavaScript]

For client-side websites and scripts that use JavaScript, use the JavaScript code in the preceding [Reading file properties](#) section, but insert `/content` after the file ID, photo ID, or video ID, for example `FILE_ID/content`.

To get a download link to a file's, photo's, or video's contents, use the JavaScript code in the preceding [Reading file properties](#) section, but insert `/content?suppress_redirects=true` after the file ID, photo ID, or video ID, for example `FILE_ID/content?suppress_redirects=true`.

For Metro style apps using JavaScript, use code like this.

```

function downloadFile_onClick() {
    try {
        var path = "file.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!126/picture";
        var picker = setupSavePicker();
        var onPickFile = function (file) {
            if (file && (file instanceof Windows.Storage.StorageFile)) {
                WL.download({
                    path: path,
                    file_output: file
                });
            }
        };
        picker.onFilePicked(onPickFile);
    }
}

```

```

        }, onDownloadCompleted);
    }
};

var onDownloadCompleted = function (response) {
    if (response) {
        var msg = response.message | response.error;
        document.getElementById("infoLabel").innerText = msg;
    }
    else {
        document.getElementById("infoLabel").innerText = "Download completed.";
    }
};

var filePickOp = picker.pickSaveFileAsync().then(onPickFile, onPickFile);
} catch (err) {
    document.getElementById("infoLabel").innerText = err;
}
}

function setupSavePicker() {
    var savepicker = new Windows.Storage.Pickers.FileSavePicker();
    savepicker.suggestedStartLocation = Windows.Storage.Pickers.PickerLocationId.documentsLibrary;
    savepicker.fileTypeChoices.insert("Picture", [".jpg"]);
    return savepicker;
}

```

[C#]

For C#, you use the **LiveConnectClient.DownloadAsync** method. However, the approach to getting a file's contents will vary based on your app type.

For Windows Phone apps, you could capture the file as a **System.IO.MemoryStream** object by reading the **LiveConnectClient.DownloadCompleted** event's **LiveOperationCompletedEventArgs.UserState** property and then displaying the contents of the **MemoryStream** object in the Windows Phone UI, like this.

```

private void btnDownloadFile_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        MemoryStream fileContent = new MemoryStream();
        LiveConnectClient client = new LiveConnectClient(session);
        client.DownloadCompleted += new EventHandler<LiveOperationCompletedEventArgs>(DownloadFile_Completed);
        client.DownloadAsync("file.d6619aad0e87f473.D6619AAD0E87F473!707", fileContent);
    }
}

void DownloadFile_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MemoryStream memoryStream = null;
        if (e.Error == null)
        {
            memoryStream = e.UserState as MemoryStream;
            if (memoryStream != null)
            {
                MessageBox.Show("File downloaded.");
                // Insert code to save or display image code.
            }
            else
            {
                MessageBox.Show("Error downloading file: " + e.Error.ToString());
            }
        }
        else
        {
            MessageBox.Show(e.Error.Message);
        }
    }
}

```

```
    }  
}
```

For Metro style apps using C#, you could capture the file as a **Windows.Storage.Streams.IRandomAccessStream** object by reading the **LiveConnectClient.DownloadCompleted** event's **LiveOperationCompletedEventArgs.UserState** property and then displaying the contents of the **IRandomAccessStream** object in the Windows Developer Preview UI, like this.

```
private void DownloadCompleted(object sender, LiveOperationCompletedEventArgs args)  
{  
    Windows.Storage.Streams.IRandomAccessStream fileStream = null;  
    if (args.Error == null)  
    {  
        fileStream = args.UserState as Windows.Storage.Streams.IRandomAccessStream;  
        if (fileStream != null)  
        {  
            var stream = args.UserState as Windows.Storage.Streams.IRandomAccessStream;  
            if (stream != null)  
            {  
                this.imageFrame.Visibility = Windows.UI.Xaml.Visibility.Visible;  
  
                Windows.UI.Xaml.Media.Imaging.BitmapImage imgSource =  
                    new Windows.UI.Xaml.Media.Imaging.BitmapImage();  
                imgSource.SetSource(stream);  
                this.imageFrame.Source = imgSource;  
            }  
        }  
    }  
    else  
    {  
        this.infoTextBlock.Text =  
            "Error downloading file: " + args.Error.ToString();  
    }  
}
```

[Top](#)

Uploading files

How to add a file's, photo's or video's contents depends on your app type. The **wl.skydrive_update** scope is required.

[REST]

For REST, you can use PUT or POST, depending on your platform's recommended approach.

Here's an example of some code that uses PUT.

```
PUT https://apis.live.net/v5.0/me/skydrive/files/HelloWorld.txt?access_token=ACCESS_TOKEN  
Hello, World!
```

Note By default the file, photo, or video will be overwritten if it already exists. You can add an **Overwrite** header with a value of **false** to not overwrite the file, photo, or video if it already exists.

Here's an example of some code that uses POST.

```
POST https://apis.live.net/v5.0/me/skydrive/files?access_token=ACCESS_TOKEN  
Content-Type: multipart/form-data; boundary=A300x  
  
--A300x  
Content-Disposition: form-data; name="file"; filename="HelloWorld.txt"  
Content-Type: application/octet-stream  
  
Hello, World!  
--A300x
```

For REST, if you're using POST, as shown here you must format the request as multipart/form-data. You provide the file to upload in a multipart section, while you provide the file name in the

Content-Disposition header's *filename* parameter. For more info, see [RFC 2388](#).

Also for REST, if you're using POST, change `me/skydrive/files` to one of these.

- The folder ID of the folder into which the file, photo, or video will be uploaded.
- The value of the **upload_link** structure for the folder into which the file, photo, or video will be uploaded.

Note By default the file, photo, or video will be overwritten if it already exists. You can add an **Overwrite** header with a value of **false** to not overwrite the file, photo, or video if it already exists.

[JavaScript]

For JavaScript, the Live Connect APIs don't support uploading files for client-side websites and scripts with JavaScript. This code sample works only for Metro style apps using JavaScript.

```
function uploadFile_onClick() {
    try {
        var picker = setupOpenPicker();
        var onUploadCompleted = function (response) {
            if (response) {
                var msg = response.message | response.error;
                document.getElementById("infoLabel").innerText = msg;
            }
            else {
                document.getElementById("infoLabel").innerText = "Upload completed.";
            }
        };
        var onPickFile = function (file) {
            if (file && (file instanceof Windows.Storage.StorageFile)) {
                WL.upload({
                    path: "me/skydrive/files",
                    file_name: file.fileName,
                    file_input: file
                }, onUploadCompleted);
            }
        };
        var filePickOp = picker.pickSingleFileAsync().then(onPickFile, onPickFile);
    } catch (err) {
        document.getElementById("infoLabel").innerText = err;
    }
}

function setupOpenPicker() {
    var openpicker = new Windows.Storage.Pickers.FileOpenPicker();
    openpicker.fileTypeFilter.replaceAll(["*"]);
    return openpicker;
}
```

[C#]

For C#, you use the **LiveConnectClient.UploadAsync** method. However, the approach to specifying a file to upload will vary based on your app type.

For Windows Phone apps, you could use the **Microsoft.Phone.Tasks.PhotoChooserTask** class to capture info about a picture that a user selects from the phone's **Pictures** app, like this.

```
private void createFile_Click(object sender, RoutedEventArgs e)
{
    var picker = new PhotoChooserTask();
    picker.Completed += OnPhotoChooserTaskCompleted;
    picker.Show();
}

private void OnPhotoChooserTaskCompleted(object sender, PhotoResult e)
{
    if (e.Error != null)
    {
        this.infoTextBlock.Text = "Error choosing photo: " + e.Error.ToString();
    }
    else
```

```

    {
        string[] filePathSegments = e.OriginalFileName.Split('\\');
        string fileName = filePathSegments[filePathSegments.Length - 1];
        var scopes = new List<string>(1);
        this.client.UploadCompleted
            += new EventHandler<LiveOperationCompletedEventArgs>(OnPhotoChooserTaskCompleted_UploadCompleted);
        this.client.UploadAsync("me/skydrive/files", fileName, e.ChosenPhoto);
    }
}

private void OnPhotoChooserTaskCompleted_UploadCompleted(object sender, LiveOperationCompletedEventArgs args)
{
    if (args.Error == null)
    {
        Dictionary<string, object> file = (Dictionary<string, object>)args.Result;
        this.infoTextBlock.Text = "File uploaded. Link = " + file["link"];
    }
    else
    {
        this.infoTextBlock.Text =
            "Error uploading file: " + args.Error.ToString();
    }
}

```

For Metro style apps using C#, you could use the **Windows.Storage.Pickers.FileOpenPicker** class to capture info about a picture that a user selects, like this.

```

private void createFile_Click(object sender, RoutedEventArgs e)
{
    string folderPath = "me/skydrive/files";
    var picker = new Windows.Storage.Pickers.FileOpenPicker();
    picker.ViewMode = Windows.Storage.Pickers.PickerViewMode.Thumbnail;
    picker.SuggestedStartLocation = Windows.Storage.Pickers.PickerLocationId.PicturesLibrary;
    picker.FileTypeFilter.Add("*");
    picker.PickSingleFileAsync().StartAsTask().ContinueWith(
        task => this.OnSelectFile(folderPath, task.Result));
}

private void OnSelectFile(string folderPath, Windows.Storage.StorageFile file)
{
    if (file != null)
    {
        file.OpenForReadAsync().StartAsTask().ContinueWith(
            task => this.client.UploadAsync(folderPath, file.Name, task.ResultAsStream()));
    }
}

private void UploadCompleted(object sender, LiveOperationCompletedEventArgs args)
{
    if (args.Error == null)
    {
        System.Collections.Generic.Dictionary<string, object> file =
            (System.Collections.Generic.Dictionary<string, object>)args.Result;
        infoTextBlock.Text = "File uploaded. Link = " + file["link"];
    }
    else
    {
        this.infoTextBlock.Text =
            "Error uploading file: " + args.Error.ToString();
    }
}

```

If you're using any approach other than POST for REST, change `me/skydrive/files` to one of these.

- The value of the **upload_link** structure for the folder into which the file, photo, or video will be uploaded, followed by a forward slash character (/) and the desired file name.
- The folder ID of the folder into which the file, photo, or video will be uploaded, followed by /files/ and the desired file name.
- If the file already exists from a previous upload operation, the value of the existing file's

upload_link structure.

- If the file already exists from a previous upload operation, the file ID of the file, followed by /content.

For details about the minimum required and optional structures that your app must provide when using POST or PUT to upload a photo or video, see the [REST API](#) topic's "Photo object" or "Video object" section.

For details about the minimum required and optional structures that your app must provide when using POST or PUT to upload any other file type, see the [REST API](#) topic's "File object" section.

[Top](#)

Updating uploaded files

To change an existing file's, photo's, or video's contents, use the code in the preceding [Uploading files](#) section; and use POST, PUT, or **LiveConnectClient.UploadAsync**, depending on your app type's supported approach.

[Top](#)

Deleting uploaded files

To remove a file, photo, or video, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```
DELETE http://apis.live.net/v5.0/file.b7c3b8f9g3616f6f.B7CB8F9G3626F6!225?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function deleteFile_onClick() {
    var scopes_needed = ["wl.skydrive_update"];
    WL.login({ scope: scopes_needed }, onWLLoginComplete);
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            file = "file.22609f034aebe211.22609F034AEBE211!135"
            WL.api({ path: file, method: "DELETE" }, onDeleteFileComplete)
        }
    };
    function onDeleteFileComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error deleting file: " + result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "Deleted file.";
        }
    };
}
```

[C#]

```
private void btnDeleteFileOrFolder_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.DeleteCompleted += new EventHandler<LiveOperationCompletedEventArgs>(DeleteFileOrFolder_Completed);
        client.DeleteAsync("folder.22609f034aebe211.22609F034AEBE211!127");
    }
}
```

```

}

void DeleteFileOrFolder_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("File or folder deleted.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the file ID to a different file ID, photo ID, or video ID to remove a different file, photo, or video.

[Top](#)

Getting links to folders and files

Your apps can get four types of links to SkyDrive folders and files that are publicly shared.

- An *embedded link*, which is an HTML code snippet that you can then insert into a webpage to provide an interactive view of the corresponding file.
Note Embedded links are supported only for publicly-shared Microsoft PowerPoint and Microsoft Excel files.
- A *read-only* link, which is a link to a read-only version of the folder or file.
- A *read-write* link, which is a link to a read-write version of the folder or file.

To get a link to a folder or file, use code like this.

[REST]

```
GET http://apis.live.net/v5.0/file.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!126/shared_read_link?access_token=ACCESS_TOKEN
```

[JavaScript]

```

function getSharedLink_onClick() {
    var scopes = ["wl.skydrive"];
    WL.login({ scope: scopes }, onWlLoginComplete)
    function onWlLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            itemid = "file.22609f034aebe211.22609F034AEBE211!105"
            path = itemid + "/shared_read_link"
            WL.api({ path: path, method: "GET" }, onGetSharedLinkComplete)
        }
    };
    function onGetSharedLinkComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Could not get a shared link: " + result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "Shared link = " + result.link;
        }
    };
}

```

[C#]

```

private void btnGetSharedLink_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {

```

```

        MessageBox.Show("You need to Sign In");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.GetCompleted += new EventHandler<LiveOperationCompletedEventArgs>(GetSharedLink_Completed);
        client.GetAsync("file.22609f034aebe211.22609F034AEBE211!115/shared_read_link");
    }
}

void GetSharedLink_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        Dictionary<string, object> shared = (Dictionary<string, object>)e.Result;
        MessageBox.Show("Shared link= " + shared["link"].ToString());
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the file ID to the folder ID or file ID that you want a link to. Also, change `shared_read_link` to one of these.

- To `embed` for an embedded link. You get the embed HTML code snippet from the returned JSON-formatted object's **embed_html** structure's value.
- To `shared_read_link` for a read-only link. You get the read-only link from the returned JSON-formatted object's **link** structure's value.
- To `shared_edit_link` for a read-write link. You get the read-write link from the returned JSON-formatted object's **link** structure's value.

When your app requests a file, photo, or video, the returned object includes a pre-authenticated URL as the value of the `source` attribute. Pre-authenticated URLs allow anybody to link directly to a file without the need for additional permissions. This example shows what a pre-authenticated URL looks like.

```
"source": "http://storage.live.com/s1pk0Hsk4twGIBzleNLCqhVhp5yvj9aPLPMHq2GDTRFBeuicbS27Fx5xBnmyy5J41DSEtdvYm2FxOB0-s8Heb1l6hg3A1E2bJ9qWQ3GdIUTnnG6CHLTBLdCHRPUAunekX1dnLS49vG-fUV708ZvrkiLTubXXFqrT1roxZ1oJjELE/Test.txt:Binary"
```

[Top](#)

Moving and copying folders and files

To move a folder, file, photo, or video, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```
MOVE https://apis.live.net/v5.0/file.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!126

Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json

{
    destination: "folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!114"
}
```

[JavaScript]

```
function moveFile_onClick() {
    var scopes = ["wl.skydrive_update"];
    WL.login({ scope: scopes }, onWLLoginComplete)
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.err
```

```

        or;
    }
    else {
        var item = "file.22609f034aebe211.22609F034AEBE211!105"
        var body = {
            destination: "folder.22609f034aebe211.22609F034AEBE211!116"
        };
        WL.api({ path: item, method: "MOVE", body: body }, onMoveFileComplete)
    }
};

function onMoveFileComplete(result) {
    if (result.error) {
        document.getElementById("infoLabel").innerText = "Error moving file or folder: " +
    result.error.code;
    }
    else {
        document.getElementById("infoLabel").innerText = "Item moved.";
    }
}
}

[C#]

private void btnMoveFile_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.MoveCompleted += new EventHandler<LiveOperationCompletedEventArgs>(MoveFile_Move
Completed);
        client.MoveAsync("file.22609f034aebe211.22609F034AEBE211!105", "folder.22609f034aebe211
.22609F034AEBE211!116");
    }
}

void MoveFile_MoveCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("File or folder move completed.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change this.

- For REST and JavaScript, change MOVE to COPY to copy the item. For C#, use a **CopyCompleted** event handler and the **CopyAsnyc** method.
- Change the file ID to the file ID or folder ID of the file or folder that you want to move or copy.
- Change the folder ID to the folder ID of the target folder that you want to move or copy the source file or folder to.

Important The destination of a move or copy operation must be a folder. Also, SkyDrive storage is structured so that move and copy operations cannot occur between the SkyDrive folder hierarchies of different users. For example, even if user A can read user B's files, user A cannot copy or move them into his or her own SkyDrive folders.

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Albums, photos, videos, and tags

[This documentation is preliminary and is subject to change.]

Your apps can use the Live Connect APIs to create, read, update, and delete a SkyDrive user's *albums* (a special type of folder that can store photos and videos as well as child albums and child folders), photos, and videos. Your apps can also read *tags* (a type of comment) that users can add to photos and videos.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#). To request the required scopes, see the code patterns in [Obtaining user consent](#).

- [Reading albums](#)
- [Deleting albums](#)
- [Creating albums](#)
- [Updating albums](#)
- [Reading photo and video properties and their contents](#)
- [Updating photo and video properties](#)
- [Uploading, updating, and deleting photos and videos](#)
- [Reading tags](#)

Reading albums

To get info about an album, use code like this. The **wl.photos** or **wl.skydrive** scope is required.

[REST]

```
GET http://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!111?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function readAlbumProperties_onClick() {
    var scopes = ["wl.photos"]; // Or wl.skydrive.
    WL.login({ scope: scopes }, onWLLoginComplete)
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        } else {
            album = "folder.22609f034aebe211.22609F034AEBE211!107"
            WL.api({ path: album, method: "GET" }, onReadAlbumPropertiesComplete)
        }
    };
    function onReadAlbumPropertiesComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error reading album properties: " + result.error.code;
        } else {
            document.getElementById("infoLabel").innerText = "Album properties: name = " + result.name + ", ID = " + result.id;
        }
    };
}
```

[C#]

```
private void btnGetAlbumProperties_Click(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.GetCompleted += new EventHandler<LiveOperationCompletedEventArgs>(getAlbumProperties
```

```

        _GetCompleted);
        client.GetAsync("album.d6619aad0e87f473.D6619AAD0E87F473!719");
    }
}

void getAlbumProperties_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        Dictionary<string, object> album = (Dictionary<string, object>)e.Result;
        MessageBox.Show("Album name:" + album["name"].ToString() + ", album id= " + album["id"].ToString());
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the album ID to this.

- A different album ID to get info about another album.
- `me/albums` to get info about all of the signed-in user's albums.
- `USER_ID/albums` to get info about all of a user's albums corresponding to a valid `USER_ID`.

You can also do this.

- `ALBUM_ID/photos` or `ALBUM_ID/videos`, to get a list of all photos or videos in an album. Then your code could make another call using the desired photo ID or video ID to get the photo's or video's info.
- `ALBUM_ID/files` to get a list of all photos, videos, child albums, and child folders in an album. Then your code could make another call using the desired photo ID, video ID, album ID, or folder ID to get the photo's, video's, album's, or folder's info.

Note You can get info about an album's photos, videos, child albums, and child folders only.

- `ALBUM_ID/picture` to get a picture of an album's cover.
 - Get a limited number of items by using the `limit` parameter in the preceding code, specifying the number of items to get. For instance, to get the first two items, use `ALBUM_ID/files?limit=2`.
 - Set the first item to start getting by using the `offset` parameter in the preceding code, specifying the index of the first item to get. For instance, to get two items starting at the third item, use `ALBUM_ID/files?limit=2&offset=3`.
- Note** In the JavaScript Object Notation (JSON)-formatted object that's returned, you can look in the **paging** object for the **previous** and **next** and structures, if they apply, to get the `offset` and `limit` parameter values of the previous and next entries, if they exist.
- Set the items' sort criteria by using the `sort_by` parameter in the preceding code and specifying the criteria: `created`, `updated`, `name`, `size`, or `default`. For example, to sort the items by name, use `ALBUM_ID/files?sort_by=name`.
 - Set the items' sort order by using the `sort_order` parameter in the preceding code and specifying the order: `ascending` or `descending`. For example, to sort the items in descending order, use `ALBUM_ID/files?sort_by=descending`.

[Top](#)

Deleting albums

To remove an album, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```
DELETE http://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!111?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function deleteAlbum_onClick() {
```

```

var scopes_needed = ["wl.skydrive_update"];
WL.Login({ scope: scopes_needed }, onWLLoginComplete);
function onWLLoginComplete() {
    var session = WL.getSession();
    if (session.error) {
        document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
    }
    else {
        album = "album.22609f034aebe211.22609F034AEBE211!135"
        WL.api({ path: album, method: "DELETE" }, onDeleteAlbumComplete)
    }
};
function onDeleteAlbumComplete(result) {
    if (result.error) {
        document.getElementById("infoLabel").innerText = "Error deleting album: " + result.error;
    }
    else {
        document.getElementById("infoLabel").innerText = "Deleted album.";
    }
};

```

[C#]

```

private void btnDeleteAlbum_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.DeleteCompleted += new EventHandler<LiveOperationCompletedEventArgs>(DeleteAlbum_Completed);
        client.DeleteAsync("album.d6619aad0e87f473.D6619AAD0E87F473!707");
    }
}

void DeleteAlbum_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Album deleted.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the album ID to the album ID of the album that you want to remove.

[Top](#)

Creating albums

To add a new album, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```

POST https://apis.live.net/v5.0/me/albums

Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json

{
    name: "My example album"
}

```

[JavaScript]

```

function createAlbum_onClick() {
    var scopes_needed = ["wl.skydrive_update"];
    WL.login({ scope: scopes_needed }, onWLLoginComplete);
    function onWLLoginComplete() {
        var session = WL.getSession();
    }
};

```

```

        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            var body = {
                "name": "My album",
                "description": "A new photo album"
            }
            WL.api({ path: "/me/albums", method: "POST", body: body }, onCreateAlbumComplete)
        }
    };
    function onCreateAlbumComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error creating album: " + result.error;
        }
        else {
            document.getElementById("infoLabel").innerText = "Created album. Name = " + result.name +
            ", ID = " + result.id;
        }
    };
}

```

[C#]

```

private void btnCreateAlbum_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        Dictionary<string, object> fileData = new Dictionary<string, object>();
        fileData.Add("name", "New Album");
        LiveConnectClient client = new LiveConnectClient(session);
        client.PostCompleted += new EventHandler<LiveOperationCompletedEventArgs>(CreateAlbum_Completed);
        client.PostAsync("me/albums", fileData);
    }
}

void CreateAlbum_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Album created.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

Note You can only create albums in the user's top-level SkyDrive folder, represented here as `me/albums`.

For details about the minimum required and optional structures that your app must provide when using POST, see the [REST API](#) topic's "Album object" section.

[Top](#)

Updating albums

To change info for an existing album, use code like this. The **wl.skydrive_update** scope is required.

[REST]

```

PUT https://apis.live.net/v5.0/folder.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!111

Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json

{
    name: "My example album has changed"
}

```

[JavaScript]

```
function updateAlbumProperties_onClick() {
    var scopes_needed = ["wl.skydrive_update"];
    WL.login({ scope: scopes_needed }, onWLLoginComplete);
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            var body = {
                name: "My sample album new name"
            };
            album = "album.22609f034aebe211.22609F034AEBE211!129";
            WL.api({ path: album, method: "PUT", body: body }, onUpdateAlbumPropertiesComplete)
        }
    };
    function onUpdateAlbumPropertiesComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error updating album: " + result.error;
        }
        else {
            document.getElementById("infoLabel").innerText = "Updated album. Name = " + result.name
            + ", ID = " + result.id;
        }
    };
}
```

[C#]

```
private void btnUpdateAlbum_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        Dictionary<string, object> folderData = new Dictionary<string, object>();
        folderData.Add("name", "my new album name");
        LiveConnectClient client = new LiveConnectClient(session);
        client.PutCompleted += new EventHandler<LiveOperationCompletedEventArgs>(UpdateAlbum_Completed);
        client.PutAsync("album.d6619aad0e87f473.D6619AAD0E87F473!707", folderData);
    }
}

void UpdateAlbum_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Album updated.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}
```

For details about the minimum required and optional structures that your app must provide when using PUT, see the [REST API](#) topic's "Album object" section.

[Top](#)

Reading photo and video properties and their contents

To get info about a photo or video, use code like this. The **wl.skydrive** scope is required.

[REST]

```
GET http://apis.live.net/v5.0/photo.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!112?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function readPhotoProperties_onClick() {
```

```

var scopes = ["wl.skydrive"];
WL.Login({ scope: scopes }, onWLLoginComplete)
function onWLLoginComplete() {
    var session = WL.getSession();
    if (session.error) {
        document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
    }
    else {
        photo = "file.22609f034aebe211.22609F034AEBE211!125"
        WL.api({ path: photo, method: "GET" }, onReadPhotoPropertiesComplete)
    }
};
function onReadPhotoPropertiesComplete(result) {
    if (result.error) {
        document.getElementById("infoLabel").innerText = "Error reading photo properties: " + r
    result.error.code;
    }
    else {
        document.getElementById("infoLabel").innerText = "Photo properties: name = " + result.n
ame + ", ID = " + result.id;
    }
};
}

```

[C#]

```

private void btnGetPhotoImage_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        MemoryStream photoImage = new MemoryStream();
        LiveConnectClient client = new LiveConnectClient(session);
        client.DownloadCompleted += new EventHandler<LiveOperationCompletedEventArgs>(GetPhotoImage
_DownloadCompleted);
        client.DownloadAsync("photo.d6619aad0e87f473.D6619AAD0E87F473!707/picture?type=thumbnail",
photoImage);
    }
}

void GetPhotoImage_DownloadCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MemoryStream memoryStream = null;
        if (e.Error == null)
        {
            memoryStream = e.UserState as MemoryStream;
            if (memoryStream != null)
            {
                BitmapImage imgSource = new BitmapImage();
                imgSource.SetSource(memoryStream);
                MessageBox.Show("Photo image downloaded.");
                // Insert code to save or display image code.
            }
        }
        else
        {
            MessageBox.Show("Error downloading file: " + e.Error.ToString());
        }
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code the photo ID to a different photo ID or video ID to get info about another photo or video.

You can also do this.

- Get a picture of a photo or video by using `/picture` along with the optional `type` parameter in the preceding code along one of these values: `small`, `normal` (default if the `type` parameter is not

specified), album, thumbnail, or full. For instance, to get a photo's thumbnail picture, use `PHOTO_ID/picture?type=thumbnail`.

- Get a video's contents by using `VIDEO_ID/video`.

[Top](#)

Updating photo and video properties

To change info about a photo or video, use code like this. The `wl.skydrive_update` scope is required.

[REST]

```
PUT https://apis.live.net/v5.0/photo.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!112
```

```
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    name: "My example photo has changed"
}
```

[JavaScript]

```
function updatePhotoProperties_onClick() {
    var scopes_needed = ["wl.skydrive_update"];
    WL.login({ scope: scopes_needed }, onWLLoginComplete);

    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            var body = {
                name: "My new photo"
            };
            photo = "file.22609f034aebe211.22609F034AEBE211!125"
            WL.api({ path: photo, method: "PUT", body: body }, onUpdatePhotoPropertiesComplete)
        }
    };
    function onUpdatePhotoPropertiesComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error updating photo properties: " +
            result.error.code;
        }
        else {
            document.getElementById("infoLabel").innerText = "Updated photo properties. Name = " +
            result.name + ", ID = " + result.id;
        }
    };
}
```

[C#]

```
private void btnUpdatePhoto_Begin(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        Dictionary<string, object> fileData = new Dictionary<string, object>();
        fileData.Add("name", "Renamed Photo");
        LiveConnectClient client = new LiveConnectClient(session);
        client.PutCompleted += new EventHandler<LiveOperationCompletedEventArgs>(UpdatePhoto_PutCompleted);
        client.PutAsync("photo.d6619aad0e87f473.D6619AAD0E87F473!707", fileData);
    }
}

void UpdatePhoto_PutCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("File renamed.");
```

```

    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the photo ID here to the photo ID or video ID of the photo or video for which you want to update the properties.

For details about the minimum required and optional structures that your app must provide when using PUT, see the [REST API](#) topic's "Photo object" or "Video object" section.

[Top](#)

Uploading, updating, and deleting photos and videos

The Live Connect APIs consider a photo or a video to be the same as a file when uploading, updating, and deleting a photo's or video's contents.

To add a photo's or video's contents, see the [Folders and files](#) topic's "Uploading files" section.

To update a photo's or video's contents, see the [Folders and files](#) topic's "Updating uploaded files" section.

To update a photo's or video's contents, see the [Folders and files](#) topic's "Deleting uploaded files" section.

[Top](#)

Reading tags

To get info about tags, use code like this. The **wl.skydrive** scope is required.

[REST]

```
GET http://apis.live.net/v5.0/photo.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!112/tags?access_token=ACCESS_TOKEN
```

[JavaScript]

```

function enumeratePhotoTags_onClick() {
    var scopes = ["wl.skydrive"];
    WL.login({ scope: scopes }, onWLLoginComplete)
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            photoid = "file.22609f034aebe211.22609F034AEBE211!114"
            path = photoid + "/tags"
            WL.api({ path: path, method: "GET" }, onEnumeratePhotoTagsComplete)
        }
    };
    function onEnumeratePhotoTagsComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error enumerating photo tags: " + result.error.code;
        }
        else {
            msg = "People tagged in the photo: ";
            items = result.data
            for (index in items) {
                if (index != 0)
                    msg += ", "
                msg += items[index].name
            }
            document.getElementById("infoLabel").innerText = msg;
        }
    };
}

```

[C#]

```

private void btnGetPhotoTags_Click(object sender, RoutedEventArgs e)
{

```

```

if (session == null || session.Status != LiveConnectSessionStatus.Connected)
{
    MessageBox.Show("You must sign in first.");
}
else
{
    LiveConnectClient client = new LiveConnectClient(session);
    client.GetCompleted += new EventHandler<LiveOperationCompletedEventArgs>(getPhotoTags_Completed);
    client.GetAsync("file.22609f034aebe211.22609F034AEBE211!112/tags");
}
}

void getPhotoTags_Completed(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        String tags = null;
        List<object> data = (List<object>)e.Result["data"];

        foreach (Dictionary<string, object> datum in data)
        {
            if (tags != null)
                tags += "\n";
            tags += "\t" + datum["name"].ToString() + "\n";
        }
        MessageBox.Show("People tagged:\n" + tags);
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}

```

In the preceding code, change the photo ID to a different photo ID or video ID to get info about another photo's or video's tags.

You can also do this.

- Get a limited number of tags by using the *limit* parameter in the preceding code, specifying the number of items to get. For instance, to get the first two tags, use *PHOTO_ID/tags?limit=2*.
 - Set the first tag to start getting by using the *offset* parameter in the preceding code, specifying the index of the first item to get. For instance, to get two items starting at the third item, use *PHOTO_ID/tags?limit=2&offset=3*.
- Note** In the JSON-formatted object that's returned, you can look in the **paging** object for the **previous** and **next** and structures, if they apply, to get the *offset* and *limit* parameter values of the previous and next entries, if they exist.

[Top](#)

Comments

[This documentation is preliminary and is subject to change.]

Your apps can use the Live Connect APIs to read comments that SkyDrive users can add to folders, files, albums, photos, and videos.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#). To request the required scopes, see the code patterns in [Obtaining user consent](#).

To get info about comments, use code like this. The **wl.photos** or **wl.skydrive** scopes are required.

[REST]

```
GET http://apis.live.net/v5.0/photo.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!112/comments?access_token=ACCESS_TOKEN
```

[JavaScript]

```
function enumerateItemComments_onClick() {
    var scopes = ["wl.skydrive"];
    WL.login({ scope: scopes }, onWLLoginComplete)
    function onWLLoginComplete() {
        var session = WL.getSession();
        if (session.error) {
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;
        }
        else {
            itemid = "file.22609f034aebe211.22609F034AEBE211!114"
            path = itemid + "/comments"
            WL.api({ path: path, method: "GET" }, onEnumerateItemCommentsComplete)
        }
    };
    function onEnumerateItemCommentsComplete(result) {
        if (result.error) {
            document.getElementById("infoLabel").innerText = "Error enumerating comments: " + result.error.code;
        }
        else {
            msg = "Comments: ";
            items = result.data
            for (index in items) {
                msg += "\n" + items[index].message + "(" + items[index].from.name + ")"
            }
            document.getElementById("infoLabel").innerText = msg;
        }
    };
}
```

[C#]

```
private void readComments_Click(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        LiveConnectClient client = new LiveConnectClient(session);
        client.GetCompleted += new EventHandler<LiveOperationCompletedEventArgs>(ReadComments_GetCompleted);
        client.GetAsync("photo.a6b2a7e8f2515e5e.A6B2A7E8F2515E5E!112/comments");
    }
}

void ReadComments_GetCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        String comments = null;
        List<object> data = (List<object>)e.Result["data"];

        foreach (Dictionary<string, object> datum in data)
        {
            if (comments != null)
                comments += "\n";
            comments += "\t" + datum["message"].ToString() + "\n";
        }
    }
}
```

```
        MessageBox.Show("Comments:\n" + comments);
    }
    else
    {
        MessageBox.Show(e.Error.Message);
    }
}
```

In the preceding code, change the photo ID to a different folder ID, file ID, album ID, photo ID, or video ID to get info about another folder's, file's, album's, photo's, or video's comments.

You can also do this.

- Get a limited number of comments by using the *limit* parameter in the preceding code, specifying the number of items to get. For instance, to get the first two comments, use *PHOTO_ID/comments?limit=2*.
- Set the first comments to start getting by using the *offset* parameter in the preceding code, specifying the index of the first item to get. For instance, to get two items starting at the third item, use *PHOTO_ID/comments?limit=2&offset=3*.

Note In the JavaScript Object Notation (JSON)-formatted object that's returned, you can look in the **paging** object for the **previous** and **next** and structures, if they apply, to get the *offset* and *limit* parameter values of the previous and next entries, if they exist.

© 2011 Microsoft Corporation. All rights reserved.

Messenger (instant messaging)

[This documentation is preliminary and is subject to change.]

You can use Extensible Messaging and Presence Protocol (XMPP), an open standard for real-time communication, to enable your apps to interact with a Windows Live Messenger user and his or her buddies.

- [Getting started using Messenger with XMPP](#)

Get a user's okay for your app to act on his or her behalf, and then sign in to the Messenger XMPP service.

- [XMPP code examples](#)

Exchange instant messages and get a user's info and his or her buddies' info, including their friendly names, display tiles, presence info, and status messages. Also, update a user's status message for the duration of a session.

- [Sharing a user's status](#)

Update a user's status message between sessions by using the Live Connect APIs.

© 2011 Microsoft Corporation. All rights reserved.

Getting started using Messenger with XMPP

[This documentation is preliminary and is subject to change.]

Before your client app can start using Extensible Messaging and Presence Protocol (XMPP) to sign in to the Windows Live Messenger XMPP service and interact with a Messenger user and his or her buddies, there are several things you must do first.

- [Prerequisites](#)
- [Prepare to sign in](#)
- [Sign in](#)
- [Get a client ID](#)
- [Get an access token](#)
- [Next steps](#)

Prerequisites

Before you prepare to sign in your app to the Messenger XMPP service, you'll need to know this.

- Live Connect doesn't provide any XMPP libraries. If you don't have a preferred XMPP library to use, you may want to explore the list of available XMPP libraries at [XMPP Standards Foundation - Libraries](#).
- For the XMPP extensions and features that we support, see [Messenger XMPP reference](#). Your app must support XMPP: Core functionality and may optionally implement the XMPP extensions we support.

[Top](#)

Prepare to sign in

Before your app can sign in to the Messenger XMPP service, you'll need this info.

- To connect to the Messenger XMPP service, either use Domain Name System (DNS) SRV records for messenger.live.com, or connect directly to xmpp.messenger.live.com on port 5222.
- Secure Sockets Layer (SSL) is required.
- We use X-MESSENGER-OAUTH2 Simple Authentication and Security Layer (SASL) authentication to get a user's okay for permission, or *consent*, for your app to represent him or her when signing in to the Messenger XMPP service. To do this, we use a custom implementation of the [OAuth 2.0](#) standard. To use our implementation, your app needs a *client ID* and an *access token* (and possibly other info, depending on your app type). To get these, see the [Get a client ID](#) and [Get an access token](#) sections later in this topic.

[Top](#)

Sign in

Once you have an access token, your app can sign in to the Messenger XMPP service on behalf of the consenting user. (To get an access token, see the [Get an access token](#) section later in this topic.)

As stated in the previous section, the Messenger XMPP service uses X-MESSENGER-OAUTH2 SASL authentication. Your app sends the access token as the SASL token for authentication, as shown in this partial exchange of protocol info, where *ACCESS_TOKEN* represents your access token.

```
...
<stream:features xmlns:stream="http://etherx.jabber.org/streams"><mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl"><mechanism>X-ME<
> <auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanism="X-MESSENGER-OAUTH2">ACCESS_TOKEN</auth>
<success xmlns="urn:ietf:params:xml:ns:xmpp-sasl" />
...
```

Remember that your code must first connect to the Messenger XMPP service by either using DNS SRV records for messenger.live.com or connecting directly to xmpp.messenger.live.com on port 5222.

Note The Messenger XMPP service uses the user ID within the access token to both sign in the user and to bind the user to the session. To do this, Jabber IDs (JIDs) assigned by the Messenger XMPP service use the format *identifier@messenger.live.com* instead of a Microsoft account user ID.

[Top](#)

Get a client ID

Before your app can attempt to sign in to the Messenger XMPP service, you first need a *client ID*. A client ID is a unique code that identifies your app to the Live Connect authentication service and is exchanged for an *access token*, which is explained in the next section. To get a client ID, follow the instructions in [Getting a client ID and configuring your app](#). (Depending on your app type, you may also need to get a *client secret* and specify a *redirect domain*. Instructions for how to get these are also in that topic.)

[Top](#)

Get an access token

Once you have a client ID (and possibly other info, depending on your app type), your app needs to give this info to the Live Connect authentication service in exchange for an *access token*. This access token allows your app to sign in to the Messenger XMPP service for a specific user who has okayed for permission, or *consented*, for your app to act on his or her behalf. (To get a client ID, see the [Get a client ID](#) section earlier in this topic.)

To get an access token, you'll need to use the client ID along with specifying a user consent level, which we call a *scope*. The specific scope that you'll need to use here is **wl.messenger**.

To use the **wl.messenger** scope along with your client ID to get an access token, follow the guidance and code examples in [Signing users in](#) and [Obtaining user consent](#). For additional guidance and more complete code samples, go here depending on your app type.

- **Metro style apps:** Leverage the JavaScript and C# code in [Metro style apps](#) and [Working with the code examples](#). To get the access token, for JavaScript, one way is to call the **WL.getSession** method and look in the returned **session** object's **access_token** property. For C#, in the custom event handler method that you specify for the **LiveAuthClient.LoginCompleted** event, look in the **LoginCompletedEventArgs.Session.AccessToken** property.
- **Windows Phone app:** Leverage the C# code in [Windows Phone apps](#) and [Working with the code examples](#). To get the access token, in the custom event handler method that you specify for the **LiveAuthClient.LoginCompleted** event, look in the **LoginCompletedEventArgs.Session.AccessToken** property.
- **Client-side website or script that uses JavaScript:** Leverage the code in [Web apps](#). To get the access token, one way is to call the **WL.getSession** method and look in the returned **session** object's **access_token** property.
- **All other app types:** Leverage the Representational State Transfer (REST) code in the [Signing users in](#) topic's "Signing In Users with REST" section to learn how to get the access token.

[Top](#)

Next steps

Now that you've learned the basics, you can use our [XMPP code examples](#).

© 2011 Microsoft Corporation. All rights reserved.

XMPP code examples

[This documentation is preliminary and is subject to change.]

We provide these code samples that demonstrate how to work with the Windows Live Messenger Extensible Messaging and Presence Protocol (XMPP) service from an Android client app, a Java console client app, and a Windows Presentation Foundation (WPF) client app.

- [Android XMPP client code sample](#)
- [Java console-based XMPP client code sample](#)
- [WPFXMPP client code sample](#)

Android XMPP client code sample

[Download the Android XMPP client code sample.](#)

We built our Android XMPP client code sample on the Android 2.3.3 platform with the asmack open-source XMPP library, version 2010.05.07.

[Download the asmack library, version 2010.05.07.](#)

This code sample uses our implementation of the OAuth implicit grant flow for getting an access token. You'll need an access token to sign in to the Messenger XMPP service. To get an access token, see the [Getting started using Messenger with XMPP](#) topic's "Get an access token" section. For details about our OAuth implicit grant flow, see [OAuth 2.0](#).

This code sample highlights these features.

- **WLMainActivity.java**—This is the app's main class. This class gets the access token and maintains the UI. We've hard-coded the sample client ID and redirect domain that we use to get the access token. You must change these two constants in the code to use your own client ID and redirect domain.

```
public static final String OAuthClientId = "0000000048058C2E";
public static final String OAuthRedirectUri = "http://xmpp.msgr-tst.com/desktop";
```

- **XmpClient.java**—This is our XMPP client implementation using the asmack library. This class has all the logic for XMPP-specific operations like signing in to the Messenger XMPP service using an access token and handling XMPP stanzas.
- **XMessengerOAuth2.java**—This class uses the asmack library to handle X-MESSENGER-OAUTH2 authentication.

[Top](#)

Java console-based XMPP client code sample

[Download the Java console-based XMPP client code sample.](#)

We built our Java console-based XMPP client code sample with JRE version 1.6 and the Smack open-source XMPP library, version 3.2.1.

[Download the Smack library, version 3.2.1.](#)

You must change the code to add your access token to sign in to the Messenger XMPP service. To get an access token, see the [Getting started using Messenger with XMPP](#)

topic's "Get an access token" section.

This code sample highlights these features.

- **Program.java**—This is the app's main entry point. You must change this class's `TestAccessToken` constant to use your access token, like this.

```
public static final String TestAccessToken = "ACCESS_TOKEN"
```

- **XmpClient.java**—This is the XMPP client implementation using the Smack library. The class has all the logic for XMPP-specific operations like signing in to the Messenger XMPP service using an access token and handling XMPP stanzas.
- **XMessengerOAuth2.java**—This class uses the Smack library to handle X-MESSENGER-OAUTH2 authentication.

[Top](#)

WPF XMPP client code sample

[Download the WPF XMPP client code sample.](#)

We built our WPF XMPP client code sample with the .NET Framework 4.0 and the agsXMPP open-source XMPP library, version 1.1 (under the GNU General Public License (GPL)).

[Download the agsXMPP library, version 1.1.](#)

This code sample uses our implementation of the OAuth implicit grant flow for getting an access token. You'll need an access token to sign in to the Messenger XMPP service. To get an access token, see the [Getting started using Messenger with XMPP](#) topic's "Get an access token" section. For details about our OAuth implicit grant flow, see [OAuth 2.0](#).

This code sample highlights these features.

- **XmppMessengerOAuth2.cs**—This code gets an access token. We've hard-coded the sample client ID and redirect domain that we use to get the access token. You must change these two constants in the code to use your own client ID and redirect domain.

```
public const string OAuthAppClientId = "0000000048058C2E";  
public const string OAuthAppRedirectUri = "http://xmpp.msgr-tst.com/desktop";
```

- **XmppUser.cs**—This class represents a user and includes all the logic for XMPP specific operations like signing in and signing out, sending a message to a buddy, and sending the user's presence info.
- **XmppBuddy.cs**—This class represents one of the user's buddies and includes all buddy-related properties in the contact list UI.
- **XmppMessage.cs**—This class represents an incoming or outgoing message in the conversation window's message list UI.
- **OAuthMechanism.cs**—This class wraps a custom Simple Authentication and Security Layer (SASL) implementation for X-MESSENGER-OAUTH2 authentication.

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Sharing a user's status

[This documentation is preliminary and is subject to change.]

You can use code to update a Live Connect user's status. When you update a user's status, it shows up in the user's status message.

To fully leverage the code examples in this topic, you can use them in the context of larger code reference samples that we provide in [Working with the code examples](#).

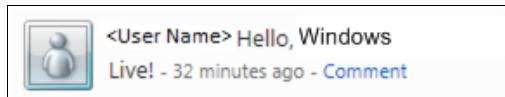
- [Update a user's status message by using JavaScript](#)
- [Update a user's status message by using C#](#)
- [Update a user's status message by using REST](#)

Update a user's status message by using JavaScript

To share text, use code similar to this.

```
function shareUserStatus_onClick() {  
    var scopes = ["wl.share"];  
    WL.login({ scope: scopes }, onWLLoginComplete)  
    function onWLLoginComplete() {  
        var session = WL.getSession();  
        if (session.error) {  
            document.getElementById("infoLabel").innerText = "Error signing in: " + session.error;  
        }  
        else {  
            path = "me/share"  
            var body = {  
                message: "Hello, Windows Live!"  
            };  
            WL.api({ path: path, method: "POST", body: body }, onShareStatusComplete)  
        }  
    };  
    function onShareStatusComplete(result) {  
        if (result.error) {  
            document.getElementById("infoLabel").innerText = "Error enumerating comments: " + result.error.code;  
        }  
        else {  
            msg = "Status: " + result.message;  
            document.getElementById("infoLabel").innerText = msg;  
        }  
    };  
}
```

The result looks like this.

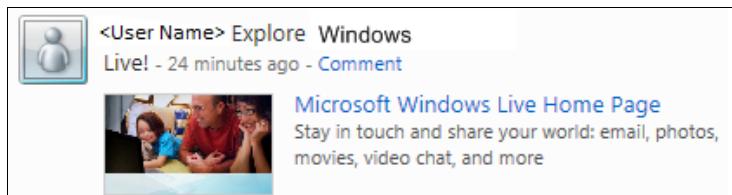


To share text with a link, with all possible keys specified, use code like the following.

```
var body = {  
    message: "Explore Windows Live",  
    link: "http://explore.live.com/home",  
    description: "Stay in touch and share your world: email, photos, movies, video chat, and more!"  
,  
    picture: "http://res2.explore.live.com/resbox/en/Live%20Explore/Main/a/1/a1ce31bd-6291-42fb-9b5  
1-b4cf2013481c/a1ce31bd-6291-42fb-9b51-b4cf2013481c.jpg",  
    name: "Windows Live Home"  
};
```

Note In the preceding code, only the **message** and **link** key-value pairs are required.

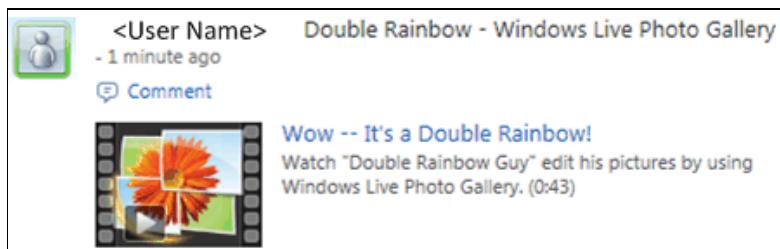
The result looks like this.



To share video, with all possible keys specified, use code like the following.

```
var body = {
    message: "Double Rainbow - Windows Live Photo Gallery",
    link: "http://www.youtube.com/watch?v=8jXz7NrfzsI",
    description: "Watch \"Double Rainbow Guy\" edit his pictures by using Windows Live Photo Galler y. (0:43)",
    picture: "http://res2.explore.live.com/resbox/en/Live%20Explore/Main/2/f/2f70d4d7-1640-4414-929 8-9957ab5e01c3/2f70d4d7-1640-4414-9298-9957ab5e01c3.png",
    name: "Wow -- It's a Double Rainbow!",
    source: "http://www.youtube.com/v/8jXz7NrfzsI?version=3"
};
```

The result looks like this.



Note In the preceding code, only the **message**, **link**, and **source** key-value pairs are required.

[Top](#)

Update a user's status message by using C#

To share status, use code similar to this. For brevity, only sharing video is shown here, as it requires the most info.

```
private void shareUserStatus_Click(object sender, RoutedEventArgs e)
{
    if (session == null || session.Status != LiveConnectSessionStatus.Connected)
    {
        MessageBox.Show("You must sign in first.");
    }
    else
    {
        client = new LiveConnectClient(session);
        var status = new Dictionary<string, object>();
        status.Add("message", "Double Rainbow - Windows Live Photo Gallery");
        status.Add("link", "http://www.youtube.com/watch?v=8jXz7NrfzsI");
        status.Add("description", "Watch \"Double Rainbow Guy\" edit his pictures by using Windows Live Photo Gallery. (0:43)");
        status.Add("picture", "http://res2.explore.live.com/resbox/en/Live%20Explore/Main/2/f/2f70d4d7-1640-4414-9298-9957ab5e01c3/2f70d4d7-1640-4414-9298-9957ab5e01c3.png");
        status.Add("name", "Wow -- It's a Double Rainbow!");
        status.Add("source", "http://www.youtube.com/v/8jXz7NrfzsI?version=3");
        client.PostCompleted += new EventHandler<LiveOperationCompletedEventArgs>(ShareUserStatus_P ostCompleted);
        client.PostAsync("me/share", status);
    }
}

void ShareUserStatus_PostCompleted(object sender, LiveOperationCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show("Activity shared.");
    }
    else
    {
        MessageBox.Show(e.Error.Message);
```

```
    }  
}
```

[Top](#)

Update a user's status message by using REST

If a Live Connect user signs in and consents to a **wl.share** scope request, and you have an access token that corresponds to this consented request, you can share an activity by making a call with Representational State Transfer (REST) and these parameters.

```
POST https://apis.live.net/v5.0/me/share  
  
Authorization: Bearer ACCESS_TOKEN  
Content-Type: application/json  
  
{  
    message: "Explore Windows Live",  
    link: "http://explore.live.com/home",  
    description: "Stay in touch and share your world: email, photos, movies, video chat, and more!",  
    picture: "http://explore.live.com/alce...part of URL omitted for brevity...481c.jpg",  
    name: "Windows Live Home"  
}
```

In the preceding code example, replace *ACCESS_TOKEN* with your access token string. Also, replace the JavaScript Object Notation (JSON)-formatted data with key-value pairs appropriate to sharing text or to sharing text with a link, as shown in the preceding section's code examples.

For more information about how to make server-side HTTP calls with REST, see [Server-side scenarios](#). For example, using PHP, in either of the `callback.php` or `callback_mysql.php` scripts that are described in that topic, you could replace the call to the `callRestApi` function with something like the following.

```
$activityData = array('message' => 'Explore Windows Live',  
    'link' => 'http://explore.live.com/home',  
    'description' => 'Stay in touch and share your world: email, photos, movies, video chat, and more!',  
    'picture' => 'http://explore.live.com/alce...part of URL omitted for brevity...481c.jpg',  
    'name' => 'Windows Live Home'  
);  
$jsonActivityData = json_encode($activityData);  
$results = callRestApi($tokens['access_token'],  
    REST_PATH_ME . REST_PATH_SHARE,  
    REST_API_POST,  
    array('Content-Type: application/json'),  
    $jsonActivityData);
```

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

Live Connect reference

[This documentation is preliminary and is subject to change.]

This reference provides information about how to call the Live Connect APIs for JavaScript, C#, Representational State Transfer (REST), and Windows Live Messenger Extensible Messaging and Presence Protocol (XMPP).

To learn how to use these APIs to accomplish common solution-development scenarios, see the [Live Connect website](#).

This reference contains these topics:

- [Scopes and permissions](#)
- [JavaScript API](#)
- [Managed API](#)
- [REST API](#)
- [Controls API](#)
- [Messenger XMPP reference](#)
- [Server-Side scenarios](#)
- [OAuth 2.0](#)
- [Messenger share button](#)
- [Supported locales](#)

© 2011 Microsoft Corporation. All rights reserved.

Scopes and permissions

[This documentation is preliminary and is subject to change.]

Before your application makes requests of the Live Connect APIs to work with Live Connect info, in most cases you must get permission from the user who owns the info to access his or her info or to create new objects on his or her behalf. In the Live Connect APIs, this permission is called a *scope*. Each scope grants a different permission level. More info about each scope follows later in this topic. For information about requesting a scope and obtaining user consent, see [Obtaining user consent](#).

- Scope types
- Core scopes
- Extended scopes
- Developer scopes
- Subset and superset behavior
- Accessing a user's public info
- Scope details

Scope types

There are three types of scopes:

- **Core** scopes are central to the Live Connect APIs and involve users' core profile and contact data.
- **Extended** scopes allow you to work with users' extended profile and contact data.
- **Developer** scopes allow you to work with developers' client IDs.

[Top](#)

Core scopes

Scope	Enables
wl.basic	Read access to a user's basic profile info. Also enables read access to a user's list of contacts.
wl.offline_access	The ability of an application to read and update a user's info at any time. Without this scope, an app can access the user's info only while the user is signed in to Live Connect and is using your app.
wl.signin	Single sign-in behavior. With single sign-in, users who are already signed in to Live Connect are also signed in to your website.

[Top](#)

Extended scopes

Scope	Enables
wl.birthday	Read access to a user's birthday info including birth day, month, and year.
wl.calendars	Read access to a user's calendars and events.
wl.calendars_update	Read and write access to a user's calendars and events.
wl.contacts_birthday	Read access to the birth day and birth month of a user's contacts. Note that this also gives read access to the user's birth day, birth month, and birth year.
wl.contacts_create	Creation of new contacts in the user's address book.
wl.contacts_calendars	Read access to a user's calendars and events. Also enables read access to any calendars and events that other users have shared with the user.
wl.contacts_photos	Read access to a user's albums, photos, and videos, and their associated comments and tags. Also enables read access to any albums, photos, and videos that other users have shared with the user.
wl.contacts_skydrive	Read access to SkyDrive documents and photos that other users have shared with the

	user. Note that this also gives read access to the user's documents and photos stored in SkyDrive.
wl.emails	Read access to a user's personal, preferred, and business email addresses.
wl.events_create	Creation of events on the user's default calendar.
wl.phone_numbers	Read access to a user's personal, business, and mobile phone numbers.
wl.photos	Read access to a user's photos, videos, and albums.
wl.postal_addresses	Read access to a user's postal addresses.
wl.share	Enables updating a user's status message.
wl.skydrive	Read access to a user's documents and photos stored in SkyDrive.
wl.skydrive_update	Read and write access to a user's documents and photos stored in SkyDrive.
wl.work_profile	Read access to a user's employer and work position information.

[Top](#)

Developer scopes

Scope	Enables
wl.applications	Read access to a developer's client IDs that have been created to work with the Live Connect APIs.
wl.applications_create	Creation of new client IDs on behalf of a developer.

[Top](#)

Subset and superset behavior

Certain scopes give access to a subset of the data addressed by other scopes. For example, [wl.birthday](#) gives access to the user's birthday, but [wl.contacts_birthday](#) gives access both to the user's birthday and to birthdays of the user's Messenger friends. In requests that specify multiple scopes, if one scope is a superset of another, the subset scope is ignored. Likewise, if an app has been granted access to a subset scope (for example, [wl.birthday](#)), and the user later grants access to a superset scope (for example, [wl.contacts_birthday](#)), the subset scope is revoked as redundant. The following table shows the scopes that share a subset/superset relationship.

Subset scope	Superset scope(s)
wl.birthday	wl.contacts_birthday
wl.calendars	wl.contacts_calendars wl.calendars_update
wl.photos	wl.contacts_photos
wl.skydrive	wl.contacts_skydrive wl.skydrive_update
wl.applications	wl.applications_create

[Top](#)

Accessing a user's public info

There is an exception to the rule that you must get the permission from the user before you can access his or her information: your application can access a user's publicly available information without requesting any scope. Public information includes the user's ID, his or her first and last names, display name, gender, locale, and picture. The user's

friends list is also available if the user has elected to make it public. For example, the following **GET** request, without any access token specified, would return the user's public profile information.

```
https://apis.live.net/v5.0/8c8ce076ca27823f
```

The information returned by Live Connect would look like the following.

```
{
  "id": "8c8ce076ca27823f",
  "name": "Roberto Tamburello",
  "first_name": "Roberto",
  "last_name": "Tamburello",
  "gender": null,
  "locale": "en_US"
}
```

As another example, a **GET** request for the user's picture, also without any access token specified, would look like the following.

```
https://apis.live.net/v5.0/8c8ce076ca27823f/picture
```

The request would be redirected to a URL that would look something like the following.

```
http://blufiles.storage.msn.com/y1m9UZK4sELhooi0vvVFy0DvE0xIMPK-lZXeZQohhW9LmEXwLHZHyh9ue2c3oWnrTqx0r5q3J9N5KtFI58Rfy-u-Q
```

[Top](#)

Scope details

The following sections provide additional details about the available scopes.

In several of these sections, tables describe the Live Connect Representational State Transfer (REST) objects and the corresponding structures that these scopes can access. For more information about these REST objects and structures, see [REST API](#).

wl.applications

The **wl.applications** scope enables read access to a developer's client IDs.

This scope enables read access to all of the [Application](#) object's structures.

[Top](#)

wl.applications_create

The **wl.applications_create** scope enables the programmatic creation of new client IDs on behalf of a developer.

Similar to the **wl.applications** scope, this scope enables access to all of the [Application](#) object's structures.

Note When a user consents to the **wl.applications_create** scope, the user also implicitly consents to the info that is accessible with the **wl.applications** scope. (However, if the user consents to the **wl.applications** scope and then later consents to the **wl.applications_create** scope, the **wl.applications** scope is revoked because **wl.applications** is a subset of **wl.applications_create** and therefore becomes redundant.)

[Top](#)

wl.basic

The **wl.basic** scope enables read access to a user's basic profile info and to the user's list of contacts.

The following table lists the structures that can be accessed with user consent to the **wl.basic** scope.

REST object	Structure
User	link
User	updated_time
Contact	id
Contact	first_name
Contact	last_name
Contact	name
Contact	gender
Contact	is_friend

Contact	is_favorite
Contact	user_id
Contact	email_hashes
Contact	birth_day (also requires the wl.contacts_birthday scope)
Contact	birth_month (also requires the wl.contacts_birthday scope)
Contact	updated_time

[Top](#)

wl.birthday

The **wl.birthday** scope enables read access to a user's birth-date info.

The following table lists the structures that can be accessed with user consent to the **wl.birthday** scope.

REST object	Structure
User	birth_day
User	birth_month
User	birth_year

[Top](#)

wl.calendars

The **wl.calendars** scope enables read access to a user's calendars and events.

wl.calendars_update

The **wl.calendars_update** scope enables read and write access to a user's calendars and events.

wl.contacts_birthday

The **wl.contacts_birthday** scope enables read access to birthdate info of a user's contacts.

The following table lists the structures that can be accessed with user consent to the **wl.contacts_birthday** scope.

REST object	Structure
Contact	birth_day (also requires the wl.basic scope)
Contact	birth_month (also requires the wl.basic scope)

Note When a user consents to the **wl.contacts_birthday** scope, the user also implicitly consents to access to the info that is covered by the **wl.birthday** scope. (However, if the user consents to the **wl.birthday** scope and then later consents to the **wl.contacts_birthday** scope, the **wl.birthday** scope is revoked because it is a subset of **wl.contacts_birthday** and therefore becomes redundant.)

[Top](#)

wl.contacts_create

The **wl.contacts_create** scope enables the creation of contacts within a user's address book.

wl.contacts_calendars

The **wl.contacts_calendars** scope enables read access to a user's calendars and events. Also enables read access to calendars and events that other users have shared with the user. Permissions to shared calendars and events are

restricted by the permissions that have been granted to the consenting user.

Note When a user consents to the **wl.contacts_calendars** scope, the user also implicitly consents to access to the info that is covered by the **wl.calendars** scope. (However, if the user consents to the **wl.calendars** scope and then later consents to the **wl.contacts_calendars** scope, the **wl.calendars** scope is revoked because it is a subset of **wl.contacts_calendars** and therefore becomes redundant.)

wl.contacts_photos

The **wl.contacts_photos** scope enables read access to a user's albums, photos, and videos, and to their associated comments and tags. This scope also enables read access to any albums, photos, and videos that other users have shared with the user.

This scope enables read access to all of the structures of the **Album**, **Photo**, and **Video** objects for a friend.

Note When a user consents to the **wl.contacts_photos** scope, the user also implicitly consents to access to the info that is covered by the **wl.photos** scope. (However, if the user consents to the **wl.photos** scope and then later consents to the **wl.contacts_photos** scope, the **wl.photos** scope is revoked because it is a subset of **wl.contacts_photos** and therefore becomes redundant.)

[Top](#)

wl.contacts_skydrive

The **wl.contacts_skydrive** scope enables read access to SkyDrive documents and photos that other users have shared with the user.

Note When a user consents to the **wl.contacts_skydrive** scope, the user also implicitly consents to access to the info that is covered by the **wl.skydrive** scope. (However, if the user consents to the **wl.skydrive** scope and then later consents to the **wl.contacts_skydrive** scope, the **wl.skydrive** scope is revoked because it is a subset of **wl.contacts_skydrive** and therefore becomes redundant.)

[Top](#)

wl.emails

The **wl.emails** scope enables read access to a user's email addresses.

The following table lists the structures that can be accessed with user consent to the **wl.emails** scope.

REST object	Structure
User	emails
User	preferred (emails object)
User	account (emails object)
User	personal (emails object)
User	business (emails object)

[Top](#)

wl.events_create

The **wl.events_create** scope enables the creation of events on the user's default calendar.

This scope enables access to all of the **Event** object's structures.

[Top](#)

wl.offline_access

The **wl.offline_access** scope enables an app to read and update a user's info at any time. Without this scope, an app can access the user's info only while the user is signed in to Live Connect and is using the app.

[Top](#)

wl.phone_numbers

The **wl.phone_numbers** scope enables access to a user's phone numbers.

The following table lists the structures that can be accessed with user consent to the **wl.phone_numbers** scope.

REST object	Structure
-------------	-----------

User	phones
User	personal (phones object)
User	business (phones object)
User	mobile (phones object)

[Top](#)

wl.photos

The **wl.photos** scope enables read access to a user's photos, videos, and albums.

This scope enables read access to all of the structures of the **Album**, **Photo**, and **Video** objects for a user.

[Top](#)

wl.postal_addresses

The **wl.postal_addresses** scope enables read access to a user's postal addresses.

The following table lists the structures that can be accessed with user consent to the **wl.postal_addresses** scope.

REST object	Structure
User	addresses
User	personal (addresses object)
User	street (personal object)
User	street_2 (personal object)
User	city (personal object)
User	state (personal object)
User	postal_code (personal object)
User	region (personal object)
User	business (addresses object)
User	street (business object)
User	street_2 (business object)
User	city (business object)
User	state (business object)
User	postal_code (business object)
User	region (business object)

[Top](#)

wl.share

The **wl.share** scope enables the creation of activities on a user's activity feed and status message.

This scope enables access to all of the structures of the **Activity** object.

[Top](#)

wl.signin

The **wl.signin** scope enables single sign-in behavior. Users who are already signed in to Live Connect are also signed in to your app and therefore do not have to enter their credentials.

For info about the sign-in process, see [Signing users in](#).

wl.skydrive

The **wl.skydrive** scope enables read access to a user's documents and photos stored on SkyDrive.

[Top](#)

wl.skydrive_update

The **wl.skydrive_update** scope enables read and write access to a user's documents and photos stored on SkyDrive.

[Top](#)

wl.work_profile

The **wl.work_profile** scope enables read access to a user's employer and work position info.

The following table lists the structures that can be accessed with user consent to the **wl.work_profile** scope.

REST object	Structure
User	work
User	employer (work array)
User	name (employer object)
User	position (work array)
User	name (position object)

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

REST API

[This documentation is preliminary and is subject to change.]

The Live Connect Representational State Transfer (REST) API enables apps that work with Live Connect to programmatically access info about users, their contacts and friends, their web-based activities and personal status, their media stored on SkyDrive, and more.

This topic contains the following sections:

- [Supported HTTP verbs](#)
- [Support for Resumable Downloads](#)
- [Understanding the objects](#)
- [Activity object](#)
- [Album object](#)
- [Application object](#)
- [Calendar object](#)
- [Comment object](#)
- [Contact object](#)
- [Error object](#)
- [Event object](#)
- [File object](#)
- [Folder object](#)
- [Friend object](#)
- [Permissions object](#)
- [Photo object](#)
- [Tag object](#)
- [User object](#)
- [Video object](#)
- [Additional URL parameters](#)

Supported HTTP verbs

The Live Connect REST API supports the standard HTTP verbs.

- GET—Returns a representation of a resource.
- POST—Adds a new resource to a collection
- PUT—Updates a resource at the location pointed by the URL or creates a new one if it doesn't exist.
- DELETE—Deletes a resource.
- MOVE—Moves the location of a resource.
- COPY—Duplicates a resource.

Request URLs must reference <https://apis.live.net/v5.0/>, followed by the path for the object you are working with, any parameters, and finally the access token (represented here as ACCESS_TOKEN).

https://apis.live.net/v5.0/me/albums?access_token=ACCESS_TOKEN

[Top](#)

Support for resumable downloads

Live Connect provides support for resumable downloads by using the [HTTP Range header](#). If a download is interrupted, an application can continue the download at a later time, requesting only the byte range for the remainder of the file. This is particularly useful when downloading larger files, or in situations where network reliability is less than optimal.

Understanding the objects

In the Live Connect REST API, each category of info is contained in an *object*; furthermore, objects consist of a combination of child *objects*, *arrays*, and *values*. To access a specific object, you need to know the unique *path* to, and in many cases also the unique ID of, that object.

The following table lists the objects that are supported by Live Connect REST API requests, along with valid object paths.

Object	Description	Valid object paths
Activity	An activity in a user's activity feed and status message.	/me/share /USER_ID/share
Album	A user's album in SkyDrive. Albums can contain combinations of photos, videos, files, and folders.	/me/albums /USER_ID/albums /ALBUM_ID An Album object, if it exists, can be returned as part of /ALBUM_ID/files, /me/skydrive/files, /me/skydrive/shared/files,

		/me/skydrive/shared/albums, or /USER_ID/skydrive/files.
Application	A client ID that works with Live Connect.	/me/applications /USER_ID/applications /APPLICATION_ID
Calendar	A user's calendar.	/me/calendars /USER_ID/calendars /CALENDAR_ID
Comment	A comment that is made by a user on a photo or a video in SkyDrive.	/PHOTO_ID/comments /VIDEO_ID/comments /COMMENT_ID
Contact	A user's Hotmail contact. (If the contact's is_friend value is set to true , the contact is also the user's friend.)	/me/contacts /USER_ID/contacts /CONTACT_ID
Error	Info about an error that is returned by the Live Connect APIs.	None
Event	An event on a user's calendar.	/me/events /USER_ID/events /CALENDAR_ID/events
File	A user's file in SkyDrive.	/FILE_ID A File object, if it exists, can be returned as part of /FOLDER_ID/files, /me/skydrive/files, /me/skydrive/shared, /me/skydrive/shared/files, or /USER_ID/skydrive/files.
Folder	A user's folder in SkyDrive. Folders can contain combinations of photos, videos, files (such as Microsoft Office documents) and subfolders.	/FOLDER_ID A Folder object, if it exists, can be returned as part of /FOLDER_ID/files, /me/skydrive/files, /me/skydrive/shared/files, /USER_ID/skydrive/files, or /ALBUM_ID/files.
Friend	A user's friend. Friends are a subset of a user's contacts and reflect people the user has a publicly visible relationship with. Explicit permission is not required to view a user's friends. Friend returns less data than Contact and is thus more compact for scenarios where only the names and IDs of the user's friends are needed.	/me/friends /USER_ID/friends
Permissions	A list of scopes.	/me/permissions /USER_ID/permissions
Photo	A user's photo in SkyDrive.	/PHOTO_ID A Photo object, if it exists, can be returned as part of /ALBUM_ID/files, /ALBUM_ID/photos, /FOLDER_ID/files, /me/skydrive/files, /me/skydrive/shared/photos, or /USER_ID/skydrive/files.
Tag	A tag that is associated with a photo or a video in SkyDrive.	/PHOTO_ID/tags /VIDEO_ID/tags /TAG_ID
User	A user.	/me /USER_ID
Video	A user's video in SkyDrive.	/VIDEO_ID A Video object, if it exists, can be returned as part of /ALBUM_ID/files, /ALBUM_ID/videos, /FOLDER_ID/files,

		/me/skydrive/files, /me/skydrive/shared/videos, or USER_ID/skydrive/files.
--	--	--

In the preceding table, replace *USER_ID*, *CONTACT_ID*, *ALBUM_ID*, *PHOTO_ID*, *VIDEO_ID*, *FOLDER_ID*, or *APPLICATION_ID* with the target user ID, **Contact** object ID, **Album** object ID, **Photo** object ID, **Video** object ID, **Folder** object ID, or **Application** object ID or Live Connect client ID, respectively.

In the preceding table, you can use /me as a shortcut for the signed-in user instead of specifying /*USER_ID*. However, you cannot use /me unless you also provide an access token.

The Live Connect REST API exchanges info between apps and Live Connect and SkyDrive in JavaScript Object Notation (JSON) format. (Because of this, the JSON terms *structure*, *object*, *array*, and *value* are used throughout this reference.)

When your app makes a **GET** request for an object, the Live Connect REST API returns the requested info as a JSON-formatted structure. Similarly, when your app makes a **POST** or **PUT** request, the Live Connect REST API requires the info to be submitted to it as a JSON-formatted structure. (For more info about JSON, see [Introducing JSON](http://www.json.org) (<http://www.json.org>)).

[Top](#)

Structure value options

In the following sections, the column labeled **R/W** indicates whether a value is read-only or read/write, and the column labeled **Required on Create** indicates whether a value is required for the creation of an object.

[Top](#)

Activity object

The **Activity** object contains info about activities in a user's activity feed and status message. The Live Connect REST API supports creating **Activity** objects. Use the [wl.share](#) scope to create **Activity** objects.

The **Activity** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
from	object	R		Info about the user who shared the activity. This structure is visible only in the Activity object that is returned if the activity was successfully shared.
name (from object)	string	R		The name of the user who shared the activity. This structure is visible only in the Activity object that is returned if the activity was successfully shared.
id (from object)	string	R		The Windows Live ID of the user who shared the activity. This structure is visible only in the Activity object that is returned if the activity was successfully shared.
message	string	RW	YES	The text of the message.
link	string	RW		The URL of the content to share.
description	string	RW		A description of the content to share.
picture	string	RW		The URL of a thumbnail image of the content to share.
name	string	RW		The title of the content to share.
source	string	RW		The URL to the video source file, if this is a video activity.

Important Values for **Activity** object structures must not exceed 512 characters, with the exception of the **source** structure value, which must not exceed 487 characters. If the value for either the **link** or **source** field exceeds the respective maximum limit, the call returns an error. For all other structures, values that exceed the maximum length are truncated and sent.

The following is an example of an **Activity** object.

```
{
  message: "Explore Windows Live Hotmail",
  link: "http://explore.live.com/windows-live-hotmail",
  description: "Give us 15 seconds...we'll show you the new Hotmail!",
  picture: "http://res2.explore.live.com/...part of URL omitted for brevity...fd39.jpg",
  name: "Windows Live Hotmail Home"
}
```

To create an **Activity** object by using the Live Connect REST API, make a **POST** request to /me/share. Include the properties for the activity in the request body as shown here.

Content-Type: application/json

```
{
  message: "Explore Windows Live Hotmail",
  link: "http://explore.live.com/windows-live-hotmail",
  description: "Give us 15 seconds...we'll show you the new Hotmail!",
  picture: "http://res2.explore.live.com/...part of URL omitted for brevity...fd39.jpg",
```

```

        "name": "Windows Live Hotmail Home"
    }

```

Note

If the call is successful, the Live Connect REST API returns the following JSON-formatted object.

```
{
    "from": {
        "name": "Roberto Tamburello",
        "id": "8c8ce076ca27823f"
    },
    "message": "...The message value...",
    "link": "...The link value, if one was provided, otherwise null...",
    "description": "...The description value, if one was provided, otherwise null...",
    "picture": "...The picture value, if one was provided, otherwise null...",
    "name": "...The name value, if one was provided, otherwise null...",
    "source": "...The source value, if one was provided, otherwise null..."
}
```

[Top](#)

Album object

The **Album** object contains info about a user's albums in SkyDrive. Albums are stored at the rootlevel of a user's SkyDrive directory, and can contain combinations of photos, videos, files and folders. The Live Connect REST API supports reading **Album** objects. Use the **wl.photos** scope to read a user's **Album** objects. Use the **wl.skydrive** scope to read a user's files. Use the **wl.contacts_photos** scope to read any albums, photos, and videos that other users have shared with the user.

The **Album** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array container for Album objects when a collection of objects is returned.
id	string	R		The Album object's ID.
from	object	R		info about the user who authored the album.
name (from object)	string	R		The name of the user who authored the album.
id (from object)	string	R		The Windows Live ID of the user who authored the album.
name	string	RW	YES	The name of the album.
description	string/null	RW		A description of the album, or null if no description is specified.
count	number	R		The total number of items in the album.
link	string	R		A URL of the album, hosted in SkyDrive.
parent_id	string	R		The resource ID of the parent.
upload_location	string	R		The URL to upload items to the album, hosted in SkyDrive. Requires the wl.skydrive scope.
type	string	R		The type of object; "album" in this case.
created_time	string	R		The time, in ISO 8601 format, at which the album was created.
updated_time	string	R		The time, in ISO 8601 format, at which the album was last updated.
shared_with	object	R		The object containing permissions info for the album. Requires the wl.skydrive scope.
access (shared_with object)	string	R		A localized string that contains info about who can access the album. The options are: <ul style="list-style-type: none"> • People I selected • Just me • Everyone (public) • Friends • My friends and their friends • People with a link The default is Just me .

The following is an example of a collection of **Album** objects. (For brevity, only the first object is shown.)

```
{ "data": [ { "id": "album.8c8ce076ca27823f.8C8CE076CA27823F!126", "from": { "name": "Roberto Tamburello", "id": "8c8ce076ca27823f" }, "name": "My Sample Album 1", "description": "", "parent_id": "folder.de57f4126ed7e411", "upload_location": "https://apis.live.net/v5.0/folder.de57f4126ed7e411.DE57F4126ED7E411!126/files/", "count": 4, "link": "https://cid-8c8ce076ca27823f.skydrive.live.com/redir.aspx?page\u003dsself\u0026resid\u003d8C8CE076CA27823F!126\u0026tys", "type": "album", "shared_with": { "access": "Everyone (public)" }, "created_time": "2011-04-21T23:19:47+0000", "updated_time": "2011-04-22T19:18:12+0000" }, { ... } ] }
```

To get a collection of **Album** objects by using the Live Connect REST API, make a **GET** request to `/me/albums`.

Note To get an album's cover image, call `/ALBUM_ID/picture`.

To create a new **Album** resource, make a **POST** request to `/me/albums`. Pass the `name` and `description` attributes in the request body as shown here.

```
{ "name": "Vacation 2011", "description": "Photos from our fun vacation." }
```

To update the properties of an **Album**, make a **PUT** request to `/ALBUM_ID`. Pass the attributes to update in the request body as shown here.

```
{ "name": "Maui Vacation 2011", "description": "Photos from our vacation on Maui." }
```

To delete an album, make a **DELETE** request to `/ALBUM_ID`.

[Top](#)

Application object

The **Application** object contains info about a developer's client IDs that work with Live Connect. The Live Connect REST API supports reading and creating **Application** objects. Use the [wl.applications](#) scope to read **Application** objects, and use the [wl.applications_create](#) scope to create **Application** objects.

The ability to create, read, update, and delete **Application** objects would be most useful for providers who build comment or sign-in widgets for use by third-party websites. Supplying an application ID for each website that uses the provider's widget would enable third-party sites to provide a redirect URL for OAuth authorization and track analytics for widget usage on <http://manage.dev.live.com>.

The **Application** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array of Application objects if a collection of objects is returned.
id	string	R		The Application object's ID. Do not specify this structure when you create a client ID, because the system generates it automatically upon creation.
name	string	RW	YES	The client ID's name.
client_id	string	R		The client ID. This structure is automatically generated upon creation.
client_secret	string	R		The client ID's client secret. This structure is automatically generated upon creation.
uri	string	RW	YES	The client ID's redirect URI. This structure is required only when you create a client ID. When you read a client ID, this structure appears in the uri (uris array) structure.
uris	array	R		An array that contains the redirect URI info. This structure is automatically generated upon creation.
uri (uris array)	string	R		The redirect URI. This structure is automatically generated upon creation.

type (uris array)	string	R		The redirect URI type. The only valid value is "web". This structure is automatically generated upon creation.
terms_of_service_link	string	RW		The URL of the client ID's terms of service webpage.
privacy_link	string	RW		The URL of the client ID's privacy webpage.
created_time	string	R		The time, in ISO 8601 format, at which the client ID was created. This structure is automatically generated upon creation.
updated_time	string	R		The time, in ISO 8601 format, at which the client ID was last updated. This structure is automatically generated upon creation.

The following is an example of a collection of **Application** objects. (For brevity, only the first object is shown.)

```
{
  "data": [
    {
      "id": "application.00000004c043705",
      "name": "Pilot Live Connect App",
      "client_id": "00000004c043705",
      "client_secret": "O9L3tC4ecrCdLDaNsNC3m4mdiBm9gYi4",
      "uris": [
        {
          "uri": "http://www.contoso.com",
          "type": "web"
        }
      ],
      "terms_of_service_link": "http://www.contoso.com/terms_of_service.php",
      "privacy_link": "http://www.contoso.com/privacy.php",
      "created_time": "2011-04-18T19:03:59+0000",
      "updated_time": "2011-04-25T18:24:26+0000"
    },
    {
      ...
    }
  ]
}
```

To get an **Application** object by using the Live Connect REST API, make a **GET** request to /me/applications.

To create an **Application** object by using the Live Connect REST API, make a **POST** request to /me/applications. Include the properties for the **Application** object in the request body as shown here.

Content-Type: application/json

```
{
  "name": "Pilot Live Connect App",
  "uri": "http://www.contoso.com",
  "terms_of_service_link": "http://www.contoso.com/terms_of_service.php",
  "privacy_link": "http://www.contoso.com/privacy.php"
}
```

Note

If the call is successful, the Live Connect REST API returns the following JSON-formatted object.

```
{
  "id": "application.00000004c043705",
  "name": "Pilot Live Connect App",
  "client_id": "00000004c043705",
  "client_secret": "O9L3tC4ecrCdLDaNsNC3m4mdiBm9gYi4",
  "uris": [
    {
      "uri": "http://www.contoso.com",
      "type": "web"
    }
  ],
  "terms_of_service_link": "http://www.contoso.com/terms_of_service.php",
  "privacy_link": "http://www.contoso.com/privacy.php",
  "created_time": "2011-04-18T19:03:59+0000",
  "updated_time": "2011-04-25T18:24:26+0000"
}
```

[Top](#)

Calendar object

The **Calendar** object contains info about a user's Hotmail calendar.

The **Calendar** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
id	string	R		The Calendar object's ID.
name	string	RW	YES	Name of the calendar.
description	string/null	RW		Description of the calendar.

created_time	string	R		The time, in ISO 8601 format, at which the calendar was created.
updated_time	string	R		The time, in ISO 8601 format, that the calendar was last updated.
from	object	R		Info about the user who owns the calendar.
name (from object)	string	R		The name of the user who uploaded the file.
id (from object)	string	R		The ID of the user who owns the calendar.
is_default	true/false	R		Flag indicating if this calendar is the default calendar.
subscription_location	string	RW		A public subscription URL with which Live Connect will periodically synchronize properties and events for this calendar. A NULL value indicates this is not a subscribed calendar.
permissions	string	R		<p>Role and permissions granted to the user for the calendar. The possible values are:</p> <ul style="list-style-type: none"> • free_busy: The user can see only free/busy info. • limited_details: The user can see a subset of all details. • read: The user can only read the content of the calendar events. • read_write: The user can read and write calendar and events. • co_owner: The user is co-owner of this calendar. • owner: The user is the owner of this calendar.

The following is an example of a collection of **Calendar** objects. (For brevity, only the first object is shown.)

```
{
  "data": [
    {
      "id": "calendar.42d4dbc866f94c83849c88c6eb9985bc",
      "name": "Birthday calendar",
      "description": "If you have birthdays listed for your contacts, they'll appear on this calendar. You can add more birthdays, b",
      "created_time": "2011-08-05T19:41:04+0000",
      "updated_time": "2011-08-05T19:41:04+0000",
      "from": {
        "name": null,
        "id": null
      },
      "is_default": false,
      "subscription_location": null,
      "permissions": "read",
      "type": "calendar"
    }
  ]
}
```

To get a collection of **Calendar** objects by using the Live Connect REST API, make a **GET** request to either `/me/calendars`, or `/USER_ID/calendars`.

To get the properties for a particular **Calendar** object, make a **GET** request with the ID of the calendar.

https://apis.live.net/v5.0/calendar.b4466224b2ca42798c3d4ea90c75aa56?access_token=ACCESS_TOKEN

To create a new **Calendar**, make a **POST** request to either `/me/calendars`, or `/USER_ID/calendars`. Pass the **name** (required) and **summary** (optional) strings as properties on the request body. The properties for the new calendar are returned on the response body, and the Location header contains the URI for the newly created calendar.

```
{
  "name": "Summer Events",
  "summary": "Things we are doing this summer."
}
```

Warning

The request body must not include the **subscription_location** property.

To subscribe to a public calendar, make a **POST** request to either `/me/calendars` or `/USER_ID/calendars`. Pass the **name** and **subscription_location** strings as properties on the request body. The value of **subscription_location** must be a valid online iCal calendar. Both properties are required.

```
{
  "name": "Soccer League",
  "subscription_location": "ical.sharedcalendars.com/98754auv"
}
```

To update a **Calendar**, make a **PUT** request to `/CALENDAR_ID`.Include the properties to update in the request body as shown here.

```
{
  "name": "Fall Happenings"
}
```

To delete a **Calendar**, make a **DELETE** request to `/CALENDAR_ID`.

In the preceding examples, replace CALENDAR_ID with the target calendar ID.

[Top](#)

Comment object

The **Comment** object contains info about comments associated with a photo or a video on SkyDrive. The Live Connect REST API supports reading **Comment** objects. Use the **wl.photos** scope to read **Comment** objects. Use the **wl.contacts_photos** scope to read the **Comment** objects that are associated with any albums, photos, and videos that other users have shared with the user.

The **Comment** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array of Comment objects if a collection of objects is returned.
from	object	R		Info about the user who created the comment.
id	string	R		The Comment object's id.
name (from object)	string	R		The name of the user who created the comment.
id (from object)	string	R		The ID of the user who created the comment.
message	string	RW	YES	The text of the comment. The maximum length of the comment is 10,000 characters.
created_time	string	R		The time, in ISO 8601 format, at which the comment was created.

The following is an example of a collection of **Comment** objects. (For brevity, only the first object is shown.)

```
{ "data": [
  {
    "from": {
      "name": "Roberto Tamburello",
      "id": "8c8ce076ca27823f"
    },
    "message": "A lighthouse built on some rocks.",
    "created_time": "2011-04-21T23:21:28+0000"
  },
  ...
]
```

To get a collection of **Comment** objects by using the Live Connect REST API, make a **GET** request to `/RESOURCE_ID/comments`.

[Top](#)

Contact object

The **Contact** object contains info about a user's Hotmail contacts. The Live Connect REST API supports reading **Contact** objects.

The **Contact** object contains the following structures.

Structure	Required scopes	Type	R/W	Required on Create	Description
id	wl.basic	string	R		The Contact object's ID.
first_name	wl.basic	string/null	RW	Optional	The contact's first name, or null if no first name is specified.
last_name	wl.basic	string/null	RW	Optional	The contact's last name, or null if no last name is specified.
name	wl.basic	string	R		The contact's full name, formatted for location.
gender	wl.basic	string/null	RW		The contact's gender. Valid values are "male", "female", or null if the contact's gender is not specified.
is_friend	wl.basic	true/false	R		Value that indicates whether the contact is set as a friend. If the contact is a friend, this value is true ; otherwise, it is false .
is_favorite	wl.basic	true/false	RW		Indicates if the contact is set as a favorite contact. If the contact is a favorite, this value is true ; otherwise, it is false .
user_id	wl.basic	string/null	R		The contact's ID, if the contact has one. If not, this

					value is null .
email_hashes	wl.basic	array	R		An array containing a SHA-256 hash for each one of the contact's email addresses. For more info, see Friend finder .
birth_day	wl.basic and wl.contacts_birthday	number/null	R		The day of the contact's birth date, or null if no birth date is specified.
birth_month	wl.basic and wl.contacts_birthday	number/null	R		The month of the contact's birth date, or null if no birth date is specified.
updated_time	wl.basic	string	R		The time, in ISO 8601 format, at which the user last updated the data.

When creating a new **Contact** object, at least one of the following attributes must be provided (these are marked as **Optional** in the structures table above).

- `first_name`
- `last_name`
- `preferred` ([User object](#), emails structure)
- `personal` ([User object](#), emails structure)
- `business` ([User object](#), emails structure)
- `name` ([User object](#), work structure, employer array)

The `preferred`, `personal`, `business`, and `name` structures belong to the [User object](#), and require restricted scopes (**wl.emails** and **wl.work_profile**, respectively). Therefore these attributes are not returned when getting a **Contact** object instance. However, it is possible to add these attributes when creating a new **Contact** object. Although the value cannot be directly returned, this behavior is handy in situations when you are adding a new contact by email address only, or if you want to create a contact that is a company. To ensure that the company name or email address appear in place of the contact name, omit the `first_name` and `last_name` fields. The email address or company name that you specify will then appear in place of the name in the returned **Contact** object instance.

Use the following JSON as a guide for defining the attributes in your request:

```
{
  "first_name": "",
  "last_name": "",
  "emails": [
    {
      "preferred": "",
      "personal": "",
      "business": ""
    }
  ],
  "work": [
    {
      "employer": {
        "name": ""
      }
    }
  ]
}
```

The following is an example of a collection of **Contact** objects. (For brevity, only the first object is shown.)

```
{
  "data": [
    {
      "id": "contact.b4466224b2ca42798c3d4ea90c75aa56",
      "first_name": "Henrik",
      "last_name": "Jensen",
      "name": "Henrik Jensen",
      "gender": null,
      "is_friend": false,
      "is_favorite": false,
      "user_id": null,
      "email_hashes": [
        "9ecdb19f4eb8e04304c5d1280368c42e85b6e4fe39f08b0c837ec592b905a620",
        "fc05492f50da6488aa14dcf221d395bcb29a4e43b43b250d60c68df4f831cad3"
      ],
      "birth_day": 29,
      "birth_month": 3,
      "updated_time": "2011-04-22T00:11:13+0000"
    },
    {
      ...
    }
  ]
}
```

To get a collection of **Contact** objects by using the Live Connect REST API, make a **GET** request to `/me/contacts`.

Note

If a **Contact** references a user, you can return a **User** object by making a **GET** request using the `user_id` value for the **Contact**.

To redirect a **GET** call to the URL of the picture of a contact who is a user, call `/USER_ID/picture`. (A **User** object's ID is not the same as a contact ID.)

[Top](#)

Error object

The **Error** object contains info about an error that is returned by the Live Connect APIs.

The **Error** object contains the following structures.

Structure	Type	Description
error	object	Info about the error.
code (error object)	string	The error code.
message (error object)	string	The error message.

The following is an example of an **Error** object.

```
{  
    "error": {  
        "code": "request_token_expired",  
        "message": "The provided access token has expired."  
    }  
}
```

[Top](#)

Event object

The **Event** object contains info about events on a user's Hotmail calendar. The Live Connect REST API supports creating **Event** objects. Use the **wl.events_create** scope to create **Event** objects. Use the **wl.calendars** scope to read **Event** objects on the user's calendar. Use the **wl.contacts_calendars** scope to read **Event** objects from the user's friend's calendars.

The **Event** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
id	string	R		The ID of the event.
name	string	RW	YES	The name of the event, with a maximum length of 255 characters. This structure is required.
created_time	string	R		The time the event was created in ISO 8601 format.
updated_time	string	R		The time, in ISO 8601 format, at which the event was created. This structure is visible only in the Event object that is returned if the event was successfully created.
description	string	RW		A description of the event, with a maximum length of 32,768 characters. This structure is required.
calendar_id	object	R		The ID of the calendar containing the event.
from	object	R		The object containing the name and ID of the organizer.
name (from object)	string	R		The name of the organizer.
id (from object)	string	R		The ID of the organizer.
start_time	string	RW	YES	The start time of the event, in ISO 8601 format. If no UTC marker (for example, 'Z') is present, time is assumed to be local time.
end_time	string	RW		The end time of the event, in ISO 8601 format. If no end time is specified, the default value is 30 minutes after start_time. This structure is optional when creating an event. If no UTC marker (for example, 'Z') is present, time is assumed to be local time.
location	string	RW		The name of the location at which the event will take place. The maximum length is 1,000 characters.
is_all_day_event	true/false	RW		A value that specifies whether the event is an all-day event. If the event is an all-day event, this value is true ; otherwise, it is false . If this structure is missing the default value is false .
is_recurrent	true/false	R		Specifies whether the event is recurring. The value is true if this is a recurrent event; false otherwise.
recurrence	string	R		The text description of the recurrence pattern (for example, "Occurs every week on Tuesday."). The value is Null if this is not a recurrent event. The

				value comes from the RecurrenceDescription attribute in the calendar backend).
reminder_time	number	RW		The time (in minutes) before the event for the reminder alarm.
availability	string	RW		<p>The user's availability status for the event. Valid values are:</p> <ul style="list-style-type: none"> • "free" • "busy" • "tentative" • "out_of_office " <p>The default is free.</p>
visibility	string	RW		<p>A value that specifies whether the event is publicly visible. Valid values are:</p> <ul style="list-style-type: none"> • "public"—the event is visible to anyone who can view the calendar. • "private"—the event is visible only to the event owner. <p>The default is public.</p>

The following is an example of an **Event** object.

```
{
  "id": "event.611afb17fa9448f28cdb8277e8ffeb77.e9f015000d0249ce847c5306a25d7d75",
  "name": "Global Project Risk Management Meeting",
  "description": "Generate and assess risks for the project",
  "calendar_id": "calendar.611afb17fa9448f28cdb8277e8ffeb77",
  "from": {
    "name": "William Flash",
    "id": "de57f4126ed7e411"
  },
  "start_time": "2011-04-20T01:00:00+0000",
  "end_time": "2011-04-20T02:00:00+0000",
  "location": "Building 81, Room 9981, 123 Anywhere St., Redmond WA 19599",
  "is_all_day_event": false,
  "availability": "busy",
  "visibility": "public",
  "updated_time": "2011-04-19T20:23:03+0000"
}
```

To return a list of events for a calendar by using the Live Connect REST API, make a **GET** request to `/CALENDAR_ID/events`. This will return all events between now and the next 30 days by default.

To return a list of events for a user, make a **GET** request to either `/me/events`, or `/USER_ID/events`.

To retrieve the properties for an **Event**, make a **GET** request to `/EVENT_ID`.

To create an **Event** object on the user's default calendar by using the Live Connect REST API, make a **POST** request to `/me/events`. Pass the properties for the event in the request body as shown here.

Content-Type: application/json

```
{
  name: "Global Project Risk Management Meeting",
  description: "Generate and assess risks for the project",
  start_time: "2011-04-20T01:00:00-07:00",
  end_time: "2011-04-20T02:00:00-07:00",
  location: "Building 81, Room 9981, 123 Anywhere St., Redmond WA 19599",
  is_all_day_event: false,
  availability: "busy",
  visibility: "public"
}
```

Note

If the call is successful, the Live Connect REST API returns the following JSON-formatted object.

```
{
  "name": "Global Project Risk Management Meeting",
  "description": "Generate and assess risks for the project",
  "start_time": "2011-04-20T01:00:00+0000",
  "end_time": "2011-04-20T02:00:00+0000",
  "location": "Building 81, Room 9981, 123 Anywhere St., Redmond WA 19599",
  "is_all_day_event": false,
  "availability": "busy",
  "visibility": "public",
  "updated_time": "2011-04-19T20:23:03+0000"
}
```

To delete an **Event**, make a **DELETE** request to `/EVENT_ID`.

Tip

The Live Connect REST API does not support creating or updating recurring events, however you can delete recurring events the same way you would delete a non-recurring event. Deleting a recurring event deletes all of its occurrences.

[Top](#)

File object

The **File** object contains info about a user's files in SkyDrive. The Live Connect REST API supports creating, reading, updating, and deleting **File** objects. Use the **wl.skydrive** scope to read **File** objects. Use the **wl.contacts_skydrive** scope to read any files that other users have shared with the user. Use the **wl.skydrive_update** scope to create, update, or delete **File** objects.

The **File** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array of File objects, if a collection of objects is returned.
id	string	R		The File object's ID.
from	object	R		Info about the user who uploaded the file.
name (from object)	string	R		The name of the user who uploaded the file.
id (from object)	string	R		The ID of the user who uploaded the file.
name	string	RW	YES	The name of the file.
parent_id	string	R		The id of the folder the file is currently stored in.
description	string/null	RW		A description of the file, or null if no description is specified.
size	number	R		The size, in bytes, of the file.
comments_count	number	R		The number of comments associated with the file.
comments_enabled	true/false	R		A value that indicates whether comments are enabled for the file. If comments can be made, this value is true ; otherwise, it is false .
is_embeddable	true/false	R		Flag indicating if this file can be embedded.
source	string	R		The URL to use to download the file fromSkyDrive.
upload_location	string	R		The URL to upload file content hosted in SkyDrive.
type	string	R		The type of object; "file" in this case.
created_time	string	R		The time, in ISO 8601 format, at which the file was created.
updated_time	string	R		The time, in ISO 8601 format, that the file was last updated.
shared_with	object	R		Object containing permission info.
access (shared_with object)	string	R		Info about who can access the folder. The options are: <ul style="list-style-type: none">• People I selected• Just me• Everyone (public)• Friends• My friends and their friends• People with a link The default is Just me .

To get properties for a **File** resource, make a **GET** request to **/FILE_ID** (the target file ID).

To download the content of a **File** resource, make a **GET** request to **/FILE_ID/content**. The file content is returned in the response body.

To return a download link for a **File** resource, make a **GET** request to **/FILE_ID/content** with the **suppress_redirects=true** parameter. The URL is returned in the response body in the following format.

```
{
  "location": "..."
}
```

Tip If the client has already requested the properties for a file, the download link will be in the **source** field.

To create a new **File** resource, you can either make a **POST** request to **/FOLDER_ID/files**, a **POST** request to the **/UPLOAD_LOCATION** for the target folder, or a **PUT** request to **/FOLDER_ID/files/<file name>**. For a **POST** request, you must format the request as a multipart/form-data as described in [RFC 2388](#). You must specify the Content-Type as multipart/form-data, and specify the boundary like this:

```
Content-Type: multipart/form-data; boundary=AaB03x
```

Provide the item to upload on a multipart section, and the name of the file as the value of the filename parameter of the Content-Disposition header, as shown here.

```
--AAB03x
Content-Disposition: file; filename="file1.txt"
Content-Type: text/plain
...contents of file1.txt...
--AAB03x
```

We only support one multipart section per request. For a **PUT** request, leave the Content-Type blank and put the contents of the file in the request body.

Upon successful creation, the location header points to the location for the newly created file, and the response body contains the following properties.

```
{
  "id": "ID of the new file",
  "source": "URL where the file can be downloaded from"
}
```

You can upload a file to an existing resource in any of the following ways.

- Make a **PUT** request to */UPLOAD_LOCATION/filename*.
- Make a **PUT** request to */FOLDER_ID/files/filename*.
- Make a **PUT** request to */UPLOAD_LOCATION* for the file to update.
- Make a **PUT** request to */FILE_ID/content*.

To update the properties for a **File** resource, make a **PUT** request to */FILE_ID*, and specify the changes in the request body as shown here.

```
{
  "name": "UFO_Sightings_1979"
}
```

To delete a file, make a **DELETE** request to */FILE_ID*.

To move a file, make a **MOVE** request to */FILE_ID* or */FOLDER_ID*, and specify the ID of the destination folder in the request body as shown here.

```
{
  "destination": "FOLDER_ID"
}
```

To copy a file, make a **COPY** request to */FILE_ID* or */FOLDER_ID*, and specify the ID of the destination folder in the request body as shown here.

```
{
  "destination": "FOLDER_ID"
}
```

For PUT and POST requests, you can use the Overwrite query string parameter to indicate whether the request should fail if the resource already exists, as shown here.

```
overwrite = true
or
overwrite = false
```

If the value of the overwrite query sting is set to **true**, and the file exists, it is overwritten. If the value is set to **false**, the file is not overwritten and an **resource_already_exists** error is returned. If no overwrite query string is specified, the default value is **true**.

[Top](#)

Folder object

The **Folder** object contains info about a user's folders in SkyDrive. Folders can contain combinations of photos, videos, and subfolders. The Live Connect REST API supports reading **Folder** objects. Use the **wl.photos** scope to read **Folder** objects. Use the **wl.contacts_photos** scope to read any albums, photos, and videos that other users have shared with the user.

The **Folder** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array container for Folder objects if a collection of objects is returned.
id	string	R		The Folder object's ID.
from	object	R		Info about the user who created the folder.
name (from object)	string	R		The name of the user who created the folder.
id (from object)	string	R		The ID of the user who created the folder.
name	string	RW	YES	The name of the folder.
description	string/null	RW		A description of the folder, or null if no description is specified.

count	number	R		The total number of items in the folder.
link	string	R		A URL of the folder, hosted in SkyDrive.
upload_location	string	R		The URL to upload items to the folder hosted in SkyDrive. Requires the wl.skydrive scope.
parent_id	string	R		The resource ID of the parent.
type	string	R		The type of object; "folder" in this case.
created_time	string	R		The time, in ISO 8601 format, at which the folder was created.
updated_time	string	R		The time, in ISO 8601 format, at which the contents of the folder were last updated.
shared_with	object	R		Permissions info for the folder. Requires the wl.skydrive scope.
access (shared_with object)	string	R		<p>Info about who can access the folder. The options are:</p> <ul style="list-style-type: none"> • People I selected • Just me • Everyone (public) • Friends • My friends and their friends • People with a link <p>The default is Just me.</p>

The following is an example of a collection of **Folder** objects. (For brevity, only the first object is shown.)

```
{
  "data": [
    {
      "id": "folder.8c8ce076ca27823f.8C8CE076CA27823F!142",
      "from": {
        "name": "Roberto Tamburello",
        "id": "8c8ce076ca27823f"
      },
      "name": "My Sample Folder in Album 1",
      "description": "",
      "parent_id": "folder.de57f4126ed7e411",
      "upload_location": "https://apis.live.net/v5.0/folder.de57f4126ed7e411.DE57F4126ED7E411!126/files/",
      "count": 3,
      "link": "https://cid-8c8ce076ca27823f.skydrive.live.com/redir.aspx?page\u003dself\u0026resid\u003d8C8CE076CA27823F!142\u0026par",
      "type": "folder",
      "created_time": "2011-04-22T00:36:30+0000",
      "updated_time": "2011-04-22T19:18:12+0000"
    },
    {
      ...
    }
  ]
}
```

To get the root **Folder** resource by using the Live Connect REST API, make a **GET** request to either /me/skydrive, or /*USER_ID*/skydrive.

To get a subfolder resource, make a **GET** request to /*FOLDER_ID*.

To enumerate the contents of a folder, make a **GET** request to /*FOLDER_ID*/files.

A folder may contain the following objects.

- [Folder object](#)
- [File object](#)
- [Album object](#)
- [Photo object](#)
- [Video object](#)

To create a new **Folder** resource, make a **POST** request to /*FOLDER_ID*. Pass the *name* and *description* attributes in the request body as shown here.

```
{
  "name": "Informative Spreadsheets",
  "description": "A folder full of useful data visualizations."
}
```

The folder will be created as a child of the folder specified in the request URL. The location header points to the newly created folder location, and the response body contains properties for the newly created folder.

To update the properties of a **Folder**, make a **PUT** request to /*FOLDER_ID*. Pass the attributes to update in the request body as shown here.

```
{
  "name": "Very Informative Spreadsheets",
  "description": "A folder full of even more useful data visualizations."
}
```

To delete a folder, make a **DELETE** request to `/FOLDER_ID`.

In the preceding examples, replace `FOLDER_ID` with the target folder ID.

[Top](#)

Friend object

The **Friend** object contains info about a user's friends. A **Friend** object represents a user's contact whose `is_friend` value is set to `true`. The Live Connect REST API supports reading **Friend** objects.

The **Friend** object contains the following structures.

Structure	Required scopes	Type	Description
data	<code>wl.basic</code>	array	An array container for Friend objects when a collection of objects is returned.
id	<code>wl.basic</code>	string	The friend's ID.
name	<code>wl.basic</code>	string	The friend's full name, formatted for location.

The following is an example of a collection of **Friend** objects. (For brevity, only the first object is shown.)

```
{
  "data": [
    {
      "id": "d09ea18fafc39a0c",
      "name": "Henrik Jensen"
    },
    ...
  ]
}
```

To get a collection of **Friend** objects by using the Live Connect REST API, make a **GET** request to `/me/friends`.

[Top](#)

Permissions object

The **Permissions** object contains a list of scopes, showing which scopes the user has consented to. The response body contains a JavaScript Object Notation (JSON) object listing all consented scopes as a name value pair. Each scope the user consented to will be present as a key.

The following is an example of a **Permissions** object. (For brevity, not all scopes are shown.)

```
{
  "wl.basic": 1,
  "wl.offline_access": 1,
  "wl.signin": 1,
  ...
}
```

To get a **Permissions** object by using the Live Connect REST API, make a **GET** request to `/me/permissions`.

[Top](#)

Photo object

The **Photo** object contains info about a user's photos on SkyDrive. The Live Connect REST API supports creating, reading, updating, and deleting **Photo** objects. Use the `wl.photos` scope to read **Photo** objects. Use the `wl.contacts_photos` scope to read any albums, photos, and videos that other users have shared with the user. Use the `wl.skydrive_update` scope to create, update, or delete **Photo** objects.

The **Photo** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array of Photo objects, when a collection of objects is returned.
id	string	R		The Photo object's ID.
from	object	R		Info about the user who uploaded the photo.
name (from object)	string	R		The name of the user who uploaded the photo. Part of the from object.
id (from object)	string	R		The ID of the user who uploaded the photo. Part of the from object.
name	string	RW	YES	The file name of the photo.

parent_id	string	R	The ID of the folder where the item is stored.
description	string/null	RW	A description of the photo, or null if no description is specified.
size	number	R	The size, in bytes, of the photo.
comments_count	number	R	The number of comments associated with the photo.
comments_enabled	true/false	R	Value that indicates whether comments are enabled for the photo. If comments can be made, this value is true ; otherwise, it is false .
tags_count	number	R	The number of tags on the photo.
tags_enabled	true/false	R	Indicates whether tags are enabled for the photo. If users can tag the photo, this value is true ; otherwise, it is false .
picture	string	R	A URL of the photo's picture.
source	string	R	The download URL for the photo. Warning This value is not persistent. Use it immediately after making the request, and avoid caching.
images	array	R	Info about various sizes of the photo.
height (images array)	number	R	The height, in pixels, of this image of this particular size.
width (images array)	number	R	The width, in pixels, of this image of this particular size.
source (images array)	string	R	A URL of the source file of this image of this particular size.
type (images array)	string	R	The type of this image of this particular size. Valid values are "normal", "album", and "thumbnail".
link	string	R	A URL of the photo, hosted in SkyDrive.
upload_location	string	R	The URL to upload photo content hosted in SkyDrive. This value is only returned if the wl.skydrive scope is present.
when_taken	string/null	R	The date, in ISO 8601 format, on which the photo was taken, or null if no date is specified.
height	number	R	The height, in pixels, of the photo.
width	number	R	The width, in pixels, of the photo.
type	string	R	The type of object; "photo" in this case.
created_time	string	R	The time, in ISO 8601 format, at which the photo was created.
updated_time	string	R	The time, in ISO 8601 format, at which the photo was last updated.
shared_with	object	R	The object containing permissions info for the photo.
access (shared_with object)	string	R	Info about who can access the photo. The options are: <ul style="list-style-type: none">• People I selected• Just me• Everyone (public)• Friends• My friends and their friends• People with a link The default is Just me .

The following is an example of a collection of **Photo** objects. (For brevity, only the first object is shown.)

{

```

"data": [
  {
    "id": "file.de57f4126ed7e411.DE57F4126ED7E411!128",
    "from": {
      "name": "Nuno Bento",
      "id": "de57f4126ed7e411"
    },
    "name": "Maui-2012_0034.JPG",
    "description": null,
    "parent_id": "folder.de57f4126ed7e411.DB57F4126ED7E411!126",
    "size": 561683,
    "comments_count": 1,
    "comments_enabled": true,
    "tags_count": 0,
    "tags_enabled": true,
    "picture": "http://storage.live.com/slpKk5vzd-gdPanbzKYhB0nQGn8wGq5DSggvrgIHU1NTXA4e2-spGkAhQjWld9pcgKAGLB4NsEsSvDoREmdx5w-JiFr",
    "source": "http://storage.live.com/slpKk5vzd-gdPanbzKYhB0nQGn8wGq5DSggvrgIHU1NTXA4e2-spGkAhQjWld9pcgKAGLB4NsEsSvDoREmdx5w-JiFr",
    "upload_location": "https://apis.live.net/v5.0/file.de57f4126ed7e411.DE57F4126ED7E411!128/content/",
    "images": [
      {
        "height": 450,
        "width": 600,
        "source": "http://storage.live.com/slpKk5vzd-gdPanbzKYhB0nQGn8wGq5DSggvrgIHU1NTXA4e2-spGkAhQjWld9pcgKAGLB4NsEsSvDoREmdx5w-JiFr",
        "type": "normal"
      },
      {
        "height": 132,
        "width": 176,
        "source": "http://storage.live.com/slpKk5vzd-gdPanbzKYhB0nQGn8wGq5DSggvrgIHU1NTXA4e2-spGkAhQjWld9pcgKAGLB4NsEsSvDoREmdx5w-JiFr",
        "type": "album"
      },
      {
        "height": 72,
        "width": 96,
        "source": "http://storage.live.com/slpKk5vzd-gdPanbzKYhB0nQGn8wGq5DSggvrgIHU1NTXA4e2-spGkAhQjWld9pcgKAGLB4NsEsSvDoREmdx5w-JiFr",
        "type": "thumbnail"
      },
      {
        "height": 1200,
        "width": 1600,
        "source": "http://storage.live.com/slpKk5vzd-gdPanbzKYhB0nQGn8wGq5DSggvrgIHU1NTXA4e2-spGkAhQjWld9pcgKAGLB4NsEsSvDoREmdx5w-JiFr",
        "type": "full"
      }
    ],
    "link": "https://skydrive.live.com/redir.aspx?cid\u003dde57f4126ed7e411\u0026page\u003dview\u0026resid\u003dDE57F4126ED7E411!128",
    "when_taken": null,
    "height": 1200,
    "width": 1600,
    "type": "photo",
    "shared_with": {
      "access": "Everyone (public)"
    },
    "created_time": "2012-12-03T18:14:03+0000",
    "updated_time": "2012-12-03T18:31:01+0000"
  },
  ...
]
}

```

To get a collection of **Photo** objects by using the Live Connect REST API, make a **GET** request to `/ALBUM_ID/files`.

Note You can also get a collection of **Photo** objects by calling either `/ALBUM_ID/photos`, or `/FOLDER_ID/photos`.

Note To display a photo, call `/PHOTO_ID/picture`. To display a photo of a specific image size, follow `/PHOTO_ID/picture` with `?type=` and one of the following: **thumbnail**, **small**, **album**, **normal**, or **full**. (The default is **normal**.) The following is an example of a call that displays a thumbnail image.

```
https://apis.live.net/v5.0/PHOTO_ID/picture?type=thumbnail&access_token=ACCESS_TOKEN
```

In the preceding examples, replace `ALBUM_ID` and `PHOTO_ID` with the target album or photo ID.

To create a new **Photo** resource, you can either make a **POST** request to `/FOLDER_ID`, or make a **POST** request to the `/UPLOAD_LOCATION` for the target folder. You must format the request as a multipart/form-data as described in [RFC 2388](#). You must specify the Content-Type as multipart/form-data, and specify the boundary like this:

```
Content-Type: multipart/form-data; boundary=AaB03x
```

Provide the item to upload on a multipart section, and the name of the photo as the value of the filename parameter of the Content-Disposition header, as shown here.

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: file; filename="hovering_orbs.jpg"
Content-Type: image/jpeg
...contents of hovering_orbs.jpg...
--AaB03x
```

We only support one multipart section per request. For a **PUT** request, leave the Content-Type blank and put the contents of the file in the request body.

Upon successful creation, the location header points to the location for the newly created photo, and the response body contains the following properties.

```
{
  "id": "ID of the new photo",
  "source": "URL where the photo can be dowloaded from"
}
```

You can upload a photo to an existing resource in any of the following ways.

- Make a **PUT** request to `/UPLOAD_LOCATION/filename` for the folder you would like to upload to..

- Make a **PUT** request to `/FOLDER_ID/files`.
- Make a **PUT** request to `/ALBUM_ID/files`.
- Make a **PUT** request to `/UPLOAD_LOCATION` for the file to update.
- Make a **PUT** request to `/PHOTO_ID/content`.

To update the properties for a **Photo** resource, make a **PUT** request to `/PHOTO_ID`, and specify the changes in the request body as shown here.

```
{
  "name": "fast-moving_orbs.jpg"
}
```

To delete a photo, make a **DELETE** request to `/PHOTO_ID`.

To move a photo, make a **MOVE** request to `/PHOTO_ID` or `/FOLDER_ID`, and specify the ID of the destination folder in the request body as shown here.

```
{
  "destination": "FOLDER_ID"
}
```

To copy a photo, make a **COPY** request to `/PHOTO_ID` or `/FOLDER_ID`, and specify the ID of the destination folder in the request body as shown here.

```
{
  "destination": "FOLDER_ID"
}
```

For **PUT** and **POST** requests, you can use the Overwrite header to indicate whether the request should fail if the resource already exists, as shown here. Note that **MOVE** and **COPY** do not support overwriting at this time.

`Overwrite: (true | false)`

If the value of the overwrite header is set to **true**, and the file exists, it is overwritten. If the value is set to **false**, the file is not overwritten and an **resource_already_exists** error is returned. If no overwrite header is specified, the default value is **true**.

[Top](#)

Tag object

The **Tag** object contains info about tags associated with a photo or a video on SkyDrive. The Live Connect REST API supports reading **Tag** objects. Use the **wl.photos**, and **wl.skydrive** scopes to read **Tag** objects. Use the **wl.contacts_photos** and **wl.contacts_skydrive** scopes to read the **Tag** objects that are associated with any photos that other users have shared with the user.

The **Tag** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array of Tag objects if a collection of objects is returned.
name	string	RW		The name of the tagged person.
user_id	string	R		The Tag object's ID.
text	string	RW		The text of the tag.
x	number	RW		The center of the tag's horizontal position, measured as a floating-point percentage from 0 to 100, from the left edge of the photo. This value is not returned for Video objects.
y	number	RW		The center of the tag's vertical position, measured as a floating-point percentage from 0 to 100, from the top edge of the photo. This value is not returned for Video objects.
created_time	string	R		The time, in ISO 8601 format, at which the tag was created.

The following is an example of a collection of **Tag** objects. (For brevity, only the first object is shown.)

```
{
  "user_id": "tag.48f295ff17612208.48F295FF17612208!124.oneZnop0AZ5opoxXVg=c5vbB6FU",
  "user": {
    "name": "Roberto Tamburello",
    "id": "8c8ce076ca27823f"
  },
  "x": 43.8986,
  "y": 54.4138,
  "created_time": "2011-04-22T01:17:00+0000"
}
```

Note

Information about the tagged user is stored within the **user** attribute.

The **x** and **y** members are not present on tags returned from a **video** object.

To get a collection of **Tag** objects by using the Live Connect REST API, make a **GET** request to `/RESOURCE_ID/tags`.

[Top](#)

User object

The **User** object contains info about a user. The Live Connect REST API supports reading **User** objects.

The **User** object contains the following structures.

Structure	Required scopes	Type	Description
id	None	string	The user's ID.
name	None	string	The user's full name.
first_name	None	string/null	The user's first name.
last_name	None	string/null	The user's last name.
link	wl.basic	string	The URL of the user's profile page.
birth_day	wl.birthday	number/null	The day of the user's birth date, or null if no birth date is specified.
birth_month	wl.birthday	number/null	The month of the user's birth date, or null if no birth date was specified.
birth_year	wl.birthday	number/null	The year of the user's birth date, or null if no birth date is specified.
work	wl.work_profile	array	An array containing the user's work info.
employer (work array)	wl.work_profile	object	Info about the user's employer.
name (employer object)	wl.work_profile	string/null	The name of the user's employer, or null if the employer's name is not specified.
position (work array)	wl.work_profile	object	Info about the user's work position.
name (position object)	wl.work_profile	string/null	The name of the user's work position, or null if the name of the work position is not specified.
gender	None	string/null	The user's gender. Valid values are "male", "female", or null if the user's gender is not specified.
emails	wl.emails	object	The user's email addresses.
preferred (emails object)	wl.emails	string/null	The user's preferred email address, or null if one is not specified.
account (emails object)	wl.emails	string	The email address associated with the account.
personal (emails object)	wl.emails	string/null	The user's personal email address, or null if one is not specified.
business (emails object)	wl.emails	string/null	The user's business email address, or null if one is not specified.
addresses	wl.postal_addresses	object	The user's postal addresses.
personal (addresses object)	wl.postal_addresses	object	The user's personal postal address.
street (personal object)	wl.postal_addresses	string/null	The user's personal street address, or null if one is not specified.
street_2 (personal object)	wl.postal_addresses	string/null	The second line of the user's personal street address, or null if one is not specified.
city (personal object)	wl.postal_addresses	string/null	The city of the user's personal address, or null if one is not specified.

state (personal object)	wl.postal_addresses	string/null	The state of the user's personal address, or null if one is not specified.
postal_code (personal object)	wl.postal_addresses	string/null	The postal code of the user's personal address, or null if one is not specified.
region (personal object)	wl.postal_addresses	string/null	The region of the user's personal address, or null if one is not specified.
business(addresses object)	wl.postal_addresses	object	The user's business postal address.
street (business object)	wl.postal_addresses	string/null	The user's business street address, or null if one is not specified.
street_2 (business object)	wl.postal_addresses	string/null	The second line of the user's business street address, or null if one is not specified.
city (business object)	wl.postal_addresses	string/null	The city of the user's business address, or null if one is not specified.
state (business object)	wl.postal_addresses	string/null	The state of the user's business address, or null if one is not specified.
postal_code (business object)	wl.postal_addresses	string/null	The postal code of the user's business address, or null if one is not specified.
region (business object)	wl.postal_addresses	string/null	The region of the user's business address, or null if one is not specified.
phones	wl.phone_numbers	object	The user's phone numbers.
personal (phones object)	wl.phone_numbers	string/null	The user's personal phone number, or null if one is not specified.
business (phones object)	wl.phone_numbers	string/null	The user's business phone number, or null if one is not specified.
mobile (phones object)	wl.phone_numbers	string/null	The user's mobile phone number, or null if one is not specified.
locale	None	string	The user's locale code.
updated_time	wl.basic	string	The time, in ISO 8601 format, at which the user last updated the object.

The following is an example of a **User** object.

```
{
  "id": "8c8ce076ca27823f",
  "name": "Roberto Tamburello",
  "first_name": "Roberto",
  "last_name": "Tamburello",
  "link": "http://cid-8c8ce076ca27823f.profile.live.com/",
  "birth_day": 20,
  "birth_month": 4,
  "birth_year": 2010,
  "work": [
    {
      "employer": {
        "name": "Microsoft Corporation"
      },
      "position": {
        "name": "Software Development Engineer"
      }
    }
  ],
  "gender": "male",
  "emails": {
    "preferred": "Roberto.Tamburello@contoso.com",
    "account": "Roberto.Tamburello@live.com",
    "personal": "Roberto.Tamburello@contoso.com",
    "business": "r_tamburello@example.com"
  },
  "addresses": {
    "personal": {
      "street": "123 Main St.",
      "street_2": "Apt. A",
      "city": "Redmond",
      "state": "WA",
      "postal_code": "12990",
      "country": "USA"
    }
  }
}
```

```

        "region": "United States"
    },
    "business": {
        "street": "456 Anywhere St.",
        "street_2": "Suite 1",
        "city": "Redmond",
        "state": "WA",
        "postal_code": "12399",
        "region": "United States"
    }
},
"phones": {
    "personal": "(555) 555-1212",
    "business": "(555) 111-1212",
    "mobile": null
},
"locale": "en_US",
"updated_time": "2011-04-21T23:55:34+0000"
}

```

To get a **User** object by using the Live Connect REST API, make a **GET** request to either /me, or /USER_ID.

Note To redirect a **GET** call to the URL of a user's picture, you can call /me/picture or /USER_ID/picture.

[Top](#)

Video object

The **Video** object contains info about a user's videos on SkyDrive. The Live Connect REST API supports creating, reading, updating, and deleting **Video** objects. Use the **wl.photos** scope to read **Video** objects. Use the **wl.contacts_photos** scope to read albums, photos, and videos that other users have shared with the user. Use the **wl.skydrive_update** scope to create, update, or delete **Video** objects.

The **Video** object contains the following structures.

Structure	Type	R/W	Required on Create	Description
data	array	R		An array of Video objects, if a collection of objects is returned.
id	string	R		The Video object's ID.
from	object	R		Info about the user who uploaded the video.
name (from object)	string	R		The name of the user who uploaded the video.
id (from object)	string	R		The ID of the user who uploaded the video.
name	string	RW	YES	The file name of the video.
parent_id	string	R		The id of the folder where the item is stored.
description	string/null	RW		A description of the video, or null if no description is specified.
size	number	R		The size, in bytes, of the video.
comments_count	number	R		The number of comments associated with the video.
comments_enableddd	true/false	R		A value that indicates whether comments are enabled for the video. If comments can be made, this value is true ; otherwise, it is false .
tags_count	number	R		The number of tags on the video.
tags_enabled	true/false	R		A value that indicates whether tags are enabled for the video. If tags can be set, this value is true ; otherwise, it is false .
picture	string	R		A URL of a picture that represents the video.
source	string	R		The download URL for the video. Warning This value is not persistent. Use it immediately after making the request, and avoid caching.
link	string	R		A URL of the video, hosted in SkyDrive.
upload_location	string	R		The URL to upload video content hosted in SkyDrive. This value is only returned if the wl.skydrive scope is present.
height	number	R		The height, in pixels, of the video.
width	number	R		The width, in pixels, of the video.

duration	number	R		The duration, in milliseconds, of the video run time.
bitrate	number	R		The video's bit rate, in bits per second.
type	string	R		The type of object; "video" in this case.
created_time	string	R		The time, in ISO 8601 format, at which the video was created.
updated_time	string	R		The time, in ISO 8601 format, that the video was last updated.
shared_with	object	R		The object containing permission info.
access (shared_with object)	string	R		<p>Info about who can access the file. The options are:</p> <ul style="list-style-type: none"> • People I selected • Just me • Everyone (public) • Friends • My friends and their friends • People with a link <p>The default is Just me.</p>

The following is an example of a collection of **Video** objects. (For brevity, only the first object is shown.)

```
{
  "data": [
    {
      "id": "file.de57f4126ed7e411.DE57F4126ED7E411!135",
      "from": {
        "name": "Nuno Bento",
        "id": "de57f4126ed7e411"
      },
      "name": "Wildlife.wmv",
      "description": null,
      "parent_id": "folder.de57f4126ed7e411.DE57F4126ED7E411!126",
      "size": 26246026,
      "comments_count": 0,
      "comments_enabled": true,
      "tags_count": 0,
      "tags_enabled": true,
      "picture": "http://storage.live.com/slpKk5vzd-gdPaJ5Q1MKN34itsyRlUkAYzD_zsr0Dg-5r4bH8Qo8XRgsuna0M-V4G-XPpu1spowx4xwfjCuDcWQVa7",
      "source": "http://storage.live.com/slpKk5vzd-gdPaJ5Q1MKN34itsyRlUkAYzD_zsr0Dg-5r4bH8Qo8XRgsuna0M-V4G-XPpu1spowx4xwfjCuDcWQVa7&upload_location": "https://apis.live.net/v5.0/file.de57f4126ed7e411.DE57F4126ED7E411!135/content/",
      "link": "https://skydrive.live.com/redir.aspx?cid\u003dde57f4126ed7e411\u0026page\u003dview\u0026resid\u003dDE57F4126ED7E411!126&height": 720,
      "width": 1280,
      "duration": 30093,
      "bitrate": 5942130,
      "type": "video",
      "shared_with": {
        "access": "Everyone (public)"
      },
      "created_time": "2011-08-23T23:41:18+0000",
      "updated_time": "2011-08-23T23:41:32+0000"
    },
    ...
  ]
}
```

To get a collection of **Video** objects by using the Live Connect REST API, make a **GET** request to `/ALBUM_ID/files`.

Note You can also get a collection of **Video** objects by calling either `/ALBUM_ID/videos`, or `/FOLDER_ID/videos`.

In the preceding examples, replace `ALBUM_ID` with the target Live Connect album ID.

To create a new **Video** resource, you can either make a **POST** request to `/FOLDER_ID`, or make a **POST** request to the `/UPLOAD_LOCATION` for the target folder. You must format the request as a multipart/form-data as described in [RFC 2388](#). You must specify the Content-Type as multipart/form-data, and specify the boundary like this:

`Content-Type: multipart/form-data; boundary=AaB03x`

Provide the item to upload on a multipart section, and the name of the photo as the value of the filename parameter of the Content-Disposition header, as shown here.

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: file; filename="strange_lights.avi"
Content-Type: video/x-msvideo
...contents of strange_lights.avi...
--AaB03x
```

We only support one multipart section per request. For a **PUT** request, leave the Content-Type blank and put the contents of the file in the request body.

Upon successful creation, the location header points to the location for the newly created video, and the response body contains the

following properties.

```
{  
    "id": "ID of the new video",  
    "source": "URL where the video can be downloaded from"  
}
```

You can upload a video to an existing resource in any of the following ways.

- Make a **PUT** request to */UPLOAD_LOCATION/filename*.
- Make a **PUT** request to */FOLDER_ID/files*.
- Make a **PUT** request to */ALBUM_ID/files*.
- Make a **PUT** request to */UPLOAD_LOCATION* for the file to update.
- Make a **PUT** request to */VIDEO_ID/content*.

To update the properties for a **Video** resource, make a **PUT** request to */VIDEO_ID*, and specify the changes in the request body as shown here.

```
{  
    "name": "triangle_formation.avi"  
}
```

To delete a video, make a **DELETE** request to */VIDEO_ID*.

To move a video, make a **MOVE** request to */VIDEO_ID* or */FOLDER_ID*, and specify the ID of the destination folder in the request body as shown here.

```
{  
    "destination": "FOLDER_ID"  
}
```

To copy a video, make a **COPY** request to */VIDEO_ID* or */FOLDER_ID*, and specify the ID of the destination folder in the request body as shown here.

```
{  
    "destination": "FOLDER_ID"  
}
```

For PUT and POST requests, you can use the Overwrite header to indicate whether the request should fail if the resource already exists, as shown here.

Overwrite: (true | false)

If the value of the overwrite header is set to **true**, and the file exists, it is overwritten. If the value is set to **false**, the file is not overwritten and an **resource_already_exists** error is returned. If no overwrite header is specified, the default value is **true**.

[Top](#)

Additional URL parameters

You can specify additional parameters, as part of the URL of a **GET** request to the Live Connect REST API, to instruct Live Connect API how to return its results. The following table explains these parameters.

Parameter	Description	Valid values	Default value	Examples
pretty	When this value is set to true , Live Connect returns the response in a more human-readable format. When this value is set to false , white space and new lines are not included, which improves performance but makes the response less readable.	true/false	true	pretty=false
callback	The name of the function that is to handle the JSON with padding (JSONP) function call. When this parameter is specified, all error responses use the HTTP status code 200 to help ensure proper handling in the web browser.	A string that represents the name of the callback function.	None	callback=onResponse
suppress_response_codes	When this value is set to true , Live Connect returns the HTTP status code 200 for all error responses. This option is useful with web-browser plug-ins like Microsoft Silverlight or Adobe Flash Player.	true/false	false	suppress_response_code=true
method	The HTTP method to be used for request processing instead of GET .	A string that represents the HTTP request method, such as POST .	None	method=POST
return_ssl_resources	When this value is set to true , Live Connect uses HTTPS to return the response. This is useful to prevent web-browser warnings when an originating website also uses HTTPS.	true/false	false	return_ssl_resources=true



For example, the following URL makes a **GET** request for the user's contacts list, with the **pretty** parameter set to **false**:

`https://apis.live.net/v5.0/me/contacts?pretty=false&access_token=ACCESS_TOKEN`

[Top](#)

© 2011 Microsoft Corporation. All rights reserved.

JavaScript API

[This documentation is preliminary and is subject to change.]

The Live Connect JavaScript API, together with the [REST API](#), enables apps to read, update, and share user data by using the JavaScript programming language. The JavaScript API provides methods for signing users in and out, getting user status, subscribing to events, creating UI controls, and calling the Representational State Transfer (REST) API.

The **WL** object is a global object that encapsulates all functions of the JavaScript API. Your app uses the **WL** object to call all of the JavaScript API functions, which enable it to sign users in and out, subscribe to and unsubscribe from events, create UI elements, and even make calls to the REST API.

This section contains the following topics.

Topic	Description
Using the JavaScript Library	Shows how to include the Live Connect JavaScript API in your app.
WL.api	Enables you to make calls to the Live Connect REST API.
WL.download	Makes a call to download a file from SkyDrive.
WL.Event.subscribe	Subscribes to an event.
WL.Event.unsubscribe	Unsubscribes from an event.
WL.getLoginStatus	Gets the user's status as he or she is signing in.
WL.getSession	Gets the session object for the user.
WL.init	Initializes the JavaScript API.
WL.login	Signs a user in or requests additional permissions.
WL.logout	Signs a user out.
WL.ui	Enables you to add UI elements, such as a sign-in control, to your app.
WL.upload	Makes a call to upload a file to SkyDrive.
JSONP Scenarios	Shows how to use <i>JavaScript Objection Notation with padding</i> (JSONP) in client-side JavaScript

webpage code to interact directly with the Live Connect REST API.

© 2011 Microsoft Corporation. All rights reserved.

Using the JavaScript library

[This documentation is preliminary and is subject to change.]

This topic covers the basic tasks involved in using the Live Connect JavaScript library in web apps and Metro style apps using JavaScript.

- [Referencing the JavaScript library in a web app](#)
- [Referencing the JavaScript library in a Metro style app using JavaScript](#)
- [Loading the debug version](#)
- [International support](#)
- [Live Connect authentication cookies](#)

Referencing the JavaScript library in a web app

To use the Live Connect JavaScript API with a web-based app, you must load the JavaScript library (wl.js) by adding a reference in your code to the location of the latest version of the library file. The United States English version of the JavaScript library is loaded by default. The following is an example of how to reference the JavaScript library in a standard web-based app.

```
<script src="//js.live.net/v5.0/wl.js"></script>
```

Referencing the JavaScript library in a Metro style app using JavaScript

To reference the Live Connect APIs from a Microsoft Visual Studio 11 Express for Windows Developer Preview project, do this:

1. Install the [Live Software Development Kit \(SDK\)](#), if you have not already done so.
2. In your project, in **Solution Explorer**, right-click **References** > **Add Reference**.
3. Click **Windows** > **Extension SDKs** > **Live SDK for Metro style HTML Applications**.
4. Click **Add**, and then click **Close**.
5. If you want to enable Microsoft IntelliSense in the default.html file, add this `<script>` element within the `<head>` element.

```
<script src="ms-wwa:///LiveSDKHTML.5.0/js/wl.js"></script>
```

6. If you want to enable IntelliSense in any other JavaScript (.js) file in your project, add this comment to the top of that .js file.

```
/// <reference path="ms-wwa:///LiveSDKHTML.5.0/js/wl.js" />
```

[Top](#)

Loading the debug version

If you need to debug the JavaScript API, you can reference an uncompressed version of the library file, named wl.debug.js. Use the following code to reference the debugging version in a web app.

```
<script src="//js.live.net/v5.0/wl.debug.js"></script>
```

If you are creating a Metro style app using JavaScript, the solution configuration that is selected in Visual Studio (for example, **debug** or **release**) determines which version of wl.js will be loaded.

International support

Specifying a locale in a Metro style app using JavaScript must be done programmatically, as shown in the following example.

```
...
var ns = Windows.ApplicationModel.Resources.Core;
var rm = ns.ResourceManager;
var context = rm.current.defaultContext;
context.language = "fr";
...
```

For web apps, specifying a locale can help to improve the performance of your app, by ensuring that the JavaScript library is loaded from the server that is closest in geographic proximity. To load a different localized version of the JavaScript library, you must add to the URL a segment that specifies the culture name string for the localized version that you want. Insert this segment immediately after the version number segment. For example, to load the Japanese version of the JavaScript library, you could use the following code.

```
<script src="//js.live.net/v5.0/ja/wl.js"></script>
```

For a complete list of localized versions, see [Supported locales](#).

Live Connect authentication cookies

The JavaScript library uses a session cookie, wl_auth, to track and transmit OAuth authentication data as the user signs in to Live Connect. For more info about OAuth authentication, see [OAuth 2.0](#).

When the JavaScript library is loaded as part of the OAuth redirect Uniform Resource Identifier (URI), it sets this cookie automatically. When it is loaded as part of a normal page, the library polls the cookie to determine when the sign-in state of the user changes.

The following table lists the data that is stored in subkeys in the wl_auth cookie.

Subkey	Data
access_token	The OAuth access token.
authentication_token	The authentication token.

error	The OAuth error, if one occurred.
error_description	The description of the OAuth error, if one occurred.
expires_in	The time remaining, in seconds, until the OAuth access token expires, if the token was provided.
scope	The OAuth scope value, if it was provided.

Note The wl_auth cookie is stored in the domain as the domain of your redirect URL.

Important Cookies are not written for Metro style apps using JavaScript.

In general, you do not have to handle the wl_auth cookie directly; the JavaScript library handles it for you. However, if your callback page processes the user's authentication token through server-side code, such as with the **wl.offline_access** scope, your code is responsible for updating the cookie according to the following rules:

- If the app successfully retrieves the authentication token, it must write the **access_token**, **scope**, and **expires_in** subkeys in the cookie that was received.
- If an error is received from the authentication server, the app must write two subkeys: **error** and **error_description**. The value of the **error** subkey must be "access_denied". The value of the **error_description** subkey must be "Failed to retrieve user access token".
- The callback page must load wl.js.

In all cases, your app must not overwrite any existing subkeys.

WL.api function

[This documentation is preliminary and is subject to change.]

Makes a call to the Live Connect REST API. This method encapsulates a REST API request and then calls a callback function to process the response.

Parameters

properties

Required. A JSON object that contains the following properties, which are necessary to make the REST API call:

Name	Type	Description	Default Value
path	string	Required. Contains the path to the REST API object. For information on specifying paths for REST objects, see REST API .	None
method	string	Optional. An HTTP method that specifies the action required for the API call. These actions are standard REST API actions: COPY , GET , MOVE , PUT , POST , and DELETE .	GET
body	object	Optional. A JSON object that specifies the REST API request body. The body property is used only for POST and PUT requests.	None

callback

Required. Specifies a callback function that is executed when the REST API call is complete. The callback function takes the API response object as a parameter. The response object exposes the data returned from Live Connect, or, if an error occurs, an error property that contains the error code.

The following example demonstrates how to use the API callback to determine whether a GET call was successful.

```
WL.api(
{
    path: "me",
    method: "GET"
},
function (response) {
    if (!response.error) {
        strGreeting = "Hi, " + response.first_name + "!"
        document.getElementById("greeting").innerHTML = strGreeting;
    }
});
```

Return value

This function can return the following errors:

- WL.api: The input parameter 'path' must be included.
- WL.api: The input parameter 'method' is not valid.

Examples

```
WL.api(  
{  
    path: "me",  
    method: "GET"  
},  
function (response) {  
    if (!response.error) {  
        strGreeting = "Hi, " + response.first_name + "!"  
        document.getElementById("greeting").innerHTML = strGreeting;  
    }  
});
```

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

WL.download function

[This documentation is preliminary and is subject to change.]

Makes a call to download a file from SkyDrive.

Important WL.download is only supported for use with Metro style apps using JavaScript. If you are writing a web app, use [WL.api](#) instead.

Parameters

properties

Required. A JSON object that contains the following properties, which are necessary to make the REST API call:

Name	Type	Description	Default Value
path	string	Required. The path to the file to download. For information on specifying paths for REST objects, see REST API .	None
file_output	StorageFile	Required. The file output object to write the downloaded file data to.	None

callback

Required. Specifies a callback function that is executed when the REST API call is complete. The callback function takes the API response object as a parameter. The response object exposes the data returned from Live Connect, or, if an error occurs, an error property that contains the error code.

Return value

This function does not return a value.

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

© 2011 Microsoft Corporation. All rights reserved.

WL.Event.subscribe function

[This documentation is preliminary and is subject to change.]

Adds a handler to an event.

Parameters

event

Required. The name of the event to which to add a handler. The JavaScript API supports the following events:

Name	Occurs when
auth.login	The user completes the sign-in process.
auth.logout	The user completes the sign-out process.
auth.sessionChange	The user's access token has changed.
auth.statusChange	The user's status has changed.
wl.log	An error has occurred with an API call.

callback

Required. Specifies the name of the callback function to handle the event.

Return value

This function can return the following errors:

- WL.Event.subscribe: The input parameter 'callback' must be included.
- WL.Event.subscribe: The input parameter 'event' is not valid.

Examples

```
WL.Event.subscribe("auth.login", onLogin);
```

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

© 2011 Microsoft Corporation. All rights reserved.

WL.Event.unsubscribe function

[This documentation is preliminary and is subject to change.]

Removes a handler from an event.

Parameters

event

Required. The name of the event from which to remove a handler. The JavaScript API supports the following events:

Name	Occurs when
auth.login	The user completes the sign-in process.
auth.logout	The user completes the sign-out process.
auth.sessionChange	The user's access token has changed.
auth.statusChange	The user's status has changed.
wl.log	An error has occurred with an API call.

callback

Optional. Removes the callback function from the event. If this parameter is omitted or **null**, all callback functions registered to the event are removed.
Removes the callback function from the specified event.

Examples

```
WL.Event.unsubscribe("auth.logout");
```

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

WL.getLoginStatus function

[This documentation is preliminary and is subject to change.]

Returns the login status of the current user. If the user is signed in and connected to your app, this function returns the session object. This is an asynchronous function that returns the user's status by contacting the Live Connect OAuth server. If the *callback* parameter is missing, execution continues and no error is logged.

Parameters

callback

Optional. Specifies the name of a callback function to execute when the user's login status is retrieved. This function takes the **status** object as a parameter.

force

Optional. If set to **true**, the function contacts the Live Connect authentication web service to determine the user's status. If set to **false** (default), the function can return the user status currently in memory, if there is one. If the user's status has already been retrieved, the library can return the cached value. However, you can force the library to retrieve up-to-date status from Live Connect by setting the *force* parameter to **true**.

Return value

Returns a **status** object, which contains the user's sign-in status and the session object. This object contains the following properties:

Property	Type	Description
status	enum	The sign-in status of the user. Valid values are "connected", "notConnected", or "unknown".
session	object	A JSON object that contains the properties of the current session. For a detailed description of the session object's properties, see the "Return Values" section of WL.getSession .

This function does not return any errors. If the *callback* parameter is missing, execution continues and no error is logged.

Examples

```
function loginStatus() {
    WL.getLoginStatus(function(response) { alert("Your status is: " + response.status) });
}
```

Requirements

Header	Not applicable
Library	WI.js
DLL	Not applicable

© 2011 Microsoft Corporation. All rights reserved.

WL.getSession function

[This documentation is preliminary and is subject to change.]

Retrieves the current session object synchronously, if a session object exists. For situations in which performance is critical, such as page loads, use the asynchronous [WL.getLoginStatus](#) method instead.

Parameters

This function has no parameters.

Return value

Returns the current session as a **session** object instance. The **session** object contains the following properties:

Property	Type	Description
access_token	string	The user's access token.
authentication_token	string	The authentication token.
scope	Array of string values	A list of scopes that the app has requested and that the user has consented to.
expires_in	number	The amount of time remaining, in seconds, until the user's access token expires.
expires	number	The exact time when the session will expire. This time is expressed in the number of seconds since 1 January, 1970.

Examples

The following example uses an event handler, `onSessionChange`, to handle the **auth.sessionChange** event. When called, `onSessionChange` calls **WL.getSession** to get the session object.

```
function onSessionChange() {
    var session = WL.getSession();
    if (session) {
        alert("Your session has changed.");
    }
}
```

Requirements

Header	Not applicable
Library	WI.js
DLL	Not applicable

© 2011 Microsoft Corporation. All rights reserved.

WL.init function

[This documentation is preliminary and is subject to change.]

Initializes the JavaScript library. An application must call this function before making other function calls in the library except for subscribing to events. Your app must call this function on every page before making other function calls in the library. The app should make function calls that subscribe to events before calling **WL.init**. If the JavaScript library has already been initialized on the page, calling this function succeeds silently; the client_id and redirect_uri parameters are not validated.

Parameters

properties

Required. A JSON object that must be formatted with the following properties:

Property	Type	Description	Default Value
client_id	string	Web app: required Metro style apps using JavaScript: optional Specifies your app's OAuth client ID (for web apps), or Package Security Identifier (SID) (for Metro style apps using JavaScript).	None.
redirect_uri	string	Optional. Contains the default redirect URI to be used for OAuth authentication. Live Connect uses this property to return info to your app. The OAuth server redirects to this URI during the OAuth flow. Important For Metro style apps using JavaScript, this property is only required if response_type is set to code .	The URL of the page from which the user clicked a sign-in button or link.
logging	boolean	Optional. If set to true , the library logs error info to the web browser console and notifies your app by means of the wl.log event.	true

status	boolean	Optional. If set to true , the library attempts to retrieve the user's sign-in status from Live Connect.	true
response_type	string	Optional. Specifies the OAuth response type value. If set to "token", the client receives the access token directly. If set to code , the client receives an authorization code, and the app server that serves the redirect_uri page should handle retrieving the access_token from the OAuth server by using the authorization code and client secret.	"token"

Return value

This function returns the following errors:

- WL.init: The input parameter 'client_id' must be included.
- WL.init: The input parameter 'properties' must be included.

Examples

```
WL.init({
  client_id: APPLICATION_CLIENT_ID,
  redirect_uri: REDIRECT_URL,
  response_type: "token"
});
```

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

WL.login function

[This documentation is preliminary and is subject to change.]

Signs the user in or expands the user's permission set. Because this function can result in launching the consent page prompt, you should call it only in response to a user action such as clicking a button. Otherwise, the user's web browser might block the popup.

This function is typically used by apps that define their own sign-in controls, or by apps that ask users to grant additional permissions during an activity. For example, to enable a user to post his or her status to Live Connect, your app may have to prompt the user for permission and call this function with an expanded scope.

If you call this function when the user has already consented to the requested scope and is already signed in, the callback function is invoked immediately with the current session.

This function logs errors to the browser console.

Parameters

properties

Required. A JSON object formatted with the following properties:

Name	Type	Description	Default Value
redirect_uri	string	Optional. Contains the redirect URI to be used for OAuth authentication. This value overrides the default redirect URI that is provided in the call to WL.init .	The redirect_uri value specified in WL.init
scope	string or array of string values	Required. Specifies the scopes to which the user who is signing in consents. The value can be an array of scope string values or a single string value of multiple scopes delimited by space characters. If you do not provide a value, the user is signed in only if he or she has previously given consent, and the user is signed in with only the scope to which he or she previously consented. The offline_access scope cannot be requested by using this function or the REST API; it must be requested by using the sign-In control, which in turn	None

invokes the consent dialog box.

callback

Optional. Specifies a callback function to execute when sign-in is complete. The callback function takes the **status** object as a parameter. For a description of the **status** object, see [WL.getLoginStatus](#). If you do not specify a callback function, your app can still get the sign-in callback info by listening for an **auth.sessionChange** or **auth.statusChange** event.

Examples

The following example demonstrates how to sign in a user.

```
var scopesArr = ["wl.signin"];
function signUserIn() {
    WL.login({ scope: scopesArr });
}
document.write("<a href='/' onclick='signUserIn()'>Don't like signin controls? Click here instead!</a>");
```

The following example demonstrates how to request a new scope for a user.

```
function addScope() {
    scopesArr.push("wl.photos");
    WL.login({ scope: scopesArr });
}
```

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

WL.logout function

[This documentation is preliminary and is subject to change.]

Signs the user out of Live Connect and clears any user state that is maintained by the JavaScript library, such as cookies. This function is useful primarily for websites that are do not use the sign-in control.

Parameters

callback

Optional. Specifies a callback function that is executed when sign-out is complete. The callback function takes the **status** object as a parameter. For a description of the **status** object, see [WL.getLoginStatus](#). If you do not specify a callback function, your app can still get the sign-out callback info by listening for an **auth.sessionChange** or **auth.statusChange** event.

Examples

```
function signUserOut() {  
    WL.logout();  
}  
document.write("<a href='/' onclick='signUserOut()'>Click here to sign out!</a>"  
);
```

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

WL.ui function

[This documentation is preliminary and is subject to change.]

Creates a user interface control on the current page. Currently, only the sign-in control is supported.

Parameters

properties

Required. A JSON object containing the following properties for creating the user interface element:

Name	Type	Description	Default Value
name	string	Required. Specifies name of the UI element to create. Currently, the only valid value for this property is "signin".	None.
element	string	Optional. The DOM element to which to attach the UI element, if applicable.	None.

If the value of the UI **name** property is "signin", the properties object can also contain the following:

Name	Type	Description	Default Value
brand	string	Optional. Defines the brand, or type of icon, to be used with the sign-in control. Valid values for this property are listed in the table below.	None.
redirect_uri	string	Optional. The location of the redirect page.	URL of the current page
scope	string or array of string values	Optional. Defines the scopes, in array format, that are requested when the user signs in. If no value is specified, the default scope is applied.	WL.signin
theme	string	Optional. Defines the appearance of the sign-in control. Valid values for this property are listed	blue (web app)

		in the table below.	dark (Metro style apps using JavaScript)
type	string	Optional. Defines the type of control. Valid values for this property are listed in the table below.	signin
sign_in_text	string	Optional. If the value of the type property is custom , this value specifies the sign-in text to be displayed in the UI control.	None.
sign_out_text	string	Optional. If the value of the type property is custom , this value specifies the sign-out text to be displayed in the UI control.	None.

The sign-in control's **brand** property can have one of the following values:

Value	Description
messenger	Messenger. Shows the Messenger "buddy" icon.
hotmail	Hotmail. Shows the "envelope" icon.
windows	Live Connect. Shows the Live Connect "flag" icon. This is the default value for both web apps and Metro style apps using JavaScript.
skydrive	SkyDrive. Shows the SkyDrive icon.
"none"	If the value "none" is specified, no icon is shown. If the brand property is left blank, the default value is shown.

The sign-in control's **theme** property can have one of the following values:

Value	Description
blue	Displays the blue theme for the sign-in control. Web apps only. Default.
white	Displays the white theme for the sign-in control. Web apps only.

dark	Displays the dark theme for the sign-in control. Metro style apps using JavaScript only. Default.
light	Displays the light theme for the sign-in control. Metro style apps using JavaScript only. Default.

The sign-in control's **type** property can have one of the following values:

Value	Sign-in text	Sign-out text	Additional info
signin	Sign in	Sign out	This is the default for the control.
login	Login	Logout	Not applicable.
connect	Connect	Signout	Not applicable.
custom	The value supplied in the <i>sign_in_text</i> property.	The value supplied in the <i>sign_out_text</i> property.	Your app defines the text. If the app does not provide <i>sign_in_text</i> and <i>sign_out_text</i> property values, the library will return an error, and the button will not render.

The **WL.ui** method creates the type of UI control that is specified in the **name** property by appending it to the DOM element on your page that is called **element**.

callback

Optional. A callback function that is executed when the UI element is rendered.

Requirements

Header	Not applicable
Library	WL.js
DLL	Not applicable

WL.upload function

[This documentation is preliminary and is subject to change.]

Makes a call to upload a file to SkyDrive.

Important WL.upload is only supported for use with Metro style apps using JavaScript. If you are writing a web app, use [WL.api](#) instead.

Parameters

properties

Required. A JSON object that contains the following properties, which are necessary to make the REST API call:

Name	Type	Description	Default Value
path	string	Required. The path to the file to upload.	None
file_name	string	Optional. The name of the file to upload.	None
file_input	StorageFile , or File	Required. The file input object to read the file from.	None
overwrite	boolean	Optional. Indicates whether the uploaded file should overwrite an existing copy. This only applies when uploading to a folder.	false

callback

Required. Specifies a callback function that is executed when the REST API call is complete. The callback function takes the API response object as a parameter. The response object exposes the data returned from Live Connect, or, if an error occurs, an error property that contains the error code.

Return value

This function does not return a value.

Requirements

Header	Not applicable
---------------	----------------

Library	WI.js
DLL	Not applicable

© 2011 Microsoft Corporation. All rights reserved.

JSONP scenarios

[This documentation is preliminary and is subject to change.]

You can use *JavaScript Objection Notation with padding* (JSONP) in client-side JavaScript webpage code to interact directly with the Live Connect Representational State Transfer (REST) API. Using JSONP is especially helpful when you want to retrieve simple Live Connect info for an individual user without writing all of the code to call the Live Connect JavaScript API. For example, with as little as a single line of code, you can make a request for a user's full name, info about one of the user's contacts, or one of the user's photos.

There are two main usage patterns for using JSONP: you can call JSONP as a webpage loads, and you can call it in response to a user action on a webpage.

Calling JSONP as a webpage loads

In this usage pattern, JSONP is called while the containing webpage initially loads from beginning to end, as shown in the following code example.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <textarea id="response" style="width:100%" cols="0" rows="20"></textarea>
    >

        <script src="JSONP.js"></script>
        <script src="json2.js"></script>

        <script>
            function onResponse(response) {
                var jsonAsString = JSON.stringify(response);

                document.getElementById("response").value = jsonAsString;
            }
        </script>

        <script src="">
        </script>

        <script>
            var scripts = document.getElementsByTagName("script");
            scripts[3].src = REST_API_URL;
        </script>
    </body>
</html>
```

The following occurs in the preceding code sample:

1. While the webpage loads, when the JavaScript interpreter reaches the last **script** element, the interpreter takes the URL string from the script in the JSONP.js file and inserts it into the fourth **script** element's empty **src** attribute.

2. When the webpage finishes loading, the JavaScript interpreter invokes the fourth **script** element, which in turn calls the Live Connect REST API. The [REST API](#) delivers the results of the call as a JavaScript Object Notation (JSON)-formatted string to the `onResponse` function. (This function is specified in the `src` attribute's callback URL parameter.)
3. The `onResponse` function uses the **JSON.stringify** function to convert the JSON-formatted data into a string, which is then displayed in the webpage's **textarea** control.

The preceding code sample relies on two additional JavaScript scripts: `JSONP.js` and `json2.js`. The `JSONP.js` JavaScript file contains the following code.

```
var ENDPOINT_REST_API = "https://apis.live.net/",
    ENDPOINT_REST_API_VERSION = "v5.0/",
    REST_PATH_ME = "me/",
    REST_PATH_CONTACTS = "contacts",
    ACCESS_TOKEN_PARAM = "?access_token=",
    JSONP_CALLBACK_PARAM = "&callback=",
    JSONP_CALLBACK_FUNCTION = "onResponse";

var ACCESS_TOKEN = "EwBQAq1DBAAU1bRWyAJjk5w968Ru3Cyt/6GvwXwAAXNW6nTKzzqgl86qQPkm
d1Yo4n+RZvmkdocatHGf3ZVvW2ZnrDFYg2cpMsbYEtA8pIQmdmCmx1WceM331ELb/uj+6Ckckq8A+xmz
d00iUiwhno45ep8ty1n0vVURSHh4x9jKv7nQjGH+QJIV6AezLV+1IYYoyI+rqUzNYB+d8GGxuBSU/mgM
/N8hYZMq8t95+OP15TUmJL+ptaACnFle/ne+fCjfacq3phbWdwv7YSM69LEAaWAktP84vqNFn9Apyd
1br6ifcWE3v1P2n314gHO6xXl+40fHICp7uioVZFysDV+eGP3qMPpTsZ7FDC1G3UmSMEI3Z5MdD5Pn4Q
5r4DZgAACHSFxmD0xGYUIAGofYQ7Ft4UFC9HgsHJlhImWVZz9ZvF4xMDlfkHCuMRV9/5qn6JzA9/3UR
0Of+724+vcsik/LNJSLeFTNzCKUTML+WVUiTo2sXYDjzs5E8T4xjIKMo01AzfuCJ0bjTI1jWWuipyBa
YNhz7wt1+eSID5iAi2xoloPLHyyfgTkk1dJRK2DwR239GZLwKbK+rWpER5u1cGHBRpIFMDJZIvPUz7AI
eDaVQXF81jUAfZT1MdwsjwGdbnjni4cso0dpQNzjE0WC3HyEFM4jPSev/5eEfyaz8iM2O+RFXKA326gn
PUYjvw8O3o1yqYyzGT5tbypNvDg1R9rIt7or4790Gf1VV7yhOEE8Cub+RBtVoROsWEIBtxptrSlGmg7N
i9RahAAAA==";

var REST_API_URL = ENDPOINT_REST_API + ENDPOINT_REST_API_VERSION +
    REST_PATH_ME + REST_PATH_CONTACTS +
    ACCESS_TOKEN_PARAM + ACCESS_TOKEN +
    JSONP_CALLBACK_PARAM + JSONP_CALLBACK_FUNCTION;
```

The preceding script contains variables that can help you debug and isolate runtime errors caused by any changes to hard-coded URLs upon which the Live Connect REST API relies.

The `json2.js` script converts JSON-formatted strings and objects. Although similar functionality is supplied by default with several web browsers, this script is referenced locally to achieve compatibility with a wider range of web browsers. (You can get this script by visiting a website such as [Introducing JSON](#) (<http://www.json.org>)).

Note To adapt the preceding code example for your own use, be sure to replace the `ACCESS_TOKEN` variable's value with an access token that represents your requested scope for the user. For more info, see [Server-side scenarios](#).

Call JSONP in response to a user action in a webpage

In this usage pattern, JSONP is called only when the user performs an action in the containing webpage, as shown in the following code example.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.or
```

```

g/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title></title>
  </head>
  <body>
    <button id="callApi" onclick="onCallApiClicked()">Call API</button>
    <textarea id="response" style="width:100%" cols="0" rows="20"></textarea>
  >

    <script src="JSONP.js"></script>
    <script src="json2.js"></script>

    <script>
      function onCallApiClicked() {
        var script = document.createElement("script");
        script.type = "text/javascript";
        script.language = "javascript";
        script.src = REST_API_URL;
        document.body.appendChild(script);
      }

      function onResponse(response) {
        var jsonAsString = JSON.stringify(response);

        document.getElementById("response").value = jsonAsString;
      }
    </script>
  </body>
</html>

```

This example differs from the example in the preceding section in the following ways:

- Because none of the script elements initially contains any calls to the Live Connect REST API, nothing happens after the webpage finishes loading.
- When the user clicks the **Call API** button, the `onCallApiClicked` function creates a new **script** element at the end of the webpage. The function takes the URL string from the JSONP.js script and inserts it into the element's **src** attribute.
- After the function calls the **document.body.appendChild** function, the JavaScript interpreter invokes the new element, which then calls the Live Connect REST API.
- The code continues to run as in the preceding section.

Each time the user clicks the **Call API** button, a new **script** element is created. As each new element is appended to the end of the webpage, the new element once again calls the Live Connect REST API.

Managed API

[This documentation is preliminary and is subject to change.]

The Live Connect Managed API simplifies the process of accessing user data from within Microsoft .NET apps, and provides controls to facilitate the sign-in experience for users. It includes the following namespaces.

Namespace	Description
Microsoft.Live	Contains classes that simplify access to Live Connect OAuth authentication, user session data, and the Live Connect REST API.
Microsoft.Live.Controls	Contains the Live Connect controls, which facilitate the sign-in experience for users.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft.Live namespace

[This documentation is preliminary and is subject to change.]

Contains classes that simplify access to Live Connect OAuth authentication, user session data, and the Live Connect REST API.

Members

The **Microsoft.Live** namespace has the following types of members:

- [Classes](#)
- [Enumerations](#)

Classes

The **Microsoft.Live** namespace has the following classes.

Class	Description
LiveAuthClient	Used to retrieve session data from Live Connect.
LiveAuthException	Used to indicate when an exception has occurred during the Auth process.
LiveConnectClient	Exposes the primary functionality of the Live Connect web service endpoints, and manages token expiry.
LiveConnectException	Used to indicate when an exception has occurred during the connection process.
LiveConnectSession	Used to access the current session.
LiveDownloadProgressChangedEventArgs	Raised every time a chunk of data is downloaded during a download operation.
LiveOperationCompletedEventArgs	Contains event arguments for the events raised when an operation is completed.
LiveUploadProgressChangedEventArgs	Raised every time a chunk of data is downloaded during an upload operation.
LoginCompletedEventArgs	Contains event arguments for the events raised when login

is completed.

Enumerations

The **Microsoft.Live** namespace has the following enumerations.

Enumeration	Description
LiveConnectSessionStatus	Indicates the status of the session.
ResponseType	The type of response (Code or Token flow).

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthClient class

[This documentation is preliminary and is subject to change.]

Used to retrieve session data from Live Connect.

Syntax

C#

```
public class LiveAuthClient
```

Attributes

None

Members

The **LiveAuthClient** class has the following types of members:

- [Events](#)
- [Methods](#)
- [Properties](#)

Events

The **LiveAuthClient** class has the following events.

Event	Description
InitializeCompleted	Raised when an InitializeAsync operation is completed.
LoginCompleted	Raised when a LoginAsync operation is completed.
PropertyChanged	Raises an event when the Session property is changed.

Methods

The **LiveAuthClient** class has the following methods.

Method	Description
InitializeAsync	Initializes the Session object.
LiveAuthClient	Initializes a new instance of the LiveAuthClient class.

LoginAsync	Initializes a new instance of the LiveAuthClient class.
Logout	Signs the user out from Live Connect.

Properties

The **LiveAuthClient** class has the following properties.

Property	Access type	Description
ResponseType	Read/write	Initializes a new instance of the LiveAuthClient class.
Session	Read-only	Initializes a new instance of the LiveAuthClient class.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

InitializeAsync::InitializeAsync methods

[This documentation is preliminary and is subject to change.]

Initializes the [Session](#) object. **InitializeAsync** requests an access token and creates a new session object. If a session cannot be created, the call to **InitializeAsync** fails.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overload list

Method	Description
InitializeAsync()	Initializes the Session object by loading it from cache.
InitializeAsync(object)	Initializes the Session object by loading it from cache, specifying a user state object.

See also

[InitializeAsync](#)
[LiveAuthClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthClient.InitializeAsync() method

[This documentation is preliminary and is subject to change.]

Initializes the [Session](#) object. **InitializeAsync** requests an access token and creates a new session object. If a session cannot be created, the call to **InitializeAsync** fails.

Syntax

C#

```
public InitializeAsync()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

InitializeAsync.InitializeAsync(object) (object) method

[This documentation is preliminary and is subject to change.]

Initializes the [Session](#) object. **InitializeAsync** requests an access token and creates a new session object. If a session cannot be created, the call to **InitializeAsync** fails.

Syntax

C#

```
public InitializeAsync(object)(  
    object userState  
)
```

Parameters

userState

Type: **object**

The user state object to be used to retrieve an access token for the user.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

InitializeAsync

LiveAuthClient.InitializeCompleted event

[This documentation is preliminary and is subject to change.]

Raised when an [InitializeAsync](#) operation is completed.

Syntax

C#

```
public event EventHandler<LoginCompletedEventArgs> InitializeCompleted
```

Event information

Delegate	EventHandler<LoginCompletedEventArgs>
-----------------	---------------------------------------

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

LiveAuthClient.LiveAuthClient methods

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveAuthClient** class.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overload list

Method	Description
LiveAuthClient()	Initializes a new instance of the LiveAuthClient class.
LiveAuthClient(string, string)	Initializes a new instance of the LiveAuthClient class, and sets the client ID and redirect URI.

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthClient.LiveAuthClient() method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveAuthClient** class. This overload is only supported for Metro style apps.

Syntax

C#

```
public LiveAuthClient LiveAuthClient()
```

Parameters

This method has no parameters.

Return value

Type: [LiveAuthClient](#)

An initialized instance of the class.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

[LiveAuthClient\(string, string\)](#)

LiveAuthClient.LiveAuthClient(string, string)(String, String) method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveAuthClient** class, and sets the client ID and redirect URI.

Syntax

C#

```
public LiveAuthClient LiveAuthClient(string, string)(  
    string clientID,  
    string redirectUri  
)
```

Parameters

clientID

Type: **String**

A unique identifier used by the client app to connect to Live Connect.

redirectUri

Type: **String**

The link that the OAuth server will navigate to after the user has given consent.

Return value

Type: [LiveAuthClient](#)

An initialized instance of the class.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[**LiveAuthClient**](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthClient.LoginAsync methods

[This documentation is preliminary and is subject to change.]

Initiates the Live Connect sign-in flow, or presents a Live Connect permissions request dialog. You can use **LoginAsync** as an alternative to the sign-in control, or to prompt the user for additional permissions in the context of an activity.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overload list

Method	Description
LoginAsync(System.Collections.Generic.IEnumerable<string>)	Initiates the Live Connect sign-in flow, or presents a Live Connect permissions request dialog. A list of scopes is passed.
LoginAsync(System.Collections.Generic.IEnumerable<string>, object)	Initiates the Live Connect sign-in flow, or presents a Live Connect permissions request dialog. A list of scopes and a user state object is passed.

See also

[LiveAuthClient](#)

LiveAuthClient.LoginAsync(IEnumerable<string>) method

[This documentation is preliminary and is subject to change.]

Initiates the Live Connect sign-in flow, or presents a Live Connect permissions request dialog. You can use **LoginAsync** as an alternative to the sign-in control, or to prompt the user for additional permissions in the context of an activity.

Syntax

C#

```
public LoginAsync(  
    IEnumerable<string> scopes  
)
```

Parameters

scopes

Type: **IEnumerable<string>**

A list of scopes needed by the application.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthClient.LoginAsync(IEnumerable<string>, object) method

[This documentation is preliminary and is subject to change.]

Initiates the Live Connect sign-in flow, or presents a Live Connect permissions request dialog. You can use **LoginAsync** as an alternative to the sign-in control, or to prompt the user for additional permissions in the context of an activity.

Syntax

C#

```
public LoginAsync(  
    IEnumerable<string> scopes,  
    object userState  
)
```

Parameters

scopes

Type: **IEnumerable<string>**

A list of scopes needed by the application.

userState

Type: **object**

The current user state.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthClient.LoginCompleted event

[This documentation is preliminary and is subject to change.]

Raised when a [LoginAsync](#) operation is completed.

Syntax

C#

```
public event EventHandler<LoginCompletedEventArgs> LoginCompleted
```

Event information

Delegate	EventHandler<LoginCompletedEventArgs>
-----------------	---

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

LiveAuthClient.Logout method

[This documentation is preliminary and is subject to change.]

Signs the user out from Live Connect.

Syntax

C#

```
public Logout()
```

Parameters

This method has no parameters.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported server	Windows Server Developer Preview
Namespace	System.Live
Assembly	System.Live.dll

See also

[LiveAuthClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthClient.PropertyChanged event

[This documentation is preliminary and is subject to change.]

Raises an event when the [Session](#) property is changed.

Syntax

C#

```
public event PropertyChangedEventHandler PropertyChanged
```

Event information

Delegate	PropertyChangedEventHandler
-----------------	---

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

LiveAuthClient.ResponseType property

[This documentation is preliminary and is subject to change.]

The response type to use for the request (Code or Token flow).

Syntax

C#

```
public ResponseType ResponseType { get; set; }
```

Property value

Type: **ResponseType**

Specifies which response type to use.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

LiveAuthClient.Session property

[This documentation is preliminary and is subject to change.]

The current session for the connected user.

Syntax

C#

```
public LiveConnectSession Session { get; }
```

Property value

Type: **LiveConnectSession**

Used to access the current session.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthClient](#)

LiveAuthException class

[This documentation is preliminary and is subject to change.]

Used to indicate when an exception has occurred during the Auth process.

Syntax

C#

```
public class LiveAuthException : System.Exception
```

Attributes

None

Members

The **LiveAuthException** class has the following types of members:

- [Methods](#)
- [Properties](#)

Methods

The **LiveAuthException** class has the following methods.

Method	Description
LiveAuthException	Initializes a new instance of the LiveAuthException class.
ToString	Returns a string that represents the current object.

Properties

The **LiveAuthException** class has the following properties.

Property	Access type	Description
ErrorCode	Read-only	The error code returned from the Auth Operation.

Requirements

Minimum supported	Windows Developer Preview
--------------------------	---------------------------

client	
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthException.ErrorCode property

[This documentation is preliminary and is subject to change.]

The error code returned from the Auth Operation.

Syntax

C#

```
public string ErrorCode { get; }
```

Property value

Type: **String**

The error code.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthException](#)

[LiveAuthClient](#)

LiveAuthException.LiveAuthException methods

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveAuthException** class.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overload list

Method	Description
LiveAuthException()	Initializes a new instance of the LiveAuthException class.
LiveAuthException(string, string)	Initializes a new instance of the LiveAuthException class. Takes an error code string and a message string.
LiveAuthException(string, string, System.Exception)	Initializes a new instance of the LiveAuthException class. Takes an error code string, a message string, and a System.Exception object.

See also

[LiveAuthException](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveAuthException.LiveAuthException() method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveAuthException** class.

Syntax

C#

```
public LiveAuthException()
```

Parameters

This method has no parameters.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthException](#)

LiveAuthException.LiveAuthException(string, string) method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveAuthException** class, passing an error code string and a message string.

Syntax

C#

```
public LiveAuthException(  
    string errorCode,  
    string message  
)
```

Parameters

errorCode

Type: **string**

The error code for the exception.

message

Type: **string**

The exception message.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthException](#)

LiveAuthException.LiveAuthException(string, string, System.Exception) method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveAuthException** class, passing an error code string, a message string, and a **System.Exception** object.

Syntax

C#

```
public LiveAuthException(  
    string errorCode,  
    string message,  
    System.Exception innerException  
)
```

Parameters

errorCode

Type: **string**

The error code for the exception.

message

Type: **string**

The exception message.

innerException

Type: **System.Exception**

The inner exception.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthException](#)

LiveAuthException.ToString method

[This documentation is preliminary and is subject to change.]

Returns a string that represents the current object.

Syntax

C#

```
public string ToString()
```

Parameters

This method has no parameters.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveAuthException](#)

LiveConnectClient class

[This documentation is preliminary and is subject to change.]

Exposes the primary functionality of the Live Connect web service endpoints, and manages token expiry.

Syntax

C#

```
public class LiveConnectClient
```

Attributes

None

Members

The **LiveConnectClient** class has the following types of members:

- [Events](#)
- [Methods](#)
- [Properties](#)

Events

The **LiveConnectClient** class has the following events.

Event	Description
DeleteCompleted	Raised when a DeleteAsync operation is completed.
DownloadCompleted	Raised when a DownloadAsync operation is completed.
DownloadProgressChanged	Raised every time a chunk of data is downloaded during a DownloadAsync operation.
GetCompleted	Raised when a GetAsync operation is completed.
PostCompleted	Raised when a PostAsync operation is completed.
PropertyChanged	Occurs when a property value changes.
PutCompleted	Raised when a PutAsync operation is completed.
UploadCompleted	Raised when an UploadAsync operation is

UploadProgressChanged	complete. Raised every time a chunk of data is downloaded during an UploadAsync operation.
------------------------------	---

Methods

The **LiveConnectClient** class has the following methods.

Method	Description
DeleteAsync	Performs a COPY operation using the REST API. Used to make calls with HTTP.Delete to the Live Connect REST API.
DownloadAsync	DownloadAsync is a helper function for making calls to the REST API.
GetAsync	GetAsync is a helper function for making calls to the REST API.
LiveConnectClient	Initializes a new instance of the LiveConnectClient class.
PostAsync	PostAsync is a helper function for making calls to the REST API.
PutAsync	PutAsync is a helper function for making calls to the REST API.
UploadAsync	UploadAsync is a helper function for making calls to the REST API.
UploadCancel	Cancels an upload operation.

Properties

The **LiveConnectClient** class has the following properties.

Property	Access type	Description
Session	Read/write	Used to get the status of the current user in a synchronous manner.

Requirements

Minimum supported client	Windows Developer Preview
---------------------------------	---------------------------

Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.CopyAsync methods

[This documentation is preliminary and is subject to change.]

Performs a COPY operation using the REST API. CopyAsync is a helper function for making calls to the REST API. CopyAsync supports all paths and methods in the REST API. CopyAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, CopyAsync will use that as well.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overload list

Method	Description
CopyAsync(string, string)	Performs a COPY operation using the REST API, passing a string containing the path of the item to copy, and a string with the destination path.
CopyAsync(string, string, object)	Performs a COPY operation using the REST API, passing a string containing the path of the item to copy, a string with the destination path, and a user state object.

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.CopyAsync(string, string) method

[This documentation is preliminary and is subject to change.]

Performs a COPY operation using the REST API, passing a string containing the path of the item to copy, and a string with the destination path. CopyAsync is a helper function for making calls to the REST API. CopyAsync supports all paths and methods in the REST API. CopyAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, CopyAsync will use that as well.

Syntax

C#

```
public CopyAsync(  
    string path,  
    string destination  
)
```

Parameters

path

Type: **string**

The path for the item to copy.

destination

Type: **string**

The destination path to copy the item to.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

LiveConnectClient

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.CopyAsync(string, string, object) method

[This documentation is preliminary and is subject to change.]

Performs a COPY operation using the REST API, passing a string containing the path of the item to copy, a string with the destination path, and a user state object. CopyAsync is a helper function for making calls to the REST API. CopyAsync supports all paths and methods in the REST API. CopyAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, CopyAsync will use that as well.

Syntax

C#

```
public CopyAsync(  
    string path,  
    string destination,  
    object userState  
)
```

Parameters

path

Type: **string**

The path for the item to copy.

destination

Type: **string**

The destination path to copy the item to.

userState

Type: **object**

The current user state object.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live

Assembly	Microsoft.Live.dll
-----------------	--------------------

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.DeleteAsync methods

[This documentation is preliminary and is subject to change.]

Used to make calls with `HTTP.Delete` to the Live Connect REST API. `DeleteAsync` supports all paths and methods in the REST API. `DeleteAsync` automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to avoid working directly with access tokens. If the access token cookie is set, `DeleteAsync` will use that as well.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overload list

Method	Description
DeleteAsync(string)	Used to make calls with <code>HTTP.Delete</code> to the Live Connect REST API. Takes a string containing the path to the item to delete.
DeleteAsync(string, object)	Used to make calls with <code>HTTP.Delete</code> to the Live Connect REST API. Takes a string containing the path to the item to delete, and an object with the current user state.

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.DeleteAsync(string) method

[This documentation is preliminary and is subject to change.]

Performs a COPY operation using the REST API. Used to make calls with HTTP.Delete to the Live Connect REST API. DeleteAsync supports all paths and methods in the REST API. DeleteAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to avoid working directly with access tokens. If the access token cookie is set, DeleteAsync will use that as well.

Syntax

C#

```
public DeleteAsync(  
    string Path  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.DeleteAsync(string, object) method

[This documentation is preliminary and is subject to change.]

Used to make calls with `HTTP.Delete` to the Live Connect REST API. `DeleteAsync` supports all paths and methods in the REST API. `DeleteAsync` automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to avoid working directly with access tokens. If the access token cookie is set, `DeleteAsync` will use that as well.

Syntax

C#

```
public DeleteAsync(  
    string Path,  
    object userState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

userState

Type: **object**

The current user state.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.DeleteCompleted event

[This documentation is preliminary and is subject to change.]

Raised when a [DeleteAsync](#) operation is completed.

Syntax

C#

```
public event EventHandler<LiveOperationCompletedEventArgs> DeleteCompleted
```

Event information

Delegate

[EventHandler<LiveOperationCompletedEventArgs>](#)

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.DownloadAsync methods

[This documentation is preliminary and is subject to change.]

DownloadAsync is a helper function for making calls to the REST API. DownloadAsync supports all paths and methods in the REST API. DownloadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, DownloadAsync will use that as well.

Overload list

Method	Description
DownloadAsync(string, System.IO.Stream)	A helper function for making calls to the REST API. Takes a string containing the path to the item to download, and a System.IO.Stream object to receive the stream.
DownloadAsync(string, System.IO.Stream, object)	A helper function for making calls to the REST API. Takes a string containing the path to the item to download, a System.IO.Stream object to receive the stream, and the current user state.

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.DownloadAsync(string, Stream) method

[This documentation is preliminary and is subject to change.]

DownloadAsync is a helper function for making calls to the REST API. DownloadAsync supports all paths and methods in the REST API. DownloadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, DownloadAsync will use that as well.

Syntax

C#

```
public DownloadAsync(  
    string Path,  
    Stream Stream  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Stream

Type: **Stream**

Data stream for the content to download.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.DownloadAsync(string, Stream, Object) method

[This documentation is preliminary and is subject to change.]

DownloadAsync is a helper function for making calls to the REST API. DownloadAsync supports all paths and methods in the REST API. DownloadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, DownloadAsync will use that as well.

Syntax

C#

```
public DownloadAsync(  
    string Path,  
    Stream Stream,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Stream

Type: **Stream**

Data stream for the content to download.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.DownloadCompleted event

[This documentation is preliminary and is subject to change.]

Raised when a [DownloadAsync](#) operation is completed.

Syntax

C#

```
public event EventHandler<LiveOperationCompletedEventArgs> DownloadCompleted
```

Event information

Delegate	EventHandler<LiveOperationCompletedEventArgs>
-----------------	---

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.DownloadProgressChanged event

[This documentation is preliminary and is subject to change.]

Raised every time a chunk of data is downloaded during a [DownloadAsync](#) operation.

Syntax

C#

```
public event ProgressChangedEventHandler Delegate DownloadProgressChanged
```

Event information

Delegate	ProgressChangedEventHandler Delegate
-----------------	--

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.GetAsync methods

[This documentation is preliminary and is subject to change.]

GetAsync is a helper function for making calls to the REST API. GetAsync supports all paths and methods in the REST API. GetAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, GetAsync will use that as well.

Overload list

Method	Description
GetAsync(string)	A helper function for making calls to the REST API. Takes a string containing the path to the object to get.
GetAsync(string, object)	A helper function for making calls to the REST API. Takes a string containing the path to the object to get, and the current user state.

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.GetAsync(string) method

[This documentation is preliminary and is subject to change.]

GetAsync is a helper function for making calls to the REST API. GetAsync supports all paths and methods in the REST API. GetAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, GetAsync will use that as well.

Syntax

C#

```
public GetAsync(  
    string Path  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.GetAsync(string, Object) method

[This documentation is preliminary and is subject to change.]

GetAsync is a helper function for making calls to the REST API. GetAsync supports all paths and methods in the REST API. GetAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, GetAsync will use that as well.

Syntax

C#

```
public GetAsync(  
    string Path,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.GetCompleted event

[This documentation is preliminary and is subject to change.]

Raised when a [GetAsync](#) operation is completed.

Syntax

C#

```
public event EventHandler<LiveOperationCompletedEventArgs> GetCompleted
```

Event information

Delegate

[EventHandler<LiveOperationCompletedEventArgs>](#)

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.LiveConnectClient method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveConnectClient** class.

Syntax

C#

```
public LiveConnectClient(  
    LiveConnectSession Session  
)
```

Parameters

Session

Type: [LiveConnectSession](#)

The **LiveConnectSession** that contains all the access information for the LiveService.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.MoveAsync methods

[This documentation is preliminary and is subject to change.]

MoveAsync is a helper function for making calls to the REST API. MoveAsync supports all paths and methods in the REST API. MoveAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, MoveAsync will use that as well.

Overload list

Method	Description
MoveAsync	
MoveAsync	

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.MoveAsync(string, string) method

[This documentation is preliminary and is subject to change.]

MoveAsync is a helper function for making calls to the REST API. MoveAsync supports all paths and methods in the REST API. MoveAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, MoveAsync will use that as well.

Syntax

C#

```
public MoveAsync(  
    string path,  
    string destination  
)
```

Parameters

path

Type: **string**

Information about the API request being made including path and HTTP method.

destination

Type: **string**

The path to which to move the resource.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.MoveAsync(string, string, Object) method

[This documentation is preliminary and is subject to change.]

MoveAsync is a helper function for making calls to the REST API. MoveAsync supports all paths and methods in the REST API. MoveAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, MoveAsync will use that as well.

Syntax

C#

```
public MoveAsync(  
    string path,  
    string destination,  
    object userState  
)
```

Parameters

path

Type: **string**

Information about the API request being made including path and HTTP method.

destination

Type: **string**

The path to which to move the resource.

userState

Type: **Object**

The current user state.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live

Assembly

Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PostAsync methods

[This documentation is preliminary and is subject to change.]

TBD

Overload list

Method	Description
PostAsync(string, string)	A helper function for making calls to the WL REST API. Takes a string containing the path to the object to get, and a string containing the body of the API call.
PostAsync(string, IDictionary)	
PostAsync(string, string, object)	A helper function for making calls to the WL REST API. Takes a string containing the path to the object to get, and a string containing the body of the API call.
PostAsync(string, IDictionary, object)	

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PostAsync(string, string) method

[This documentation is preliminary and is subject to change.]

PostAsync is a helper function for making calls to the REST API. PostAsync supports all paths and methods in the REST API. PostAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PostAsync will use that as well.

Syntax

C#

```
public PostAsync(  
    string Path,  
    string Body  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Body

Type: **string**

The body that should be included in the API call.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

LiveConnectClient

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PostAsync(String, IDictionary<string , object>) method

[This documentation is preliminary and is subject to change.]

PostAsync is a helper function for making calls to the REST API. PostAsync supports all paths and methods in the REST API. PostAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PostAsync will use that as well.

Syntax

C#

```
public PostAsync(  
    string Path,  
    IDictionary<string , object> Body  
)
```

Parameters

Path

Type: **String**

Information about the API request being made including path and HTTP method.

Body

Type: **IDictionary<string , object>**

The body that should be included in the API call.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PostAsync(string, string, Object) method

[This documentation is preliminary and is subject to change.]

PostAsync is a helper function for making calls to the REST API. PostAsync supports all paths and methods in the REST API. PostAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PostAsync will use that as well.

Syntax

C#

```
public PostAsync(  
    string Path,  
    string Body,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Body

Type: **string**

The body that should be included in the API call.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live

Assembly	Microsoft.Live.dll
-----------------	--------------------

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PostAsync(string, IDictionary<string , object>, Object) method

[This documentation is preliminary and is subject to change.]

PostAsync is a helper function for making calls to the REST API. PostAsync supports all paths and methods in the REST API. PostAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PostAsync will use that as well.

Syntax

C#

```
public PostAsync(  
    string Path,  
    IDictionary<string , object> Body,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Body

Type: **IDictionary<string , object>**

The body that should be included in the API call.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7

Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PutAsync methods

[This documentation is preliminary and is subject to change.]

Overload list

Method	Description
PutAsync(string, string)	
PutAsync(string, IDictionary)	
PutAsync(string, string, object)	
PutAsync(string, IDictionary, object)	

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PutAsync(string, string)(string, string) method

[This documentation is preliminary and is subject to change.]

PutAsync is a helper function for making calls to the REST API. PutAsync supports all paths and methods in the REST API. PutAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PutAsync will use that as well.

Syntax

C#

```
public PutAsync(string, string)(  
    string Path,  
    string Body  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Body

Type: **string**

The body that should be included in the API call.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

LiveConnectClient

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PutAsync(string, IDictionary<string , object>) method

[This documentation is preliminary and is subject to change.]

PutAsync is a helper function for making calls to the REST API. PutAsync supports all paths and methods in the REST API. PutAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PutAsync will use that as well.

Syntax

C#

```
public PutAsync(  
    string Path,  
    IDictionary<string , object> Body  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Body

Type: **IDictionary<string , object>**

The body that should be included in the API call.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PutAsync(string, string, Object) method

[This documentation is preliminary and is subject to change.]

PutAsync is a helper function for making calls to the REST API. PutAsync supports all paths and methods in the REST API. PutAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PutAsync will use that as well.

Syntax

C#

```
public PutAsync(  
    string Path,  
    string Body,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Body

Type: **string**

The body that should be included in the API call.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live

Assembly	Microsoft.Live.dll
-----------------	--------------------

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PutAsync(string, IDictionary<string , object>, Object) method

[This documentation is preliminary and is subject to change.]

PutAsync is a helper function for making calls to the REST API. PutAsync supports all paths and methods in the REST API. PutAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, PutAsync will use that as well.

Syntax

C#

```
public PutAsync(  
    string Path,  
    IDictionary<string , object> Body,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

Body

Type: **IDictionary<string , object>**

The body that should be included in the API call.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview

Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.PropertyChanged event

[This documentation is preliminary and is subject to change.]

Occurs when a property value changes.

Syntax

C#

```
public event PropertyChangedEventHandler PropertyChanged
```

Event information

Delegate	PropertyChangedEventHandler
-----------------	-----------------------------

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.PostCompleted event

[This documentation is preliminary and is subject to change.]

Raised when a [PostAsync](#) operation is completed.

Syntax

C#

```
public event EventHandler<LiveOperationCompletedEventArgs> PostCompleted
```

Event information

Delegate

[EventHandler<LiveOperationCompletedEventArgs>](#)

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.PutCompleted event

[This documentation is preliminary and is subject to change.]

Raised when a [PutAsync](#) operation is completed.

Syntax

C#

```
public event EventHandler<LiveOperationCompletedEventArgs> PutCompleted
```

Event information

Delegate

[EventHandler<LiveOperationCompletedEventArgs>](#)

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.Session property

[This documentation is preliminary and is subject to change.]

Used to get the status of the current user in a synchronous manner. This is useful for getting the current status of a user in a synchronous manner but should be avoided in places where synchronous calls will impact the performance of the application, for example at load time.

Syntax

C#

```
public LiveConnectSession Session { }
```

Property value

Type: [LiveConnectSession](#)

Session

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.UploadAsync methods

[This documentation is preliminary and is subject to change.]

UploadAsync is a helper function for making calls to the REST API. UploadAsync supports all paths and methods in the REST API. UploadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, UploadAsync will use that as well.

Overload list

Method	Description
UploadAsync(string, IStorageFile)	
UploadAsync(string, IStorageFile, object)	
UploadAsync(string, string, stream)	
UploadAsync(string, string, stream, object)	
UploadAsync(string, string, long, stream, object)	

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.UploadAsync(string, IStorageFile) method

[This documentation is preliminary and is subject to change.]

UploadAsync is a helper function for making calls to the REST API. UploadAsync supports all paths and methods in the REST API. UploadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, UploadAsync will use that as well.

Syntax

C#

```
public UploadAsync(  
    string Path,  
    IStorageFile inputFile  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

inputFile

Type: **IStorageFile**

The name of the file to be uploaded.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.UploadAsync(string, IStorageFile, Object) method

[This documentation is preliminary and is subject to change.]

Syntax

C#

```
public UploadAsync(  
    string Path,  
    IStorageFile inputFile,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

inputFile

Type: **IStorageFile**

The name of the file to be uploaded.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.UploadAsync(string, string, Stream) method

[This documentation is preliminary and is subject to change.]

UploadAsync is a helper function for making calls to the REST API. UploadAsync supports all paths and methods in the REST API. UploadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, UploadAsync will use that as well.

Syntax

C#

```
public UploadAsync(  
    string Path,  
    string fileName,  
    Stream inputStream  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

fileName

Type: **string**

The name of the file to be uploaded.

inputStream

Type: **Stream**

The stream to add to the body of the API request.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

LiveConnectClient

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.UploadAsync(string, string, Stream, Object) method

[This documentation is preliminary and is subject to change.]

UploadAsync is a helper function for making calls to the REST API. UploadAsync supports all paths and methods in the REST API. UploadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, UploadAsync will use that as well.

Syntax

C#

```
public UploadAsync(  
    string Path,  
    string fileName,  
    Stream inputStream,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

fileName

Type: **string**

The name of the file to be uploaded.

inputStream

Type: **Stream**

The stream to add to the body of the API request.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview

Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.UploadAsync(string, string, Long, Stream, Object) method

[This documentation is preliminary and is subject to change.]

UploadAsync is a helper function for making calls to the REST API. UploadAsync supports all paths and methods in the REST API. UploadAsync automatically uses the access token returned from user sign-in via the constructor, making it possible for a site to omit working directly with access tokens. If the access token cookie is set, UploadAsync will use that as well.

Syntax

C#

```
public UploadAsync(  
    string Path,  
    string fileName,  
    Long fileSize,  
    Stream inputStream,  
    object UserState  
)
```

Parameters

Path

Type: **string**

Information about the API request being made including path and HTTP method.

fileName

Type: **string**

The name of the file to be uploaded.

fileSize

Type: **Long**

The size of the file to be uploaded in bytes.

inputStream

Type: **Stream**

The stream to add to the body of the API request.

UserState

Type: **Object**

Used to uniquely identify the call. The UserState object is passed back to the user when the call is completed.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7

Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectClient.UploadCancel method

[This documentation is preliminary and is subject to change.]

Cancels an upload operation.

Syntax

C#

```
public UploadCancel()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.UploadCompleted event

[This documentation is preliminary and is subject to change.]

Raised when an [UploadAsync](#) operation is complete.

Syntax

C#

```
public event EventHandler<LiveOperationCompletedEventArgs> UploadCompleted
```

Event information

Delegate

[EventHandler<LiveOperationCompletedEventArgs>](#)

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

LiveConnectClient.UploadProgressChanged event

[This documentation is preliminary and is subject to change.]

Raised every time a chunk of data is downloaded during an [UploadAsync](#) operation.

Syntax

C#

```
public event ProgressChangedEventHandler Delegate UploadProgressChanged
```

Event information

Delegate	ProgressChangedEventHandler Delegate
-----------------	--

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectClient](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectException class

[This documentation is preliminary and is subject to change.]

Used to indicate when an exception has occurred during the connection process.

Syntax

C#

```
public class LiveConnectException : System.Exception
```

Attributes

None

Members

The **LiveConnectException** class has the following types of members:

- [Methods](#)
- [Properties](#)

Methods

The **LiveConnectException** class has the following methods.

Method	Description
LiveConnectException	Initializes a new instance of the LiveConnectException class.
ToString	Returns a string that represents the current object.

Properties

The **LiveConnectException** class has the following properties.

Property	Access type	Description
ErrorCode	Read-only	The error code returned from the operation.

Requirements

Minimum supported	Windows Developer Preview
--------------------------	---------------------------

client	
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectException.ErrorCode property

[This documentation is preliminary and is subject to change.]

The error code returned from the operation.

Syntax

C#

```
public string ErrorCode { get; }
```

Property value

Type: **String**

ErrorCode

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectException](#)

LiveConnectException::LiveConnectException methods

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveConnectException** class.

Overload list

Method	Description
LiveConnectException	This overload takes no parameters.
LiveConnectException(string, string)	This overload takes a string containing the error code and a string containing the error message.
LiveConnectException(string, string, exception)	This overload takes a string containing the error code, a string containing the error message, and an exception object.

See also

[LiveConnectException](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectException.LiveConnectException() method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveConnectException** class.

Syntax

C#

```
public LiveConnectException()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectException](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectException.LiveConnectException(String, String) method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveConnectException** class.

Syntax

C#

```
public LiveConnectException(  
    string errorCode,  
    string message  
)
```

Parameters

errorCode
Type: **String**

TBD

message
Type: **String**

TBD

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported server	Windows Server Developer Preview
Namespace	System.Live
Assembly	System.Live.dll

See also

[LiveConnectException](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectException.LiveConnectException(String, String, Exception) method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveConnectException** class.

Syntax

C#

```
public LiveConnectException(  
    string errorCode,  
    string message,  
    Exception innerException  
)
```

Parameters

errorCode
Type: **String**

TBD

message
Type: **String**

TBD

innerException
Type: **Exception**

TBD

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported server	Windows Server Developer Preview
Namespace	System.Live
Assembly	System.Live.dll

See also

[LiveConnectException](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectException.ToString method

[This documentation is preliminary and is subject to change.]

Returns a string that represents the current object.

Syntax

C#

```
public string ToString()
```

Parameters

This method has no parameters.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectException](#)

LiveConnectSession class

[This documentation is preliminary and is subject to change.]

Used to access the current session.

Syntax

C#

```
public class LiveConnectSession
```

Attributes

None

Members

The **LiveConnectSession** class has the following types of members:

- Properties

Properties

The **LiveConnectSession** class has the following properties.

Property	Access type	Description
AccessToken	Read-only	The access token for a signed-in and connected user.
AuthenticationToken	Read-only	The authentication token for a signed-in and connected user.
Expires	Read-only	Exact time when the session expires. Unit is in seconds since 1970/01/01.
IsValid	Read-only	Indicates whether the session is valid.
RefreshToken	Read-only	Provides a refresh token that is specific to both the user and the app, which can be used to refresh the access token.
Scopes	Read-only	If the user has consented to one or more permissions, the list of consented scopes is included here. This includes consent given on the initial login.
Status	Read-only	Indicates the status of the user.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectSession.AccessToken property

[This documentation is preliminary and is subject to change.]

The access token for a signed-in and connected user.

Syntax

C#

```
public string AccessToken { get; }
```

Property value

Type: **String**

AccessToken

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

LiveConnectSession.AuthenticationToken property

[This documentation is preliminary and is subject to change.]

The authentication token for a signed-in and connected user.

Syntax

C#

```
public string AuthenticationToken { get; set; }
```

Property value

Type: **String**

AuthenticationToken

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveConnectSession.Expires property

[This documentation is preliminary and is subject to change.]

Exact time when the session expires. Unit is in seconds since 1970/01/01.

Syntax

C#

```
public DateTimeOffset Expires { get; set; }
```

Property value

Type: **DateTimeOffset**

Expires

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

LiveConnectSession.IsValid property

[This documentation is preliminary and is subject to change.]

Indicates whether the session is valid.

Syntax

C#

```
public bool IsValid { get; set; }
```

Property value

Type: **Boolean**

IsValid

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

LiveConnectSession.LiveConnectSession method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveConnectException** class.

Syntax

C#

```
public LiveConnectSession(  
    Microsoft.Live.LiveAuthClient authClient  
)
```

Parameters

authClient

Type: [Microsoft.Live.LiveAuthClient](#)

A **LiveAuthClient** object to receive authentication data from Live Connect.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

LiveConnectSession.RefreshToken property

[This documentation is preliminary and is subject to change.]

Provides a refresh token that is specific to both the user and the app, which can be used to refresh the access token.

Syntax

C#

```
public string RefreshToken { get; set; }
```

Property value

Type: **String**

RefreshToken

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

LiveConnectSession.Scopes property

[This documentation is preliminary and is subject to change.]

If the user has consented to one or more permissions, the list of consented scopes is included here. This includes consent given on the initial login. If this is an incremental permission request and the value of scopes is null, then the user has rejected the permission request. Note that if a user rejects a permission request, this is also explicitly returned in the error value. This object is never null.

Syntax

C#

```
public IEnumerable<String> Scopes { get; set; }
```

Property value

Type: **IEnumerable<String>**

Scopes

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

LiveConnectSession.Status property

[This documentation is preliminary and is subject to change.]

Indicates the status of the user.

Syntax

C#

```
public LiveConnectSessionStatus Status { get; set; }
```

Property value

Type: **LiveConnectSessionStatus**

Status

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveConnectSession](#)

LiveConnectSessionStatus enumeration

[This documentation is preliminary and is subject to change.]

Indicates the status of the session.

Syntax

C#

```
public enum LiveConnectSessionStatus
```

Attributes

None

Members

The **LiveConnectSessionStatus** enumeration has the following members.

Member	Value	Description
Connected	Connected	Indicates that the session is connected.
NotConnected	NotConnected	Indicates that the session is not connected.
Unknown	Unknown	Indicates that the session status is not known.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveDownloadProgressChangedEventArgs class

[This documentation is preliminary and is subject to change.]

Raised every time a chunk of data is downloaded during a download operation.

Syntax

C#

```
public class LiveDownloadProgressChangedEventArgs : System.ComponentModel.ProgressChangedEventArgs
```

Attributes

None

Members

The **LiveDownloadProgressChangedEventArgs** class has the following types of members:

- Properties

Properties

The **LiveDownloadProgressChangedEventArgs** class has the following properties.

Property	Access type	Description
BytesReceived	Read-only	Gets the bytes received.
TotalBytesToReceive	Read-only	Gets the total number of bytes in a download operation.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveDownloadProgressChangedEventArgs.BytesReceived property

[This documentation is preliminary and is subject to change.]

Gets the bytes received.

Syntax

C#

```
public long BytesReceived { get; }
```

Property value

Type: **Int64**

BytesReceived

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveDownloadProgressChangedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveDownloadProgressChangedEventArgs.LiveDownloadProgressChanged method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the [LiveDownloadProgressChangedEventArgs](#) class.

Syntax

C#

```
public LiveDownloadProgressChangedEventArgs()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveDownloadProgressChangedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveDownloadProgressChangedEventArgs.TotalBytesToRec property

[This documentation is preliminary and is subject to change.]

Gets the total number of bytes in a download operation.

Syntax

C#

```
public long TotalBytesToReceive { get; }
```

Property value

Type: **Int64**

TotalBytesToReceive

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveDownloadProgressChangedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveOperationCompletedEventArgs class

[This documentation is preliminary and is subject to change.]

Contains event arguments for the events raised when an operation is completed.

Syntax

C#

```
public class LiveOperationCompletedEventArgs
```

Attributes

None

Members

The **LiveOperationCompletedEventArgs** class has the following types of members:

- [Methods](#)
- [Properties](#)

Methods

The **LiveOperationCompletedEventArgs** class has the following methods.

Method	Description
LiveOperationCompletedEventArgs	Initializes a new instance of the LiveOperationCompletedEventArgs class.

Properties

The **LiveOperationCompletedEventArgs** class has the following properties.

Property	Access type	Description
RawResult	Read-only	The raw result of the operation in the requested format.
Result	Read-only	The result of the requesting operation.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveOperationCompletedEventArgs::LiveOperationCompletedEventArgs methods

[This documentation is preliminary and is subject to change.]

TBD

Overload list

Method	Description
LiveOperationCompletedEventArgs(Exception, bool, object)	
LiveOperationCompletedEventArgs(IDictionary<string, object>)	

See also

[LiveOperationCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveOperationCompletedEventArgs.LiveOperationComplete method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the **LiveOperationCompletedEventArgs** class.

Syntax

C#

```
public LiveOperationCompletedEventArgs LiveOperationCompletedEventArgs()
```

Parameters

This method has no parameters.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveOperationCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveOperationCompletedEventArgs.LiveOperationComplete bool, object)() method

[This documentation is preliminary and is subject to change.]

TBD

Syntax

C#

```
public LiveOperationCompletedEventArgs(Exception, bool, object)(  
)
```

VB

```
Public Function LiveOperationCompletedEventArgs(Exception, bool, object)(  
    As  
)
```

Parameters

Type:

TBD

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported server	Windows Server Developer Preview
Namespace	System.Live
Assembly	System.Live.dll

See also

[LiveOperationCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveOperationCompletedEventArgs.LiveOperationComplete(string, object)() method

[This documentation is preliminary and is subject to change.]

TBD

Syntax

C#

```
public LiveOperationCompletedEventArgs(IDictionary, string, object)(  
)
```

VB

```
Public Function LiveOperationCompletedEventArgs(IDictionary, string, object)(  
    As  
)
```

Parameters

Type:

TBD

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported server	Windows Server Developer Preview
Namespace	System.Live
Assembly	System.Live.dll

See also

[LiveOperationCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveOperationCompletedEventArgs.RawResult property

[This documentation is preliminary and is subject to change.]

The raw result of the operation in the requested format.

Syntax

C#

```
public object RawResult { get; set; }
```

Property value

Type: **Object**

RawResult

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveOperationCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveOperationCompletedEventArgs.Result property

[This documentation is preliminary and is subject to change.]

The result of the requesting operation.

Syntax

C#

```
public IDictionary<string, object> Result { get; }
```

Property value

Type: **IDictionary<string, object>**

Result

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveOperationCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveUploadProgressChangedEventArgs class

[This documentation is preliminary and is subject to change.]

Raised every time a chunk of data is downloaded during an upload operation.

Syntax

C#

```
public class LiveUploadProgressChangedEventArgs
```

Attributes

None

Members

The **LiveUploadProgressChangedEventArgs** class has the following types of members:

- [Methods](#)
- [Properties](#)

Methods

The **LiveUploadProgressChangedEventArgs** class has the following methods.

Method	Description
LiveUploadProgressChangedEventArgs	Initializes a new instance of the LiveUploadProgressChangedEventArgs class.

Properties

The **LiveUploadProgressChangedEventArgs** class has the following properties.

Property	Access type	Description
BytesSent	Read-only	Gets the bytes sent.
TotalBytesToSend	Read-only	Gets the total number of bytes in an upload operation.

Requirements

Minimum supported

Windows Developer Preview

client	
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LiveUploadProgressChangedEventArgs.BytesSent property

[This documentation is preliminary and is subject to change.]

Gets the bytes sent.

Syntax

C#

```
public long BytesSent { get; }
```

Property value

Type: **Int64**

BytesSent

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveUploadProgressChangedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveUploadProgressChangedEventArgs.LiveUploadProgress method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the [LiveUploadProgressChangedEventArgs](#) class.

Syntax

C#

```
public LiveUploadProgressChangedEventArgs()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveUploadProgressChangedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LiveUploadProgressChangedEventArgs.TotalBytesToSend property

[This documentation is preliminary and is subject to change.]

Gets the total number of bytes in an upload operation.

Syntax

C#

```
public long TotalBytesToSend { get; set; }
```

Property value

Type: **Int64**

TotalBytesToSend

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LiveUploadProgressChangedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LoginCompletedEventArgs class

[This documentation is preliminary and is subject to change.]

Contains event arguments for the events raised when login is completed.

Syntax

C#

```
public class LoginCompletedEventArgs
```

Attributes

None

Members

The **LoginCompletedEventArgs** class has the following types of members:

- [Methods](#)
- [Properties](#)

Methods

The **LoginCompletedEventArgs** class has the following methods.

Method	Description
LoginCompletedEventArgs	Used to initialize a new instance of the LoginCompletedEventArgs class.

Properties

The **LoginCompletedEventArgs** class has the following properties.

Property	Access type	Description
Session	Read-only	Returns the LiveConnectSession object which contains Login and scope information along with the token information for the logged in user.

Requirements

Minimum supported client	Windows Developer Preview
---------------------------------	---------------------------

Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

© 2011 Microsoft Corporation. All rights reserved.

LoginCompletedEventArgs::LoginCompletedEventArgs methods

[This documentation is preliminary and is subject to change.]

TBD

Overload list

Method	Description
LoginCompletedEventArgs(Exception, bool, Object)	
LoginCompletedEventArgs(LiveConnectSession, Object)	

See also

[LoginCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LoginCompletedEventArgs.LoginCompletedEventArgs method

[This documentation is preliminary and is subject to change.]

Used to initialize a new instance of the **LoginCompletedEventArgs** class.

Syntax

C#

```
public LoginCompletedEventArgs LoginCompletedEventArgs()
```

Parameters

This method has no parameters.

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LoginCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LoginCompletedEventArgs.LoginCompletedEventArgs(Exception, bool, Object) method

[This documentation is preliminary and is subject to change.]

TBD

Syntax

C#

```
public LoginCompletedEventArgs(  
    Exception error,  
    bool cancelled,  
    object userState  
)
```

Parameters

error Type: **Exception**

TBD

cancelled Type: **bool**

TBD

userState Type: **Object**

TBD

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported server	Windows Server Developer Preview
Namespace	System.Live
Assembly	System.Live.dll

See also

[LoginCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LoginCompletedEventArgs.LoginCompletedEventArgs(LiveObject) method

[This documentation is preliminary and is subject to change.]

TBD

Syntax

C#

```
public LoginCompletedEventArgs(  
    LiveConnectSession session,  
    object userState  
)
```

Parameters

session
Type: **LiveConnectSession**

TBD

userState
Type: **Object**

TBD

Return value

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported server	Windows Server Developer Preview
Namespace	System.Live
Assembly	System.Live.dll

See also

[LoginCompletedEventArgs](#)

© 2011 Microsoft Corporation. All rights reserved.

LoginCompletedEventArgs.Session property

[This documentation is preliminary and is subject to change.]

Returns the [LiveConnectSession](#) object which contains Login and scope information along with the token information for the logged in user.

Syntax

C#

```
public LiveConnectSession Session { get; }
```

Property value

Type: [LiveConnectSession](#)

Session

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[LoginCompletedEventArgs](#)

ResponseType enumeration

[This documentation is preliminary and is subject to change.]

The type of response (Code or Token flow).

Syntax

C#

```
public enum ResponseType
```

Attributes

None

Members

The **ResponseType** enumeration has the following members.

Member	Value	Description
Code	Code	Specifies that an authorization_code should be returned.
Token	Token	Specifies that an access_token should be returned.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

Microsoft.Live.Controls namespace

[This documentation is preliminary and is subject to change.]

Represents controls to access Live Connect features.

Members

The **Microsoft.Live.Controls** namespace has the following types of members:

- [Classes](#)
- [Enumerations](#)

Classes

The **Microsoft.Live.Controls** namespace has the following classes.

Class	Description
SignInButton	A button that can be placed in an application to initiate the process of connecting the application and the user's Microsoft account.

Enumerations

The **Microsoft.Live.Controls** namespace has the following enumerations.

Enumeration	Description
BrandingType	Contains the values for the branding types that can be displayed.
ButtonTextType	The text values that can be displayed in the control.

BrandingType enumeration

[This documentation is preliminary and is subject to change.]

Contains the values for the branding types that can be displayed. If "None" is specified, then no icon is displayed.

Syntax

C#

```
public enum BrandingType
```

Attributes

()

Members

The **BrandingType** enumeration has the following members.

Member	Value	Description
Hotmail	Hotmail	The Hotmail "envelope" icon.
Messenger	Messenger	The Messenger "buddy" icon.
Skydrive	Skydrive	The Skydrive "cloud" icon.
Windows	Windows	The Live Connect "flag" icon. This is the default value.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

© 2011 Microsoft Corporation. All rights reserved.

ButtonType enumeration

[This documentation is preliminary and is subject to change.]

The text values that can be displayed in the control.

Syntax

C#

```
public enum ButtonType
```

Attributes

()

Members

The **ButtonType** enumeration has the following members.

Member	Value	Description
Connect	Sign in text: Connect ; Sign out text: Sign out	
Custom	Sign in text: (application-defined); Sign out text: (application-defined)	Text for the control is taken from the <i>SignInText</i> and <i>SignOutText</i> properties defined on the site. If no values are defined, the default value is used.
Login	Sign in text: Login ; Sign out text: Logout	
Signin	Sign in text: Sign in ; Sign out text: Sign out	The default value.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7

Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

© 2011 Microsoft Corporation. All rights reserved.

SignInButton class

[This documentation is preliminary and is subject to change.]

A button that can be placed in an application to initiate the process of connecting the application and the user's Microsoft account. The button can be in one of two states:

- Sign in: Either there is no user signed in, or the signed in user has not given permission to the application. The button shows a localized string that indicates to the user that they can sign in.
- Sign out: There is a connected user. The button shows a localized string that indicates to the user that they can sign out.

Syntax

C#

```
public class SignInButton
```

Attributes

()

Members

The **SignInButton** class has the following types of members:

- [Events](#)
- [Methods](#)
- [Properties](#)

Events

The **SignInButton** class has the following events.

Event	Description
SessionChanged	

Methods

The **SignInButton** class has the following methods.

Method	Description
Initialize	Initializes a new instance of the SignInButton class with default property values.

InitializeComponent	Initializes XAML components in constructors for WPF forms or controls. This method is called from the code-behind class's constructor to merge the UI that is defined in XAML markup with the code-behind class, set properties, and register event handlers.
SignInButton	Creates a new instance of the SignInButton class.

Properties

The **SignInButton** class has the following properties.

Property	Access type	Description
Branding	Read/write	Specifies the type of branding to display on the sign in button. The brand is the icon and visual theme displayed on the button, relating one of the following products: Hotmail, Windows Live Messenger, SkyDrive, Live Connect, or none. The brands are defined by the BrandingType enumeration.
BrandingProperty	Read/write	A dependency property identifier that indicates the type of branding to display on the sign in button, as specified by the value of the Branding property.
ButtonTextTypeProperty	Read/write	A dependency property identifier that indicates the type of text displayed on the sign in button, as specified by the value of the TextType property.
clientId	Read/write	Specifies the client ID for the registered application.
ClientIdProperty	Read/write	A dependency property identifier that indicates the client ID for the registered application, as specified by the value of the ClientId property.
redirectUri	Read/write	The URI of the page to which the user is redirected after clicking the sign in button. It can be a relative URI. If this property is null or not

		specified, the redirect URI is assumed to be the URI of the current web page.
redirectUriProperty	Read/write	A dependency property identifier that indicates the URI of the redirect page (the page to which the user is redirected after clicking the sign in button), as specified by the value of the redirectUri property.
Scopes	Read/write	A list of OAuth scopes used in Windows Live sign in. Scopes specify the type of resource data that an application can request from an authenticated user. The value of this property cannot include the value offline_access.
ScopesProperty	Read/write	A dependency property identifier that indicates the OAuth scopes used in Windows Live sign in, as specified by the value of the Scopes property.
Session	Read/write	Identifies the session and specified scope for an authenticated user logged in from a particular application.
SignInText	Read/write	The text to show on the sign in button before the user signs in. The value of TextType specifies the button text (Sign in, Log in, Connect, or user-defined custom text).
SignInTextProperty	Read/write	A dependency property identifier that indicates the text to show on the sign in button before the user signs in, as specified by the value of the SignInText property.
SignOutText	Read/write	The text to show on the sign in button after the user has signed in. The value of TextType specifies the button text (Sign out, Log out, or user-defined custom text).
SignOutTextProperty	Read/write	
TextType	Read/write	Specifies the sign in button text. The values can be Sign in (the default), Login, Connect, or Custom (for user-

defined text). For each TextType value specified, there is a separate sign in text and sign out text.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

© 2011 Microsoft Corporation. All rights reserved.

SignInButton.Branding property

[This documentation is preliminary and is subject to change.]

Specifies the type of branding to display on the sign in button. The brand is the icon and visual theme displayed on the button, relating one of the following products: Hotmail, Windows Live Messenger, SkyDrive, Live Connect, or none. The brands are defined by the [BrandingType](#) enumeration.

Branding is a dependency property backed by [BrandingProperty](#).

Syntax

C#

```
public BrandingType Branding { get; set; }
```

Property value

Type: [BrandingType](#)

An enumeration that defines the branding themes that can be displayed on the sign in button.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[BrandingProperty Property](#)
[BrandingType Enumeration](#)

SignInButton.BrandingProperty property

[This documentation is preliminary and is subject to change.]

A dependency property identifier that indicates the type of branding to display on the sign in button, as specified by the value of the [Branding](#) property.

Syntax

C#

```
public DependencyProperty BrandingProperty { get; set; }
```

Property value

Type: [DependencyProperty](#)

A dependency property identifier, which can be set through data binding, styling, and inheritance.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[Branding](#)

SignInButton.ButtonTextTypeProperty property

[This documentation is preliminary and is subject to change.]

A dependency property identifier that indicates the type of text displayed on the sign in button, as specified by the value of the [TextType](#) property.

Syntax

C#

```
public DependencyProperty ButtonTextTypeProperty { get; set; }
```

Property value

Type: [DependencyProperty](#)

A dependency property identifier, which can be set through data binding, styling, and inheritance.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[TextType](#)

SignInButton.ClientId property

[This documentation is preliminary and is subject to change.]

Specifies the client ID for the registered application.

ClientId is a dependency property backed by [ClientIdProperty](#).

Syntax

C#

```
public string ClientId { get; set; }
```

Property value

Type: [String](#)

A string representing the application's client ID.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)

SignInButton.ClientIdProperty property

[This documentation is preliminary and is subject to change.]

A dependency property identifier that indicates the client ID for the registered application, as specified by the value of the [ClientId](#) property.

Syntax

C#

```
public DependencyProperty ClientIdProperty { get; set; }
```

Property value

Type: [DependencyProperty](#)

A dependency property identifier, which can be set through data binding, styling, and inheritance.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[ClientId](#)

SignInButton.Initialize method

[This documentation is preliminary and is subject to change.]

Initializes a new instance of the [SignInButton](#) class with default property values.

Syntax

C#

```
public Initialize()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[SignInButton](#)

SignInButton.InitializeComponent method

[This documentation is preliminary and is subject to change.]

Initializes XAML components in constructors for WPF forms or controls. This method is called from the code-behind class's constructor to merge the UI that is defined in XAML markup with the code-behind class, set properties, and register event handlers.

Syntax

C#

```
public InitializeComponent()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)

SignInButton.redirectUri property

[This documentation is preliminary and is subject to change.]

The URI of the page to which the user is redirected after clicking the sign in button. It can be a relative URI. If this property is null or not specified, the redirect URI is assumed to be the URI of the current web page.

redirectUri is a dependency property backed by [redirectUriProperty](#).

Syntax

C#

```
public string redirectUri { get; set; }
```

Property value

Type: **String**

A string that specifies the redirect URI.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)

SignInButton.redirectUriProperty property

[This documentation is preliminary and is subject to change.]

A dependency property identifier that indicates the URI of the redirect page (the page to which the user is redirected after clicking the sign in button), as specified by the value of the [redirectUri](#) property.

Syntax

C#

```
public DependencyProperty redirectUriProperty { get; set; }
```

Property value

Type: [DependencyProperty](#)

A dependency property identifier, which can be set through data binding, styling, and inheritance.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[redirectUri](#)

SignInButton.Scopes property

[This documentation is preliminary and is subject to change.]

A list of OAuth scopes used in Windows Live sign in. Scopes specify the type of resource data that an application can request from an authenticated user. The value of this property cannot include the value offline_access.

Scopes is a dependency property backed by [ScopesProperty](#).

Syntax

C#

```
public string Scopes { get; set; }
```

Property value

Type: **String**

A string that specifies the set of scopes for the sign in process.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)

SignInButton.ScopesProperty property

[This documentation is preliminary and is subject to change.]

A dependency property identifier that indicates the OAuth scopes used in Windows Live sign in, as specified by the value of the [Scopes](#) property.

Syntax

C#

```
public DependencyProperty ScopesProperty { get; set; }
```

Property value

Type: [DependencyProperty](#)

A dependency property identifier, which can be set through data binding, styling, and inheritance.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[Scopes](#)

SignInButton.Session property

[This documentation is preliminary and is subject to change.]

Identifies the session and specified scope for an authenticated user logged in from a particular application.

Syntax

C#

```
public LiveConnectSession Session { get; set; }
```

Property value

Type: **LiveConnectSession**

The session object.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)

SignInButton.SessionChanged event

[This documentation is preliminary and is subject to change.]

Raises change notifications for public properties, and notifies clients that a property value has changed.

Syntax

C#

```
public event TBD SessionChanged
```

Event information

Delegate	TBD
-----------------	-----

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live
Assembly	Microsoft.Live.dll

See also

[SignInButton](#)

SignInButton.SignInButton method

[This documentation is preliminary and is subject to change.]

Creates a new instance of the [SignInButton](#) class.

Syntax

C#

```
public SignInButton()
```

Parameters

This method has no parameters.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[InitializeComponent](#)

SignInButton.SignInText property

[This documentation is preliminary and is subject to change.]

The text to show on the sign in button before the user signs in. The value of **TextType** specifies the button text (Sign in, Log in, Connect, or user-defined custom text).

SignInText is a dependency property backed by [SignInTextProperty](#).

Syntax

C#

```
public string SignInText { get; set; }
```

Property value

Type: **String**

A string that represents the button's sign in text.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[SignOutText](#)
[TextType](#)

SignInButton.SignInTextProperty property

[This documentation is preliminary and is subject to change.]

A dependency property identifier that indicates the text to show on the sign in button before the user signs in, as specified by the value of the [SignInText](#) property.

Syntax

C#

```
public DependencyProperty SignInTextProperty { get; set; }
```

Property value

Type: [DependencyProperty](#)

A dependency property identifier, which can be set through data binding, styling, and inheritance.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[SignInText](#)

SignInButton.SignOutText property

[This documentation is preliminary and is subject to change.]

The text to show on the sign in button after the user has signed in. The value of **TextType** specifies the button text (Sign out, Log out, or user-defined custom text).

SignOutText is a dependency property backed by **SignOutTextProperty**.

Syntax

C#

```
public string SignOutText { get; set; }
```

Property value

Type: **String**

A string that represents the button's sign out text.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[SignInText](#)
[TextType](#)

SignInButton.SignOutTextProperty property

[This documentation is preliminary and is subject to change.]

A dependency property identifier that indicates the text to show on the sign in button after the user has signed in, as specified by the value of the [SignOutText](#) property.

Syntax

C#

```
public DependencyProperty SignOutTextProperty { get; set; }
```

Property value

Type: [DependencyProperty](#)

A dependency property identifier, which can be set through data binding, styling, and inheritance.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview
Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[SignOutText](#)

SignInButton.TextType property

[This documentation is preliminary and is subject to change.]

Specifies the sign in button text. The values can be Sign in (the default), Login, Connect, or Custom (for user-defined text). For each TextType value specified, there is a separate sign in text and sign out text.

TextType is a dependency property backed by [ButtonTypeProperty](#).

Value	Sign in text	Sign out text	Description
Signin	Sign in	Sign out	This is the default value.
Login	Log in	Log out	
Connect	Connect	Sign out	
Custom	(user-defined text)	(user-defined text)	The custom text string is defined on the website. If the website does not specify the custom text string, the default text is used, according to the value of TextType.

Syntax

C#

```
public ButtonType TextType { get; set; }
```

Property value

Type: [ButtonType](#)

An enumeration that defines the sign in button text type.

Requirements

Minimum supported client	Windows Developer Preview
Minimum supported phone	Windows Phone 7
Minimum supported server	Windows Server Developer Preview

Namespace	Microsoft.Live.Controls
Assembly	Microsoft.Live.Controls.dll

See also

[SignInButton](#)
[SignInText](#)
[SignOutText](#)

© 2011 Microsoft Corporation. All rights reserved.

Messenger XMPP reference

[This documentation is preliminary and is subject to change.]

The Windows Live Messenger Extensible Messaging and Presence Protocol (XMPP) service supports only these extensions as defined by the XMPP Standards Foundation, with partial support for these extensions as noted. For more info, see [XMPP Standards Foundation - XMPP Extensions](#).

- [RFC6120](#): XMPP: Core
- [RFC6121](#): XMPP: Instant Messaging and Presence. Roster management is not supported.
- XEP-0054: vcard-temp. The Messenger XMPP service supports fetching vCards but doesn't support updating vCards.
- XEP-0085: Chat State Notifications.
- XEP-0203: Delayed Delivery

© 2011 Microsoft Corporation. All rights reserved.

Server-side scenarios

[This documentation is preliminary and is subject to change.]

There may be scenarios in which you do not want to (or cannot) use the Live Connect JavaScript API or client-side JavaScript code. In these scenarios, you can use server-side scripts to interact directly with Live Connect and Representational State Transfer (REST) API.

Note You can use *JavaScript Objection Notation with padding* (JSONP) in client-side JavaScript webpage code to interact directly with the Microsoft Live Connect REST API. For details, see [JSONP Scenarios](#).

To write scripts that exercise the Live Connect server-side scenario, you would typically do the following:

1. Prompt the user to sign in to Live Connect and to consent to the scopes that you have requested.
2. If the user successfully signs in and consents, capture the *authorization code* that the Live Connect authorization web service returns to you.
3. Use the authorization code to call the Live Connect authorization web service to obtain an access token (and a refresh token, if the user consented to the **wl.offline_access** scope).
4. Use the access token to call the Live Connect REST API to work with the user's info, depending on the scopes for which consent has been granted. Repeat this step as needed until the Live Connect REST API indicates that the access token has expired.
 - a. If the access token has expired, use the refresh token (if you have one) to obtain a new access token, and then go back to the beginning of step 4.
 - b. If the refresh token has also expired, go back to step 1.

The preceding steps are explained in greater detail in the following sections.

Getting an authorization code

The first step to obtaining an access token (and a refresh token, if the user consented to a **wl.offline_access** scope request) is to obtain an authorization code. In a later step, you exchange this authorization code for an access token (and, optionally, a refresh token). To obtain an authorization code, make an HTTP **GET** call from a server-side script to the Live Connect authorization web service by using the following URL.

`https://oauth.live.com/authorize?client_id=CLIENT_ID&scope=SCOPES&response_type=code&redirect_uri=REDIRECT_URI`

In the preceding URL, replace the following elements:

- Replace **CLIENT_ID** with your application's client ID.
- Replace **SCOPES** with the scopes that you want the user to give you permission to access. Separate each requested scope with the URL escape code %20.
- Note** If you want to obtain a refresh token in addition to an access token, you must also include the **wl.offline_access** scope in the **scope** query parameter. (A refresh token enables you to obtain a replacement access token to interact directly with the Live Connect REST API on behalf of a user after the user ends his or her web browsing session.)
- Replace **REDIRECT_URI** with the URI of your callback webpage. The callback URI must use URL escape codes, such as %20 for spaces, %3A for colons, and %2F for forward slashes.

Note Optionally, you can also append the following query parameters to the preceding URL:

- A **display** query parameter, which represents a request for the display mode of the consent dialog box, can be set to **popup**, **touch**, or **none**.
- A **locale** query parameter, which is used by the request for consent dialog box to display localized user-interface text in the user's preferred language. This query parameter can be set to a market value such as **en**. For details, see [Supported locales](#).
- A **state** query parameter, which represents any arbitrary text string that you may want to pass on to the redirect URL for your own use.

For example, the following URL uses a client ID of 0000000603DB0F, the **wl.signin** and **wl.basic** scopes, and a redirect URI of `http://www.contoso.com/callback.php`.

`https://oauth.live.com/authorize?client_id=0000000603DB0F&scope=wl.signin%20wl.basic&response_type=code&redirect_uri=http%3A%2F%2Fwww.co`

When the URL is called from a webpage, the Live Connect authorization web service checks to determine whether there is a signed-in user. If the user is not signed in, he or she is prompted to sign in to Live Connect. After the user signs in, the Live Connect authorization web service checks to see whether the user has previously consented to the requested scopes. If the user has not previously consented, he or she is prompted to consent to the requested scopes. If the user consents, the Live Connect authorization web service sends the authorization code to the redirect URI, with the authorization code appended as a query parameter named **code**, as shown in the following example.

`http://www.contoso.com/callback.php?code=2bd12503-7e88-bfe7-c5c7-82274a740ff`

Retain this authorization code for use in the next section.

Getting an access token (and an optional refresh token)

After you have obtained an authorization code, you exchange it for an access token (and a refresh token, if the user consented to a **wl.offline_access** scope request). To obtain an access token (and, optionally, a refresh token), make an HTTP **POST** call to the Live Connect authorization web service by using the following URL.

`https://oauth.live.com/token`

The Content-Type header should have the value "application/x-www-form-urlencoded". Construct the request body as follows.

`client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&client_secret=CLIENT_SECRET&code=AUTHORIZATION_CODE&grant_type=authorization_code`

In the preceding request body, replace the following:

- Replace *CLIENT_ID* with your application's client ID.
- Replace *REDIRECT_URI* with the URI to your callback webpage. This URI must be the same as the URI that you specified when you requested an authorization code. The URI must use URL escape codes, such as %20 for spaces, %3A for colons, and %2F for forward slashes.
- Replace *CLIENT_SECRET* with your application's client secret.
- Replace *AUTHORIZATION_CODE* with the authorization code that you obtained earlier.

For example, the following request body uses a client ID of 0000000603DB0F, a redirect URI of <http://www.contoso.com/callback.php>, a client secret of LWILIT555GicSrIATma5qgyBXebRI, and an authorization code of 2bd12503-7e88-bfe7-c5c7-82274a740ff.

```
client_id=0000000603DB0F&redirect_uri=http://www.contoso.com/callback.php&client_secret=LWILIT555GicSrIATma5qgyBXebRI&code=2bd12503-7e88-bfe7-c5c7-82274a740ff
```

If the call is successful, the response for the HTTP **POST** request contains an access token. This access token is represented as a string of characters, in an **access_token** key/value pair inside a JSON-formatted object, as shown in the following example.

```
{  
    "access_token": "EwCo....//access token string shortened for brevity//...AA==",  
    "expires_in": 3600,  
    "scope": "wl.signin wl.basic",  
    "token_type": "bearer"  
}
```

Note A typical access token consists of well over 550 characters, so part of the access token string is omitted for brevity in the preceding example for brevity.

If you added **wl.offline_access** to the list of requested scopes in the previous section, and the user consented to your request, the Live Connect authorization web service instead sends to the callback URI an access token and a refresh token. These tokens are represented as strings of characters, in the **access_token** and **refresh_token** key/value pairs, respectively, inside a JSON-formatted object, as shown in the following example.

```
{  
    "access_token": "EwCo....//access token string shortened for brevity//...AA==",  
    "expires_in": 3600,  
    "refresh_token": "*LA9....//refresh token string shorted for brevity//...k%65",  
    "scope": "wl.offline_access wl.signin wl.basic",  
    "token_type": "bearer"  
}
```

Note A typical refresh token consists of well over 200 characters, so part of the refresh token string is omitted for brevity in the preceding example.

Retain this access token (and the refresh token, if you obtained one) for use in the next sections.

Using an access token to work with a user's info

After you have obtained an access token, you can use it to work with the consenting user's info. For example, to make an HTTP **GET** call to the Live Connect REST API to get user info, use the following URL.

```
https://apis.live.net/v5.0/REST_PATH?access_token=ACCESS_TOKEN
```

In the preceding code, replace the following:

- Replace *REST_PATH* with the REST path to the portion of the user's info that you want; for example, me.
- Replace *ACCESS_TOKEN* with your access token.

For example, the following URL uses a REST path of me, and an access token that starts with the characters EwCo and ends with the characters AA==.

```
https://apis.live.net/v5.0/me?access_token=EwCo....//access token string shortened for brevity//...AA==
```

If the HTTP **GET** call is successful, the Live Connect REST API sends the requested info as a JSON-formatted object or collection, depending on the shape of the info, as shown in the following example.

```
{  
    "id": "b6b2a7e8f2515e5",  
    "name": "Apurva Dalia",  
    "first_name": "Apurva",  
    "last_name": "Dalia",  
    "gender": null,  
    "link": "http://cid-b6b2a7e8f2515e5.profile.live.com/",  
    "locale": "en_US",  
    "updated_time": "2011-10-26T21:13:05+0000"  
}
```

Using a refresh token to get a new access token and a new refresh token

If you have obtained a refresh token, and the access token has expired, you can obtain a replacement access token and a replacement refresh token. To obtain replacement tokens, make an HTTP **POST** call to the Live Connect authorization web service at the following URL.

```
https://oauth.live.com/token
```

The Content-Type header should have the value "application/x-www-form-urlencoded". Construct the request body as follows.

```
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&client_secret=CLIENT_SECRET&refresh_token=REFRESH_TOKEN&grant_type=refresh_token
```

In the preceding request body replace the following:

- Replace *CLIENT_ID* with your application's client ID.

- Replace *REDIRECT_URI* with the URI to your callback webpage. This URI must be the same as the URI that you specified when you requested an authorization code. The URI must use URL escape codes, such as %20 for spaces, %3A for colons, and %2F for forward slashes.
- Replace *CLIENT_SECRET* with your application's client secret.
- Replace *REFRESH_TOKEN* with the refresh token that you obtained earlier.

For example, the following request body uses a client ID of 0000000603DB0F, a redirect URI of <http://www.contoso.com/callback.php>, a client secret of LWILIT555GicSrIATma5qgyBXebRI, and a refresh token starting with *LA9 and ending with xRoX.

```
client_id=0000000603DB0F&redirect_uri=http%3A%2F%2Fwww.contoso.com%2Fcallback.php&client_secret=LWILIT555GicSrIATma5qgyBXebRI&refresh_t...
```

If the call is successful, the Live Connect authorization web service sends to the callback URI a replacement access token and a replacement refresh token. These tokens are represented as strings of characters, in the **access_token** and **refresh_token** key/value pairs, respectively, inside a JSON-formatted object, as shown in the following example.

```
{
  "access_token": "FxDP....//access token string shortened for brevity//...BB==",
  "expires_in": 3600,
  "refresh_token": "*MB0....//refresh token string shorted for brevity//...l%76",
  "scope": "wl.offline_access wl.signin wl.basic",
  "token_type": "bearer"
}
```

PHP code sample

This section provides a code sample (designed and tested with PHP version 5.3.6.0) that demonstrates how to call the Live Connect authorization web service and REST API by using PHP scripts. This code sample consists of the following scripts:

- **signin.php**, which enables a user to sign in to Live Connect and consent to the specified scopes.
- **callback.php**, which wraps calls to the `tokens_rest_api.php` script.
- **tokens_rest_api.php**, which interacts directly with the Live Connect authorization web service and REST API.
- **tokens_mysql.php**, which retrieves and stores access and refresh tokens in a MySQL database.
- **globals.php**, which defines several constants, global variables, and a logging function, that are used by the preceding scripts.
- **callback_mysql.php**, which can be used in place of `signin.php` and `callback.php` to call the Live Connect authorization web service and REST API offline or unattended.

signin.php

This script enables a user to sign in to Live Connect and consent to the specified scopes. If the user successfully signs in and consents, this script attempts to call the Live Connect authorization web service to obtain an authorization code. If the call is successful, the Live Connect authorization web service calls the `callback.php` script and passes the authorization code to it.

```
<?php
include_once 'globals.php';

echo <<<TOP
<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd'>

<html xmlns='http://www.w3.org/1999/xhtml'>
  <head>
    <title>Sign in</title>
  </head>
  <body>
    <a href='
TOP;

/* Example authorization code URL:
https://oauth.live.com/authorize?
client_id=0000000603DB0FC
&scope=wl.offline_access%20wl.basic
&response_type=code
&redirect_uri=http%3A%2F%2Fwww.contoso.com%2Fcallback.php
*/
echo ENDPOINT_OAUTH . ENDPOINT_PATH_AUTHORIZE .
PARAM_CLIENT_ID . APP_CLIENT_ID .
PARAM_SCOPE . REQUESTED_SCOPES .
PARAM_RESPONSE_TYPE . PARAM_RESPONSE_TYPE_CODE .
PARAM_REDIRECT_URI . APP_REDIRECT_URI;

echo <<<BOTTOM
  '>Sign in</a>
</body>
</html>
BOTTOM;
?>
```

The preceding script is very straightforward. It simply displays a hyperlink for the user to click. After the user clicks the hyperlink, the following happens:

1. The script calls the attempts to call the Live Connect authorization web service (at <https://oauth.live.com/authorize>), passing to the web service the application's client ID, the requested scopes, a request for an authorization code if successful, and the URL of the script to which the authorization code should be passed.
2. If the user is not already signed in to Live Connect, Live Connect prompts the user to sign in.
3. If the user successfully signs in, the Live Connect authorization web service checks to see whether the user has previously consented to the requested scopes. If the user has not previously consented, the Live Connect authorization web service prompts the user to consent.
4. If the user consents, the Live Connect authorization web service calls the `callback.php` script and passes the authorization

code to it.

callback.php

This script calls the tokens_rest_api.php script to obtain an access token and then uses the token to call the Live Connect REST API.

```
<?php
include_once 'globals.php';
include_once 'tokens_rest_api.php';
include_once 'tokens_mysql.php';

$DISABLE_SSL_CHECKING = true;
$USE_MYSQL = true;

if (isset($_GET['code'])) {
    if (!$tokens = getTokens(APP_CLIENT_ID, APP_REDIRECT_URI, APP_CLIENT_SECRET, null, $_GET['code'])) {
        logMessage('Error retrieving tokens. Code execution has stopped.');
        die();
    }
} else {
    logMessage('Error retrieving authorization code, likely due to malformed or missing \'.
        'URL parameter \'code\'. Code execution has stopped.');
    die();
}
$results = callRestApi($tokens['access_token'],
    REST_PATH_ME,
    REST_API_GET);
if (!$results) {
    logMessage('Error calling REST API. Code execution has stopped.');
    die();
}
else {
    logMessage('REST API results:');
    logMessage(var_dump($results));
    if (array_key_exists('error', $results)) {
        logMessage('Error calling REST API. Code execution has stopped.');
        die();
    }
    if ($USE_MYSQL) {
        if (array_key_exists('refresh_token', $tokens)) {
            if (!setTokensInMySQL(APP_CLIENT_ID, $results['id'],
                REQUESTED_SCOPES, $tokens['access_token'], $tokens['refresh_token'])) {
                logMessage('Error setting access and refresh token in MySQL.');
            }
            else {
                logMessage('Access and refresh token successfully set in MySQL.');
            }
        }
        else {
            if (!setTokensInMySQL(APP_CLIENT_ID, $results['id'],
                REQUESTED_SCOPES, $tokens['access_token'])) {
                logMessage('Error setting access token in MySQL.');
            }
            else {
                logMessage('Access token successfully set in MySQL.');
            }
        }
    }
}
?>
```

The following happens in the preceding script:

1. The script checks to ensure that the authorization code is present, and if so, calls the `getTokens` function in the `tokens_rest_api.php` script to use the authorization code to get an access token (and, optionally, a refresh token).
2. The script then uses the `callRestApi` function in the `tokens_rest_api.php` script to call the Live Connect REST API and displays the results of this call.
3. If the call is successful, and if the `$USE_MYSQL` global variable is set to **true**, the script calls the `tokens_mysql.php` script to attempt to store the access token (and, optionally, the refresh token) in a MySQL server.

Note

In this version of the code sample, the `$DISABLE_SSL_CHECKING` global variable, if set to **true**, disables SSL certificate validation. This variable is helpful if you ever need to troubleshoot the Live Connect authorization web service to see whether SSL certificate validation is causing a problem.

tokens_rest_api.php

This script uses an authorization code (or a refresh token, if one exists) to obtain an access token (and a replacement refresh token, if an existing refresh token is available). This script also uses an access token to call the Live Connect REST API.

```
<?php
include_once 'globals.php';

/* Example callback URL:
http://www.contoso.com/callback.php
?code=8f61a46f-717c-793a-ee27-bdfe33153b25

Example acquire tokens URL using authorization code:
https://oauth.live.com/token?
client_id=00000000603DB0FC
&redirect_uri=http%3A%2F%2Fwww.contoso.com%2Fcallback.php
&client_secret=MLWIL1t55GicSriATma5qgyBkebRIKt
&code=8f61a46f-717c-793a-ee27-bdfe33153b25
&grant_type=authorization_code
```

```

Example acquire tokens URL using refresh token:
https://oauth.live.com/token?
client_id=00000000603DB0FC
&redirect_uri=http%3A%2F%2Fwww.contoso.com%2Fcallback.php
&client_secret=MLWIL1t555GicSrIAfma5qgyBKeRIKT
&refresh_token=LA9Y...full refresh token omitted for brevity...xRoX
&grant_type=refresh_token
*/
function getTokens($clientID, $redirectURI, $clientSecret, $refreshToken = null, $authorizationCode = null) {
    if (isset($authorizationCode)) {
        $getTokensURL = ENDPOINT_OAUTH . ENDPOINT_PATH_TOKEN .
            PARAM_CLIENT_ID . $clientID .
            PARAM_REDIRECT_URI . $redirectURI .
            PARAM_CLIENT_SECRET . $clientSecret .
            PARAM_CODE . $authorizationCode .
            PARAM_GRANT_TYPE . PARAM_GRANT_TYPE_AUTHORIZATION_CODE;
    }
    elseif (isset($refreshToken)) {
        $getTokensURL = ENDPOINT_OAUTH . ENDPOINT_PATH_TOKEN .
            PARAM_CLIENT_ID . $clientID .
            PARAM_REDIRECT_URI . $redirectURI .
            PARAM_CLIENT_SECRET . $clientSecret .
            PARAM_REFRESH_TOKEN . $refreshToken .
            PARAM_GRANT_TYPE . PARAM_GRANT_TYPE_REFRESH_TOKEN;
    }
    else {
        logMessage('Error retrieving tokens. Cannot determine whether to use ' .
            'refresh token or verification code.');
        return false;
    }

    logMessage('Request URL for token(s): ' . $getTokensURL);
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $getTokensURL);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    if (SGLOBALS['DISABLE_SSL_CHECKING']) {
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
    }

    $json = curl_exec($ch);

    if (curl_errno($ch)) {
        logMessage('Error retrieving tokens: ' . curl_error($ch));
        return false;
    }

    $decodedJSON = json_decode($json, true);

    logMessage('Response for tokens request:');
    $info = curl_getinfo($ch);
    logMessage(var_dump($info));
    logMessage(var_dump($decodedJSON));

    curl_close($ch);

    if (array_key_exists('error', $decodedJSON)) {
        logMessage('Error retrieving tokens: ');

        if (array_key_exists('message', $decodedJSON)) {
            logMessage($decodedJSON['error']['message']);
        }
        else {
            logMessage($decodedJSON['error_description']);
        }

        return false;
    }

    return $decodedJSON;
}

/* Example REST API URL:
https://apis.live.net/v5.0/me/
?oauth_token=EwCo.....EkAA

Example REST API GET call:
var_dump(callRestApi(
    'EwCo...full access token omitted for brevity...EkAA',
    REST_PATH_ME . REST_PATH_CONTACTS,
    REST_API_GET));
*/

Example REST API POST call:
$activityData = array('message' => 'Explore Windows Live',
    'link' => 'http://explore.live.com/home',
    'description' => 'Stay in touch and share your world: email, photos, movies, video chat, and more!',
    'picture' => 'http://res2.explore.live.com/resbox/en/Live%20Explore/Main/a/1/alce31bd-6291-42fb-9b51-b4cf2013481c/alce31bd-6291-42fb-9b51-b4cf2013481c.jpg',
    'name' => 'Windows Live Home');
$jsonActivityData = json_encode($activityData);

var_dump(callRestApi(
    'EwCo...part of access token omitted for brevity...EkAA',
    REST_PATH_ME . REST_PATH_SHARE,
    REST_API_POST,
    array('Content-Type: application/json')));

```

```

$jsonActivityData));
*/
function callRestApi($accessToken, $restPath, $requestMethod, $headers = NULL, $methodData = NULL) {
    $restApiURL = ENDPOINT_REST_API . ENDPOINT_REST_API_VERSION .
        $restPath . PARAM_ACCESS_TOKEN . $accessToken;

    logMessage('Request URL for REST API: ' . $restApiURL);

    $ch = curl_init();

    curl_setopt($ch, CURLOPT_URL, $restApiURL);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    if (isset($GLOBALS['DISABLE_SSL_CHECKING'])) {
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
    }

    switch ($requestMethod) {
        case REST_API_GET:
            curl_setopt($ch, CURLOPT_HTTPGET, true);
            break;
        case REST_API_POST:
            curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
            curl_setopt($ch, CURLOPT_POST, true);
            curl_setopt($ch, CURLOPT_POSTFIELDS, $methodData);
            break;
        case REST_API_PUT:
            curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
            curl_setopt($ch, CURLOPT_PUT, true);
            curl_setopt($ch, CURLOPT_INFILE, $methodData);
            curl_setopt($ch, CURLOPT_INFILESIZE, filesize($methodData));
            break;
        case REST_API_DELETE:
            curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
            break;
        default:
            logMessage('Error calling REST API. Cannot determine whether to ' .
                'use GET, POST, PUT, or DELETE request method.');
            return false;
    }

    $json = curl_exec($ch);

    if (curl_errno($ch)) {
        logMessage('Error retrieving data from REST API: ' . curl_error($ch));
        return false;
    }

    $decodedJSON = json_decode($json, true);

    logMessage('Response from REST API:');
    $info = curl_getinfo($ch);
    logMessage(var_dump($info));
    logMessage(var_dump($decodedJSON));

    curl_close($ch);

    if (array_key_exists('error', $decodedJSON)) {

        logMessage('Error retrieving data from REST API: ');

        if (array_key_exists('message', $decodedJSON['error'])) {
            logMessage($decodedJSON['error'][['message']]);
        } else {
            logMessage($decodedJSON['error_description']);
        }

        return false;
    } else {
        return $decodedJSON;
    }
}
?>

```

The preceding script consists of the following functions:

- `getTokens`, which attempts to use an authorization code (if one is specified) or a refresh token (if one is specified) to obtain an access token (and optionally, a new or replacement refresh token).
- `callRestApi`, which attempts to call the Live Connect REST API. The result of the call is a JSON-formatted object that contains the requested info, such as a list of the signed-in user's contacts. However, if the first key in this JSON-formatted object is named 'error', the `callRestApi` function fails.

tokens_mysql.php

This script attempts to retrieve and store access and refresh tokens in a MySQL database.

```

<?php
define('MYSQL_SERVER', '127.0.0.1');
define('MYSQL_USERNAME', 'root');
define('MYSQL_PASSWORD', 'mysql');

define('ACCESS_TOKEN_TYPE', 2);
define('REFRESH_TOKEN_TYPE', 3);

```

```

$LINK;

/*
The following constants assume the following MySQL database and table structure:
users.users: user_entry_id (AUTO_INCREMENT), user_id, user_name
apps.apps: app_entry_id(AUTO_INCREMENT), app_id, client_secret, redirect_uri
apps.tokens: tokens_entry_id (AUTO_INCREMENT), app_entry_id (matching entry in apps.apps),
              user_entry_id (matching entry in users.users), scopes, access_token, refresh_token
*/
define('APPS_DB', 'apps');
define('USERS_DB', 'users');
define('APPS_TABLE', APPS_DB . '.apps');
define('TOKENS_TABLE', APPS_DB . '.tokens');
define('USERS_TABLE', USERS_DB . '.users');

function getTokenFromMySQL($tokenType, $appId, $userId, $scopes) {
    if (verifyConnection() == true) {
        switch ($tokenType) {
            case ACCESS_TOKEN_TYPE:
                $tokenField = 'access_token';
                break;
            case REFRESH_TOKEN_TYPE:
                $tokenField = 'refresh_token';
                break;
            default:
                logMessage('Error determining whether to get access token or refresh token from MySQL.');
                return false;
        }
        $query = 'SELECT ' . $tokenField . ' FROM ' . TOKENS_TABLE . ' WHERE ' .
            'app_entry_id = ' .
            '(SELECT app_entry_id FROM ' . APPS_TABLE . ' WHERE app_id = \'' . $appId . '\'') AND ' .
            'user_entry_id = ' .
            '(SELECT user_entry_id FROM ' . USERS_TABLE . ' WHERE user_id = \'' . $userId . '\') AND ' .
            'scopes = \'' . $scopes . '\'';
        $results = mysql_query($query);

        if (!$results) {
            logMessage('Error using query \'' . $query . '\': ' . mysql_error());
            mysql_close($GLOBALS['LINK']);
            return false;
        } else {
            $requestedToken = array();

            while($row = mysql_fetch_array($results)) {
                array_push($requestedToken, $row[0]);
                logMessage('token = ' . $row[0]);
            }
            return $requestedToken;
        }
    } else {
        logMessage('Error getting token from MySQL.');
        mysql_close($GLOBALS['LINK']);
        return false;
    }
}

function setTokensInMySQL($appId, $userId, $scopes, $accessToken, $refreshToken = '') {
    if (verifyConnection() == true) {
        if (!getTokenFromMySQL(ACCESS_TOKEN_TYPE, $appId, $userId, $scopes)) {
            $query = 'INSERT INTO ' . TOKENS_TABLE .
                '(tokens_entry_id, app_entry_id, user_entry_id, scopes, access_token, refresh_token) VALUES(' .
                'NULL, ' .
                '(SELECT app_entry_id FROM ' . APPS_TABLE . ' WHERE app_id = \'' . $appId . '\'),' .
                '(SELECT user_entry_id FROM ' . USERS_TABLE . ' WHERE user_id = \'' . $userId . '\'),' .
                '\'' . $scopes . '\', ' .
                '\'' . $accessToken . '\', ' .
                '\'' . $refreshToken . '\')';
            $results = mysql_query($query);
            if (!$results) {
                logMessage('Error inserting token(s) as new entry in MySQL.');
                return false;
            } else {
                return true;
            }
        } else {
            $query = 'UPDATE ' . TOKENS_TABLE .
                ' SET access_token=\'' . $accessToken . '\', refresh_token=\'' . $refreshToken . '\' WHERE ' .
                'app_entry_id = ' .
                '(SELECT app_entry_id FROM ' . APPS_TABLE . ' WHERE app_id = \'' . $appId . '\')' .
                ' AND user_entry_id = ' .
                '(SELECT user_entry_id FROM ' . USERS_TABLE . ' WHERE user_id = \'' . $userId . '\')' .
                ' AND scopes = \'' . $scopes . '\'';
            $results = mysql_query($query);
            if (!$results) {
                logMessage('Error updating token(s) in existing entry in MySQL.');
                return false;
            } else {
                return true;
            }
        }
    } else {
        logMessage('Error writing token(s) to MySQL.');
        mysql_close($GLOBALS['LINK']);
        return false;
    }
}

```

```

}

function verifyConnection() {

    if (!isset($GLOBALS['LINK'])) {
        $GLOBALS['LINK'] = mysql_connect(MYSQL_SERVER, MYSQL_USERNAME, MYSQL_PASSWORD);
        if (!$GLOBALS['LINK']) {
            logMessage('Error connecting to MySQL server \'' . MYSQL_SERVER . '\'.' . mysql_error());
            return false;
        }
        return true;
    } else {
        return true;
    }
}
?>

```

The preceding script consists of the following functions, constants, and global variables:

- `getTokenFromMySQL`, which retrieves either an access token or a refresh token from the server, based on the specified application client ID, user ID, and the accompanying scopes to which the user previously consented.
- `setTokensInMySQL`, which stores in the server an access token or refresh token, or both, based on the specified application client ID, user ID, and the associated scopes to which the user previously consented. If the combination of client ID, user, and scopes already exists in the server, the existing values for the access token or refresh token, or both, are overwritten; otherwise, a new entry is created with values inserted for the new access token or refresh token, or both.
- `verifyConnection`, which ensures that a connection to the server exists before an attempt is made to retrieve or store server entries.
- A global variable, which is provided to persist the server connection.
- Constants to represent the MySQL server, the user account, the associated password, values that indicate whether to retrieve or store an access token or a refresh token, and the names of the databases and tables to be used.

For the `tokens_mysql.php` script to run correctly, the following databases, tables, and fields must already be present on the MySQL server.

```

-- apps database
CREATE DATABASE apps;
USE apps;
CREATE TABLE apps (
    app_entry_id mediumint(9) NOT NULL AUTO_INCREMENT,
    app_id text NOT NULL,
    client_secret text,
    redirect_uri text NOT NULL,
    PRIMARY KEY (app_entry_id)
);
CREATE TABLE tokens (
    tokens_entry_id mediumint(9) NOT NULL AUTO_INCREMENT,
    app_entry_id mediumint(9) NOT NULL,
    user_entry_id mediumint(9) NOT NULL,
    scopes text NOT NULL,
    access_token text NOT NULL,
    refresh_token text,
    PRIMARY KEY (tokens_entry_id)
);
-- apps sample data
INSERT INTO apps (app_entry_id, app_id, client_secret, redirect_uri) VALUES
    (NULL, '00000000603DB0FC', 'MLWILLT555GicSriATma5qgyBXebRIKT', 'http%3A%2F%2Fwww.contoso.com%2Fcallback.php');
INSERT INTO tokens (tokens_entry_id, app_entry_id, user_entry_id, scopes, access_token, refresh_token) VALUES
    (NULL, 1, 3, 'wl.basic wl.offline_access',
     'EwA4...full access token omitted for brevity...AA==',
     '*LA9...full refresh token omitted for brevity...4%24');

-- users database
CREATE DATABASE users;
USE users;
CREATE TABLE users (
    INSERT INTO tokens (tokens_entry_id, app_entry_id, user_entry_id, scopes, access_token, refresh_token) VALUES
    user_entry_id mediumint(9) NOT NULL AUTO_INCREMENT,
    user_id text NOT NULL,
    user_name text NOT NULL,
    PRIMARY KEY (user_entry_id)
);
-- users sample data
INSERT INTO users (user_entry_id, user_id, user_name) VALUES
    (NULL, '00091865', 'Roberto Tamburello'),
    (NULL, '00082974', 'Mariusz Wolodzko'),
    (NULL, '00073185', 'Henrik Jensen');

```

globals.php

This script defines several constants, global variables, and a function that are used by the preceding scripts. These constants, global variables, and function can help you debug and isolate run-time errors, especially those that are due to any changes to hard-coded URLs upon which the Live Connect authorization web service and REST API rely. These elements can also make it possible to reuse the preceding scripts in a testing environment with only minor changes to their contents.

```

<?php
define('APP_CLIENT_ID', '000000004C043701');
define('APP_CLIENT_SECRET', '09L3tC4ecekCdlDaNsNC3m4mdiBm9gYg2');
define('APP_REDIRECT_URI', 'http%3A%2F%2Fwww.contoso.com%2Fcallback.php');

define('ENDPOINT_OAUTH', 'https://oauth.live.com/');
define('ENDPOINT_PATH_AUTHORIZE', 'authorize?');
define('ENDPOINT_PATH_TOKEN', 'token?');

```

```

define('ENDPOINT_REST_API', 'https://apis.live.net/');
define('ENDPOINT_REST_API_VERSION', 'v5.0/');

define('PARAM_CODE', '&code=');
define('PARAM_CLIENT_ID', 'client_id=');
define('PARAM_CLIENT_SECRET', '&client_secret=');
define('PARAM_GRANT_TYPE', '&grant_type=');
define('PARAM_GRANT_TYPE_AUTHORIZATION_CODE', 'authorization_code');
define('PARAM_GRANT_TYPE_REFRESH_TOKEN', 'refresh_token');
define('PARAM_REDIRECT_URI', '&redirect_uri=');
define('PARAM_REFRESH_TOKEN', '&refresh_token=');
define('PARAM_RESPONSE_TYPE', '&response_type=');
define('PARAM_RESPONSE_TYPE_CODE', 'code');
define('PARAM_ACCESS_TOKEN', '?access_token=');

define('PARAM_SCOPE', '&scope=');
define('PARAM_SCOPE_WL', 'wl.');
define('PARAM_SCOPE_SIGNIN', 'signin');
define('PARAM_SCOPE_BASIC', 'basic');
define('PARAM_SCOPE_POSTAL_ADDRESSES', 'postal_addresses');
define('PARAM_SCOPE_PHONE_NUMBERS', 'phone_numbers');
define('PARAM_SCOPE_BIRTHDAY', 'birthday');
define('PARAM_SCOPE_CONTACTS_BIRTHDAY', 'contacts_birthday');
define('PARAM_SCOPE_EMAILS', 'emails');
define('PARAM_SCOPE_EVENTS_CREATE', 'events_create');
define('PARAM_SCOPE_PHOTOS', 'photos');
define('PARAM_SCOPE_CONTACTS_PHOTOS', 'contacts_photos');
define('PARAM_SCOPE_APPLICATIONS', 'applications');
define('PARAM_SCOPE_APPLICATIONS_CREATE', 'applications_create');
define('PARAM_SCOPE_WORK_PROFILE', 'work_profile');
define('PARAM_SCOPE_SHARE', 'share');
define('PARAM_SCOPE_OFFLINE', 'offline_access');

define('REQUESTED_SCOPES', PARAM_SCOPE_WL . PARAM_SCOPE_SIGNIN . ' .
PARAM_SCOPE_WL . PARAM_SCOPE_BASIC . ' .
PARAM_SCOPE_WL . PARAM_SCOPE_EVENTS_CREATE . ' .
PARAM_SCOPE_WL . PARAM_SCOPE_SHARE . ' .
PARAM_SCOPE_WL . PARAM_SCOPE_OFFLINE);

define('REST_API_GET', 2);
define('REST_API_POST', 3);
define('REST_API_PUT', 4);
define('REST_API_DELETE', 5);

define('REST_PATH_APPLICATIONS', 'applications');
define('REST_PATH_ALBUMS', 'albums');
define('REST_PATH_COMMENTS', 'comments');
define('REST_PATH_CONTACTS', 'contacts');
define('REST_PATH_EVENTS', 'events');
define('REST_PATH_FILES', 'files');
define('REST_PATH_FRIENDS', 'friends');
define('REST_PATH_ME', 'me/');
define('REST_PATH_SHARE', 'share');
define('REST_PATH_TAGS', 'tags');

$LOG_MESSAGE_FORMAT = 'HTML';

function logMessage($message) {
    switch ($GLOBALS['LOG_MESSAGE_FORMAT']) {
        case 'HTML':
            echo '<p>' . $message . '</p>';
            break;
        case 'TEXT':
            echo $message;
            break;
        default:
    }
}

?>

```

The first three constants are the most important, and their values will vary depending on your app's client ID, client secret, and redirect URI. Be sure to change these values before you try to call them from this code sample's other scripts. Other constants include the following:

- Components for building up well-formed URLs to call the Live Connect authorization web service and REST API.
- The REQUESTED_SCOPES constant, which can be used in scripts for consistency when tracking which scopes were originally requested.
- Constants to indicate whether the Live Connect REST API is to use **GET**, **PUT**, **POST**, or **DELETE** requests.

The `logMessage` function can be used to write debug messages to a specified output format, depending on the value of the global `$LOG_MESSAGE_FORMAT` variable, which is set to "HTML" output by default. In this version of the code sample, if `$LOG_MESSAGE_FORMAT` is set to any value other than "HTML" or "TEXT", no debug messages will be output.

callback_mysql.php

This script can replace both the `signin.php` and `callback.php` scripts to call the Live Connect authorization web service and REST API offline or unattended. To run this script, you must have the target user's ID already available. (It is hard-coded in the following script for illustrative purposes.)

```

<?php
include_once 'globals.php';
include_once 'tokens_mysql.php';
include_once 'tokens_rest_api.php';

$USER_ID = 'a6b2a7e8f251e5e';

```

```

$DISABLE_SSL_CHECKING = true;

if (!$tokens = getTokenFromMySQL(REFRESH_TOKEN_TYPE, APP_CLIENT_ID,
    $USER_ID, REQUESTED_SCOPES)) {
    logMessage('Error retrieving refresh token from MySQL. Code execution has stopped.');
    die();
}
else {
    if(!$tokens = getTokens(APP_CLIENT_ID, APP_REDIRECT_URI,
        APP_CLIENT_SECRET, $tokens[0])) {
        logMessage('Error retrieving replacement tokens. Code execution has stopped.');
        die();
    }
    else {
        $results = callRestApi($tokens['access_token'],
            REST_PATH_ME,
            REST_API_GET);

        if (!$results) {
            logMessage('Error calling REST API. Code execution has stopped.');
            die();
        }
        else {
            logMessage('REST API results:');
            logMessage(var_dump($results));

            if (array_key_exists('error', $results)) {
                logMessage('Error calling REST API. Code execution has stopped.');
                die();
            }
            if (array_key_exists('refresh_token', $tokens)) {

                if (!setTokensInMySQL(APP_CLIENT_ID, $USER_ID,
                    REQUESTED_SCOPES, $tokens['access_token'],
                    $tokens['refresh_token'])) {
                    logMessage('Error setting access and refresh token in MySQL.');
                }
                else {
                    logMessage('Access and refresh token successfully set in MySQL.');
                }
            }
            else {
                if (!setTokensInMySQL(APP_CLIENT_ID, $USER_ID,
                    REQUESTED_SCOPES, $tokens['access_token'])) {
                    logMessage('Error setting access token in MySQL.');
                }
                else {
                    logMessage('Access token successfully set in MySQL.');
                }
            }
        }
    }
}
?>

```

The preceding script does the following:

1. Calls the `getTokenFromMySQL` function in the `tokens_mysql.php` script to retrieve the refresh token for the specified Microsoft account credentials from the MySQL server.
2. Because access tokens are typically short-lived (they are usually valid for only one hour), uses the refresh token to call the `getTokens` function in the `tokens_rest_api.php` script. This function returns a replacement access token and a replacement refresh token.
3. Uses the access token to call the `callRestApi` function in the `tokens_rest_api.php` script. This function returns the results of the call to the Live Connect REST API.
4. If the call is successful, writes the replacement access token and replacement refresh token into the MySQL server.

OAuth 2.0

[This documentation is preliminary and is subject to change.]

Live Connect implements the [OAuth 2.0](#) protocol to authenticate users. This topic describes both the authorization flows that Live Connect uses and the supported extension parameters.

In this topic, we assume that you are familiar with OAuth 2.0 and OAuth terminology. If you're new to OAuth, we recommend that you check out the spec first, or at least refer to it when you come across a term or idea that's unfamiliar to you.

Supported OAuth flows

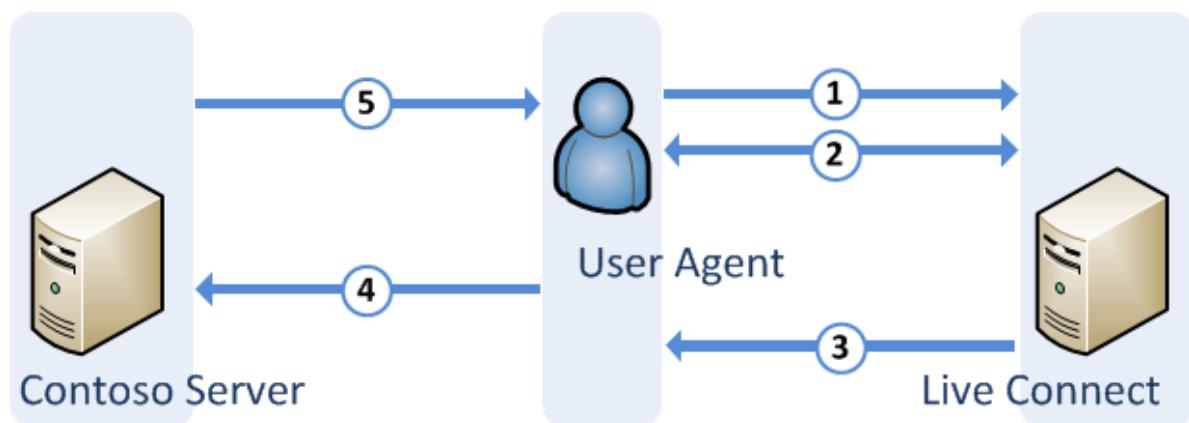
Live Connect supports the following authorization flows:

- [Implicit grant flow](#)
- [Authorization code grant flow](#)
- [Sign-In control flow](#)

Implicit grant flow

The implicit grant flow can be used by both web-based and desktop apps. In this flow, the client makes an authorization request to <https://oauth.live.com/authorize> with `request_type=token`. This is a standard OAuth 2.0 flow and is defined in full detail in the [implicit grant section of the OAuth 2.0 spec](#).

The following diagram illustrates how the implicit grant flow works.



1. The client starts the flow by directing the resource owner's user agent to the Windows Live authorization server's endpoint, by using a URL in the following format.

```
https://oauth.live.com/authorize?client_id=CLIENT_ID&scope=SCOPES&response_type=token&redirect_uri=REDIRECT_URL
```

This URL contains the client ID, requested scope(s), local state, and a redirection URI to which the authorization web service is to send the user agent after access is granted or denied.

2. The user is prompted for his or her sign-in credentials, and grants or denies the client's access request.

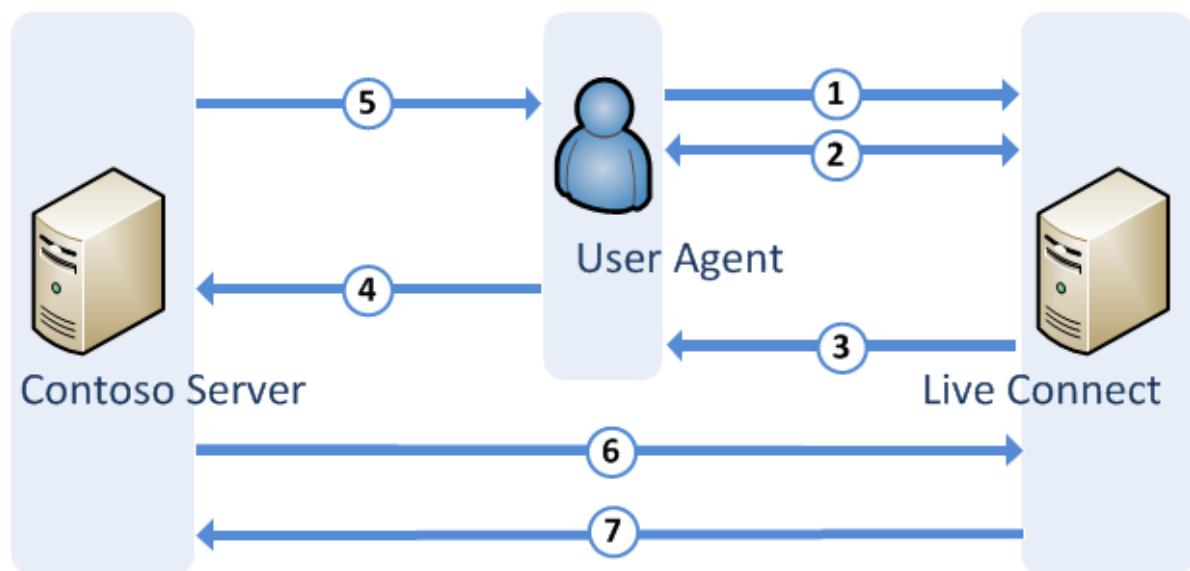
3. Assuming that the user has granted access, the Live Connect authorization server redirects the user agent back to the client by using the redirection URI that was provided in the initial request. The redirection URI includes an access token in the URI fragment. For example: `http://contoso.com/Callback.htm#access_token=[accesstoken]`.
4. The user agent follows the redirection instructions by making a request to the Contoso web server. The user agent retains the URI fragment locally but does not include it in the request to the server.
5. The Contoso server returns a webpage (typically an HTML document that contains an embedded script) that is capable of accessing the full redirection URI, including the fragment that was retained by the user agent. The script can also extract the access token and various other parameters that are contained in the fragment.

The user-agent locally executes the script that was provided by the web server, to extract the access token and pass it to the client.

Authorization code grant flow

In the authorization code grant flow, the client makes authorization requests by using `request_type=code`. It is a standard OAuth 2.0 flow, and is defined in full detail in the [authorization code grant section of the OAuth 2.0 spec](#). For further details, see [Server-side scenarios](#).

The following diagram illustrates how the authorization code grant flow works.



1. The client starts the flow by directing the resource owner's user agent to the Live Connect authorization endpoint, by using a URL in the following format.

```
https://oauth.live.com/authorize?client_id=CLIENT_ID&scope=SCOPES&response_type=code&redirect_uri=REDIRECT_URL
```

This URL contains the client ID, requested scope(s), local state, and a redirection URI to which the authorization web service is to send the user agent after access is granted or denied.

2. The authorization server authenticates the resource owner via the user-agent, and establishes whether the resource owner grants or denies the client's access

request.

3. Assuming that the user has granted access, the Live Connect authorization server redirects the user agent back to the client by using the redirection URI that was provided in the initial request.
4. The user agent calls the client with the redirection URI, which includes an authorization code and any local state that was provided by the client. For example: [http://contoso.com/Callback.htm?code=\[authorizationcode\]](http://contoso.com/Callback.htm?code=[authorizationcode]).
5. The client requests an access token from the authorization server's token endpoint by using its client credentials for authentication, and includes the authorization code that was received in the previous step. The client includes the redirection URI that was used to obtain the authorization code for verification. The request URL has the following format: https://oauth.live.com/token?client_id=CLIENT_ID&redirect_uri=REDIRECT_URL&client_secret=CLIENT_SECRET&code=AUTHORIZATION_CODE&grant_type=authorization_code.
6. The Live Connect authorization server validates the client credentials and the authorization code, and ensures that the redirection URI that was received matches the URI that was used to redirect the client in step 3.
7. If the credentials are valid, the authorization server responds by returning an access token. If the *wl.offline_access* scope was requested, then a refresh token is also returned. In order to refresh the access token, the client must make a request to the following URL.

```
https://oauth.live.com/token?client\_id=\[YOUR\_CLIENT\_ID\]&client\_secret=\[YOUR\_CLIENT\_SECRET\]&redirect\_uri=REDIRECT\_URL&grant\_type=refresh\_token&refresh\_token=\[REFRESH\_TOKEN\]
```

8. To make things easier for developers of desktop and mobile apps, the Live Connect OAuth 2.0 implementation provides a special redirect URL: <http://oauth.live.com/desktop>. This allows apps which can host a web browser control (such as a Java, Cocoa, or .NET app) to handle redirects after the user has provided consent, by listening for when the user is redirected to [http://oauth.live.com/desktop#access_token=\[access token\]](http://oauth.live.com/desktop#access_token=[access token]), then retrieving the access token from the end of the URL using whatever mechanisms are available in the chosen client platform. Because apps will also need to refresh access tokens, the Live Connect developer portal allows apps to be marked as mobile client apps. When this marker is specified and the special redirect URL (<https://oauth.live.com/desktop>) is used, the client secret is not required to refresh the access token. In this case, the URL for refreshing the access token would look like this.

```
https://oauth.live.com/token?client\_id=\[YOUR\_CLIENT\_ID\]&redirect\_uri=https://oauth.live.com/desktop&grant\_type=refresh\_token&refresh\_token=\[REFRESH\_TOKEN\]
```

Sign-in control flow

The sign-in control flow is unique to Live Connect. It uses the Live Connect Sign-In control to simplify sign-in and consent for apps. Intended for use by browser-based apps, it does not require the app to use any server-side logic. It is started by using the [JavaScript API](#) (the [WL.login](#) method) or by embedding an HTML control (by calling the [WL.ui](#) method) rather than by directly calling the consent service by using an HTTP request. The sign-in

control flow is the only web flow that supports seamless single sign-in across Live Connect and non-Microsoft web apps. *Single sign-in* is the mechanism by which a user who is already signed in to Live Connect can go to a different website and be automatically signed in to that site based on the Live Connect account. The user has control over this sign-in behavior on each website, on an opt-in basis.

The sign-in control flow is a variation of the implicit grant flow. The key difference between the two is that the sign-in control flow is implemented by using an invisible HTML **iframe** element that is embedded in the client webpage, whereas the implicit grant flow must be implemented by the client app.

Important Because the sign-in control flow requires no server-side processing, apps can access user info only after the page has loaded.

The sign-in control flow works as follows:

1. The client starts the flow by directing the resource owner's user agent to the Live Connect authorization endpoint, by using a URL with the following format:

```
https://oauth.live.com/authorize?client_id=CLIENT_ID&  
scope=SCOPES&response_type=code&redirect_uri=REDIRECT_URL.
```

This URL contains the client ID, requested scope(s), local state, and a redirection URI to which the authorization web service is to send the user agent after access is granted or denied.

2. The user is prompted for his or her credentials, and grants or denies the client's access request.

Authorization request parameters

The following table lists the OAuth authorization request parameters that are used by Live Connect.

Parameter	Notes
<i>client_id</i>	The app's client ID.
<i>display</i>	The display type to be used for the authorization page. Valid values are "popup", "touch", "page", or "none".
<i>locale</i>	Optional. A market string that determines how the consent user interface (UI) is localized. If the value of this parameter is missing or is not valid, a market value is determined by using an internal algorithm.
<i>redirect_uri</i>	Equivalent to the endpoint that is described in the OAuth 2.0 protocol spec.
<i>response_type</i>	The type of data to be returned in the response from the authorization server. Valid values are "code" or "token".
<i>scope</i>	Equivalent to the scope parameter that is described in the OAuth 2.0 protocol spec.
<i>state</i>	Equivalent to the state parameter that is described in the OAuth 2.0 protocol spec.

OAuth request and response parameters

The following table lists the OAuth request parameters that are used by Live Connect.

Parameter	Notes
<i>client_id</i>	The app's client ID.
<i>client secret</i>	The app's client secret.
<i>grant_type</i>	The authorization type that the server returns. Valid values are "authorization_code" or "refresh_token".
<i>locale</i>	Optional. A market string that determines how the consent UI is localized. If the value of this parameter is missing or is not valid, a market value is determined by using an internal algorithm.
<i>redirect_uri</i>	Equivalent to the endpoint that is described in the OAuth 2.0 protocol spec.
<i>response_type</i>	The type of data to be returned in the response from the authorization server. Valid values are "code" or "token".
<i>scope</i>	Equivalent to the scope parameter that is described in the OAuth 2.0 protocol spec.
<i>state</i>	Equivalent to the state parameter that is described in the OAuth 2.0 protocol spec.
<i>display</i>	The display type to be used for the authorization page. Valid values are "popup", "touch", "page", or "none".

The following table lists the OAuth Response parameters that are used by Live Connect.

Parameter	Notes
<i>access_token</i>	Equivalent to the profile that is described in the OAuth 2.0 protocol spec.
<i>auth_token</i>	The app's authentication token.
<i>code</i>	Equivalent to the profile that is described in the OAuth 2.0 protocol spec.
<i>expires_in</i>	Equivalent to the profile that is described in the OAuth 2.0 protocol spec.
<i>refresh_token</i>	Equivalent to the profile that is described in the OAuth 2.0 protocol spec.

<i>scope</i>	Equivalent to the profile that is described in the OAuth 2.0 protocol spec.
<i>state</i>	Equivalent to the profile that is described in the OAuth 2.0 protocol spec.
<i>token_type</i>	Equivalent to the profile that is described in the OAuth 2.0 protocol spec.

© 2011 Microsoft Corporation. All rights reserved.

Messenger share button

[This documentation is preliminary and is subject to change.]

You can enable users with a Microsoft account to share info with their friends about selected content on your website. To make this possible, you add specific HTML code to your website next to the content that you want users to be able to share. This code displays a Live Connect *sharing badge*, which consists of a clickable image file and a summary of the content.

When a user clicks a sharing badge, Live Connect prompts users to sign in if they are not already signed in. Live Connect then enables the signed-in user to annotate the content to be shared and then update his or her status message. The user's friends can then click on any of the links in the shared content within the user's activities list and any that are visible in the user's status message.

To add a sharing badge to your website, add the following HTML code next to the content that you want users to be able to share.

```
<a href="http://profile.live.com/badge?url=URL">
 sharing badge"
/>
</a>
```

In the preceding HTML code example, replace `URL` with the URL of the content to be shared, for example, `http://www.contoso.com/LatestNews.htm`. Note that any special characters in the URL, such as the colon (":") and the slash ("/"), must be escaped or otherwise encoded, for example, `http%3A%2F%2Fwww.contoso.com%2Fnewsfeed.htm`.

Note For URLs to articles and videos, you can provide [optional parameters](#).

Also, in the preceding code, replace `SHARING_BADGE_IMAGE_URL` with the URL of one of the available sharing badge images, depending on your preferences for image size and background color.

- <http://js.live.net/static/img/SharingBadge16x16White.png>
- <http://js.live.net/static/img/SharingBadge22x22White.png>
- <http://js.live.net/static/img/SharingBadge16x16Orange.png>
- <http://js.live.net/static/img/SharingBadge22x22Orange.png>

Supported locales

[This documentation is preliminary and is subject to change.]

Supported locales enable you to improve the performance of your websites and apps that access WL.js online. When you specify a locale, Live Connect will attempt to serve WL.js from the server that is closest in geographic proximity. For example, if you specify en_GB for English (United Kingdom), Live Connect will automatically redirect the request to a server somewhere in the United Kingdom. Note that this does not apply to Metro style apps, which use a locally stored copy of WL.js.

The following table lists the set of culture names and corresponding locales that are supported by the Live Connect APIs.

Culture ID	Culture Name
ar-SA	Arabic
bg-BG	Bulgarian
ca-ES	Catalan
cs-CZ	Czech
da-DK	Danish
de-DE	German
el-GR	Greek
es-ES	Spanish
et-EE	Estonian
eu-ES	Basque
fi-FI	Finnish
fr-FR	French
gu-IN	Gujarati
he-IL	Hebrew
hi-IN	Hindi

hr-HR	Croatian
hu-HU	Hungarian
id-ID	Indonesian
it-IT	Italian
ja-JP	Japanese
kn-IN	Kannada
ko-KR	Korean
lt-LT	Lithuanian
lv-LV	Latvian
ml-IN	Malayalam
mr-IN	Marathi
ms-MY	Malay (Malaysia)
nb-NO	Norwegian (Bokmål)
nl-NL	Dutch
pl-PL	Polish
pt-BR	Portuguese (Brazil)
pt-PT	Portuguese (Portugal)
ro-RO	Romanian
ru-RU	Russian
sk-SK	Slovak
sl-SI	Slovenian
sr-Cyrl-CS	Serbian (Cyrillic)

sr-Latn-CS	Serbian (Latin)
sv-SE	Swedish
ta-IN	Tamil
te-IN	Telugu
th-TH	Thai
tr-TR	Turkish
uk-UA	Ukrainian
vi-VN	Vietnamese
zh-CN	Chinese - Simplified
zh-TW	Chinese - Traditional

© 2011 Microsoft Corporation. All rights reserved.