

Inside Windows 7

Introducing Libraries

By Yochay Kiriathy and Alon Fliess

This article is based on a prerelease version of Windows 7. Details are subject to change.

This is the first article in a series of articles about Windows 7. The series focuses on new user experiences that developers can tap into to make their application shine on Windows 7. This article, part one, focuses on a new user profile storage concept in Windows 7, called Libraries. Download Windows 7 Release Candidate now to help you get the most out of this article.

What Libraries Mean for Users in Windows 7

Before we start diving into the Windows 7 Libraries API, we need to better understand why Libraries were introduced in Windows 7 and how they improve the experiences of users to manage their content. To better understand the concept of Libraries in Windows 7, we need to look back in time. Earlier versions of Windows, like Windows Vista and Windows XP, included sets of special folders for storing users content, like My Documents. Windows XP used the Constant Special Item ID List (CSIDL), which is a list of values that provide a unique system-independent way to identify special folders used frequently by applications, but which may not have the same name or location on any given system.

In the Windows Vista timeframe, CSIDL evolved into a new storage system called Known Folder IDs. As of Windows Vista, CSIDL special folders are referenced by a set of GUID values. It is important to note that the CSIDL system is still supported both in Windows Vista and Windows 7 for backwards compatibility. In the Known Folder IDs list you can find the FOLDERID_Documents folder. This folder represents both the My Documents folder as part of the users storage profile and the FOLDERID_Fonts as part of the system special folders. In Windows 7, the list of Known Folder IDs expanded to support Windows 7 Libraries. You can find there, among other things, GUIDs like the FOLDERID_DocumentsLibrary that represents the Documents Library. In Windows Vista, these special folders were automatically indexed to allow users to perform faster, more efficient, search operations on their content. However, we found that many users store files in a variety of places on their PC, not only in the "special" user profile folders, but also in various folders like c:\My Temp Folder, d:\Birthday2008\pictures, or even in remote storage. Storing files outside the users profile storage space affects the indexing, which hinders the search experience of users. We often find ourselves looking for a particular file that we worked on just a few days ago, forgetting where we saved it, and then discover that search is failing to find it because the file was never indexed.

With Windows 7, the concept of Libraries tries to address the problem of users storing content all over their PC by allowing them to have full control over their Documents Library folder structure. In other words, in Windows 7 users can define which folders to include in the Documents Library. Actually, this is true for any Library in Windows 7. We can say that Libraries are user-defined collections of folders that are logical representations of user content. By including folders in Libraries, the user is telling Windows where his or her important data is located. The system will index these folders, enabling faster searching and much richer viewing arrangement capabilities in Windows Explorer based on the file properties and metadata. **Figure 1** displays the integration of several folders into a single library view and shows the rich search and pivots of Windows Explorer in Windows 7.

Inside Windows 7

Introducing Libraries

By Yochay Kiriaty and Alon Fliess

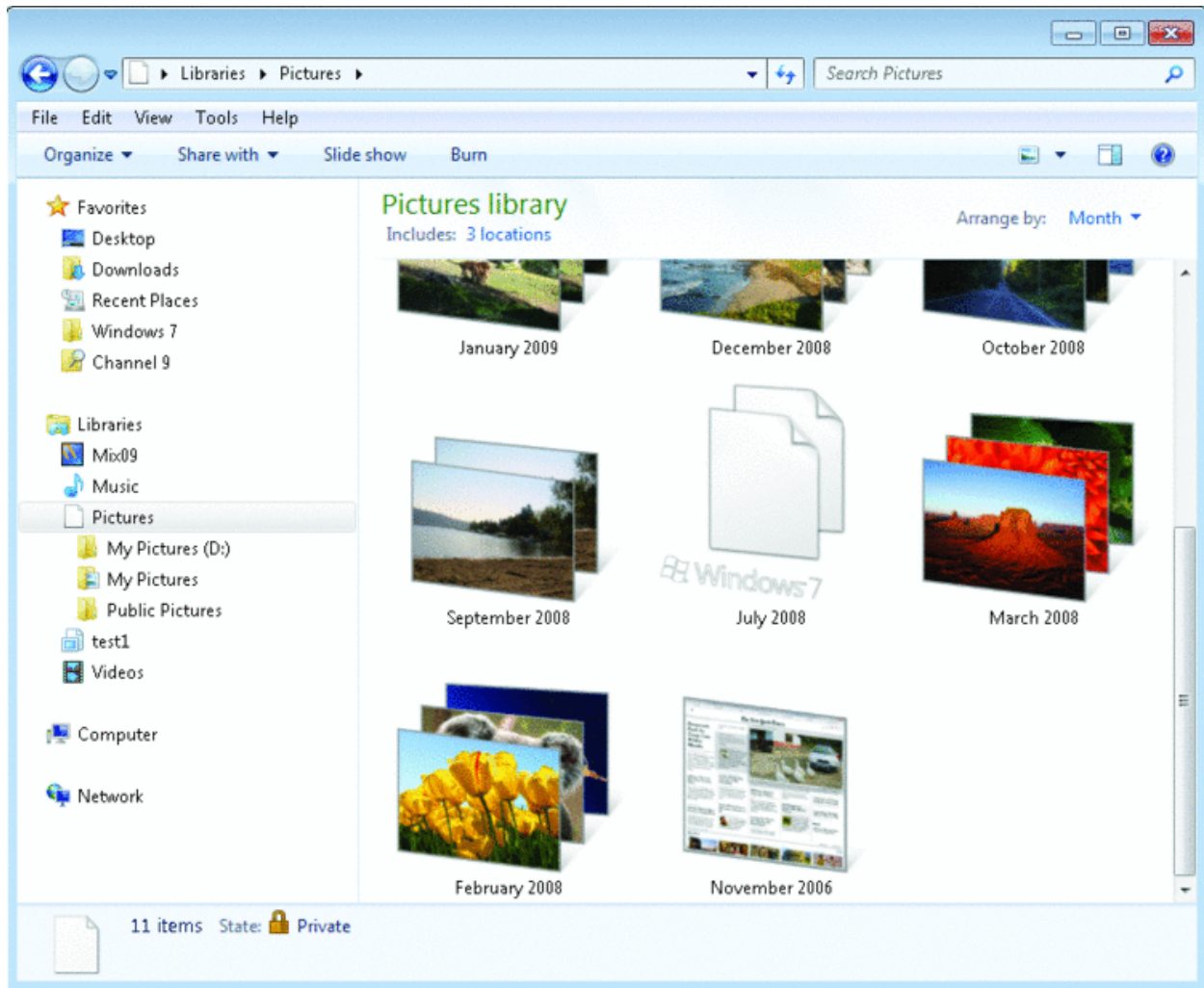


Figure 1
Integration of Several Folders into a Single Library View

Libraries are an integral part of the Windows Shell. This integration is very important because it enables users to browse their files in the same way as they would in a folder, which means there is no new behavior that users need to learn. Clicking on the Documents Library simply shows all your documents in one particular arrangement. Moreover, due to the fact that Libraries are integrated into the Windows Shell, users can perform search operations and filter results by using properties and metadata, such as the date the picture was taken, the genre of the song, and popularity of the item. In other words, by using Libraries users get to enjoy file storage that is both flexible and indexed.

Library-Aware Applications

We want to note that while most applications should work fine with Windows 7 Libraries, there is an opportunity to tap into the Windows 7 Libraries environment to provide richer user experiences. One may wonder what would happen if an application does not support Libraries. Let's imagine an application that, as part of its functionality, needs to save a file to disk. The application prompts the user with the option to select a location for the file to be saved. The user may pick the Documents Library as his or her save "location," since this is where the user usually goes when he or she needs to work on a document.

Inside Windows 7

Introducing Libraries

By Yochay Kiriaty and Alon Fliess

However, if the application doesn't recognize that the Documents Library is not a regular folder, the application will try to save the file directly to the Documents Library as if it is just another folder on disk. This represents a problem, since by now, as we already know, Libraries are a non-file system location and therefore can't be treated as regular folders. The application needs to become aware of the fact that they are dealing with Libraries. Luckily, Windows 7 includes an updated Shell API, Libraries API, and updated Common File Dialog that allow developers to handle Libraries properly.

A Library-aware application should include mechanisms for handling situations where users inadvertently try to pick Libraries as if they are folders for either saving files to a Library or loading the contents of a Library. Furthermore, most applications allow users to interact with the file system as part of the application experience. An application should provide users with the same familiar entry points and UI offered by the Windows 7 Libraries. By including folders in a Library, users designate where their important data is stored, telling us this is the content they care about. Applications should promote these locations by supporting Libraries in their application.

For developers, there are several integration points with Windows 7 that can help applications become Library-aware. Developers should review the following three integration points and select according to their needs.

1. The most basic integration point is to simply use the Common File Dialog (CFD) for picking files and folders, and saving directly to a Library.
2. The second integration point offers applications the option to shine on Windows 7 by enabling applications to select and consume Libraries content.
3. The last integration point provides the more advanced integration option of supporting the full library programming model.

Using the Common File Dialog

The good news is that Windows 7 Libraries are a first class citizen in the CFD, allowing users to browse through and search libraries. They can even pick a specific library as a save location that is not just one of the folders from within the library, but rather the library itself! Because Windows 7 Libraries are storage-backed, users can save and copy files to any folder they have permissions for that are included in a Library. Every Library has a default save location where files go when users copy and save files directly to the Library. By default, this location will be the known folder that is included in a default Library, or the first folder added to a custom Library.

But (there is always this caveat!), it is highly recommended to use the new CFD interface that was introduced with Windows Vista and not the legacy version of the CFD. It is very important to note that the APIs for using the legacy CFD have not changed since Windows Vista and Windows XP, for application compatibility reasons. However, the legacy version of the CFD doesn't directly support Libraries or the full new user experience offered in Windows 7. **Figure 2** displays the legacy and the new CFD side by side.

Inside Windows 7 Introducing Libraries By Yochay Kiriaty and Alon Fliess

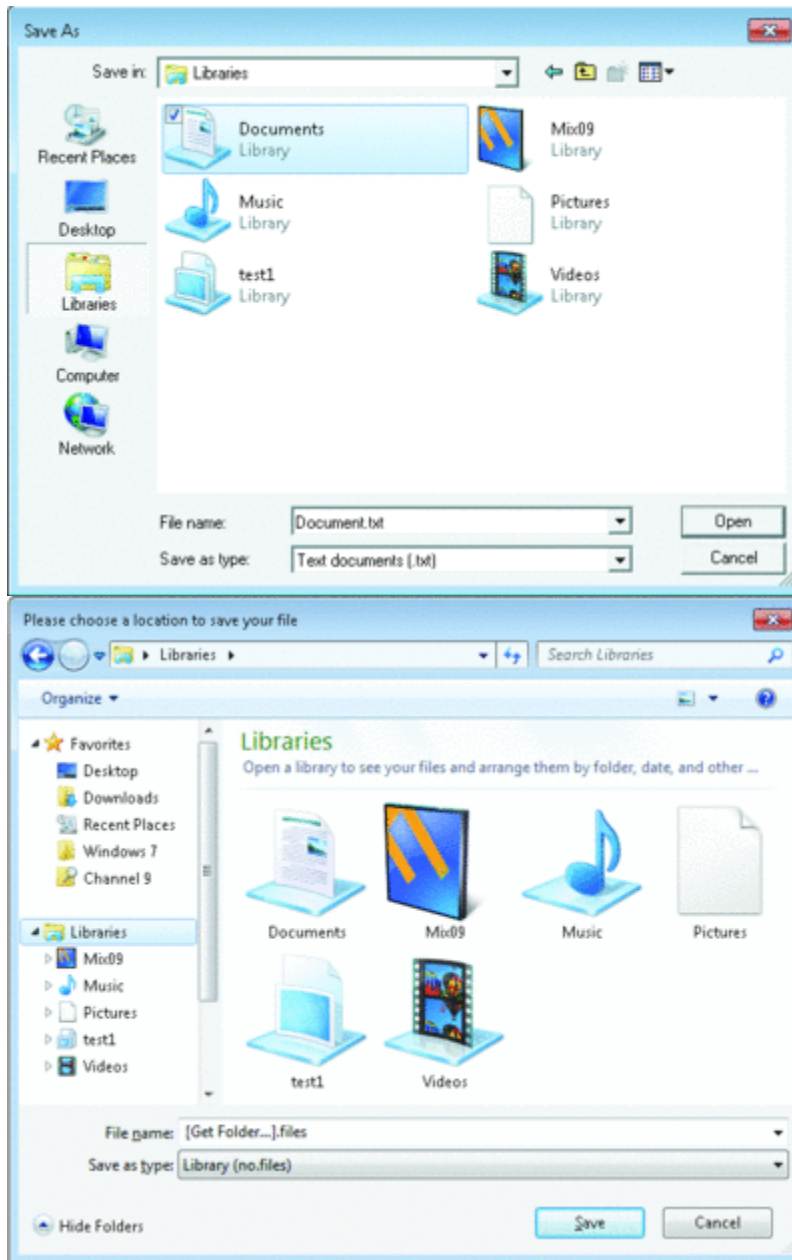


Figure 2
Old and New View of the New Common File Dialog

In the legacy CFD, even if Libraries are presented in the right navigation pane, they require an extra click to save into one of the included folders, rather than just the Library itself. Users can neither search directly from within the CFD or experience the rich preview handlers, nor select multiple files across folders, because the legacy CFD doesn't support returning multiple files from different folder locations. In contrast, this is a scenario that a Windows 7 Library supports.

It is important to use the proper APIs to show the correct version of the CFD. When it comes to showing a CFD using .NET, developers can either use the System.Windows.Forms.FileDialog or the Microsoft.Win32.FileDialog namespace. Since the latter uses the legacy version of the CFD, .NET developers

Inside Windows 7

Introducing Libraries

By Yochay Kiriathy and Alon Fliess

should always use the WinForms namespace to show the new CFD. **Figure 3** shows a code snippet that prompts the user to choose a save location by showing the common save file dialog, enabling the user to select folders or libraries.

Figure 3 SaveFileDialog File

```
System.Windows.Forms.SaveFileDialog _fd = new System.Windows.Forms. SaveFileDialo
g();
_fd.Title = "Please choose a location to save your file";
_fd.FileName = "[Get Folder...]";
_fd.Filter = "Library|no.files";
if (_fd.ShowDialog() == System.Windows.Forms.DialogResult.OK)
{
    string dir_path = System.IO.Path.GetDirectoryName(_fd.FileName);
    if (dir_path != null && dir_path.Length > 0)
    {
        lblResult.Content = dir_path;
    }
}
```

Native code developers should use the new family of IFileDialog native APIs: IFileDialog, IFileOpenDialog, IFileSaveDialog, IFileDialogCustomize, IFileDialogEvents, IFileDialogControlEvents. These replace the legacy APIs from earlier Windows versions: GetOpenFileName, GetSaveFileName.

Figure 4 shows how to use the new family of IFileDialog native APIs to prompt the user with a save dialog to save to a folder / library.

Figure 4 IFileDialog Native API

```
*ppsi = NULL;
IFileSaveDialog *pfod;
hr = CoCreateInstance(
CLSID_FileSaveDialog,
NULL,
CLSCTX_INPROC,
IID_PPV_ARGS(&pfod));
if (SUCCEEDED(hr))
{
    hr = pfod->SetOptions(FOS_PICKFOLDERS);
    if (SUCCEEDED(hr))
    {
        hr = pfod->Show(hWndParent);
        if (SUCCEEDED(hr))
        {
            hr = pfod->GetResult(ppsi);
        }
    }
    pfod->Release();
}
```

The Shell native APIs are COM-based. Before using any COM object we need to initialize the COM object by calling CoCreateInstance. After initializing the *pfod IFileSaveDialog variable, we set the dialog options to pick folders by passing the FOS_PICKFOLDERS flag to the IFileOpenDialog.SetOptions(). This code tells the Open dialog to enable the user to select folders rather than files and allows the user to choose a Library as a

Inside Windows 7

Introducing Libraries

By Yochay Kiriathy and Alon Fliess

save location. In the case of choosing a Library, the CFD will then return the default save location folder that is associated with the chosen Library.

Figure 3 and **Figure 4** are very simple and don't introduce any new code. However, it is important to promote consistency among applications running on Windows 7 and in support of Windows 7 Libraries. The CFD is designed to provide a consistent experience with the new Windows Explorer, including Libraries. All improvements in Windows 7 Explorer have been carried over to the CFD. In most cases, it is the ideal way for users to browse and interact with Libraries from inside the application.

Select and Consume Libraries Like Folders

Let's imagine a case of a slideshow application that presents pictures to the user. By using Libraries, the user is essentially telling the system that his or her important pictures are stored in the Picture Library. The application can simply point directly to the Pictures Library and show the entire collection of pictures to the user. Furthermore, from the point of view of the developer, using the Library system can eliminate the need to maintain a separate configuration file or database of pictures, for example, since developers can rely on the Library System. Before we dive into the Shell Libraries programming APIs, we need to understand a few concepts regarding the Shell Programming Model.

Shell Programming Model

A Shell item (IShellItem), commonly referred to as an item, is the currency of the Shell UI and its programming model. Items are individual, self-contained content sources. For example, quite a few of the interface methods used for controlling CFDs use Shell items to refer to folders instead of file system paths. This is important because the CFD can communicate information about both file system folders and other virtual folders that you find in the Shell, such as the Control Panel or the Computer Folder. Other important Shell COM interfaces are:

IShellLink interface represents a link, usually to a file, folder, or an executable

IShellFolder interface represents shell folder objects from the shell namespace. Using IShellFolder you can traverse through the contents of a folder, retrieve the display name of an item in a folder, parse a display name relative to the folder, and obtain an item ID list

Windows 7 introduces a new Shell API, called IShellLibrary, that you can query from IShellItem in order to work and support Libraries in Windows 7.

Now that we have defined the different components of the Shell Programming Model, we can see how Libraries fit into this model. Since Libraries are not file system locations, you cannot use file system-specific APIs like FindFirstFile. Instead, you have two main options to consume the contents of a Library.

Using the Shell Programming Model

You can use the IShellItem and IShellFolder interfaces and a number of helper functions to enumerate the contents of Libraries, just as if they were regular folders. This means that applications can consume the content of a Library without using the new Libraries API and with very little change to their existing codebase. Figure 5 shows how to use the IShellFolder interface to enumerate the entire contents of the Picture Library.

Figure 5 IShellFolder Interface

```
IShellItem *psi;  
HRESULT hr = SHGetKnownFolderItem(FOLDERID_PicturesLibrary, KF_FLAG_CREATE, NULL,  
    IID_PPV_ARGS(&psi));  
if(SUCCEEDED(hr))  
{
```

Inside Windows 7

Introducing Libraries

By Yochay Kiriathy and Alon Fliess

```
IShellFolder *psf;  
hr = psi->BindToHandler(NULL, BHID_SFObject | , IID_PPV_ARGS(&psf));  
if(SUCCEEDED(hr))  
{  
    IEnumIDList *penumIDList;  
    psf->EnumObjects(NULL, SHCONTF_FOLDERS | SHCONTF_NONFOLDERS , IID_PPV_ARGS(&penumIDList));  
    //use penumIDList to enumerate the content of the folder  
}  
}
```

Here you can see that by using the helper function SHGetKnownFolderItem we can retrieve the correct location of the Library by passing the FOLDERID_PicturesLibrary. This is a GUID representing the known folder, which in our case is the Picture Library. A successful call will fill the IShellItem *psi interface with the correct information about the Library, represented as a Shell item. The rest of the code is standard Shell programming, where we use the BindToHandler to bind the previously obtained Shell item to a Shell folder (casting). Next, we enumerate through the different items in the Shell folder, which in the case of a Library can be either files or folders. The SHGetKnownFolderItem is a Shell Helper function and is part of a larger group of helper functions that can be found in the shobj.h header file in the Windows 7 RC SDK. Note the SHCONTF_FOLDERS | SHCONTF_NONFOLDERS flags that we are passing. This is telling the shell folder that we want to return all of the files and folders in a library. We could pass SHCONTF_NAVIGATION_ENUM to get the library locations instead of the library contents.

Using the New IShellLibrary API

You can achieve the same functionality shown in **Figure 5** by using the new Windows 7 IShellLibrary API, as shown in **Figure 6**.

Figure 6 IShellLibrary API

```
IShellLibrary *pslLibrary;  
HRESULT hr = SHLoadLibraryFromKnownFolder(FOLDERID_PicturesLibrary, STGM_READ, IID_PPV_ARGS(&pslLibrary));  
if(SUCCEEDED(hr))  
{  
    IShellItemArray *psiaFolders;  
    hr = pslLibrary->GetFolders(LFF_STORAGEITEMS, IID_PPV_ARGS(&psiaFolders));  
    IEnumShellItems *penumShellItems;  
    psiaFolders->EnumItems(&penumShellItems);  
    //work with penumShellItem to enumerate the items in the library  
}
```

Here you can see that we use another helper function, SHLoadLibraryFromKnownFolder, to create the IShellLibrary object. From this object we can call the GetFolders method to return an IShellItemArray. This return is used later to obtain an enumerator to traverse through the entire contents of the Library.

In the last example, we used the helper function, SHLoadLibraryFromKnownFolder. As mentioned above, this helper function, and others related to Windows 7 Libraries, can be found in the shobj.h header file from the Windows 7 RC SDK. Here is a list of the important Library helper functions:

SHAddFolderPathToLibrary (adds a folder to a library)

SHCreateLibrary (creates an IShellLibrary object)

Inside Windows 7

Introducing Libraries

By Yochay Kiriathy and Alon Fliess

SHLoadLibraryFromItem (creates and loads an IShellLibrary object from a specified library definition file)

SHLoadLibraryFromKnownFolder (creates and loads an IShellLibrary object for a specified KNOWNFOLDERID)

SHLoadLibraryFromParsingName (creates and loads an IShellLibrary object for a specified path)

SHRemoveFolderPathFromLibrary (removes a folder from a library)

SHResolveFolderPathInLibrary (attempts to resolve the target location of a library folder that has been moved or renamed)

SHSaveLibraryInFolderPath (saves an IShellLibrary object to disk)

Let's review the following code (**Figure 7**) that uses a few of these helper functions to create a new Library, associates a folder with that Library, and "saves" the Library in the Libraries Folder.

Figure 7 IShellLibrary Using the SHCreateLibrary Helper Function

```
IShellLibrary *pIShellLibrary;
HRESULT hr = SHCreateLibrary(IID_PPV_ARGS(&pIShellLibrary));
if (SUCCEEDED(hr))
{
    IShellItem *pIShellItem;
    SHAddFolderPathToLibrary(pIShellLibrary, L"C:\\Users\\Public\\Documents");
    hr = pIShellLibrary->SaveInKnownFolder(FOLDERID_Libraries, L"My New Library",
LSF_MAKEUNIQUENAME,
    &pIShellItem);
    pIShellItem->Release();
    pIShellLibrary->Release();
}
```

Here you can see that we are creating a new IShellLibrary object using the SHCreateLibrary helper function. Next we add the public Documents folder to the Library object. Then we save the new Library in the Libraries folder, with the rest of the Libraries, and give it the name My New Library.

In **Figure 7**, we used the SaveInKnownFolder method of the IShellLibrary interface to save the new Library we just created. Most of the methods in the IShellLibrary interface are self-explanatory. However, let's review some that require our attention:

The Commit method commits library changes to an existing library file. This means that whenever you programmatically change a library, you need to call the commit method in order to save the changes.

The SetIcon method takes a resource DLL name and Icon index to set the library icon.

The SetFolderType receives a GUID of known folder-type templates as an input param.

Inside Windows 7

Introducing Libraries

By Yochay Kiriathy and Alon Fliess

This GUID defines the type of the library, which can be one of the following: Generic, Pictures, Music, Video, and Documents. Setting the folder-type template changes the Windows Explorer view of the library and enables search and pivotal view options that are specific to the library type.

Libraries under the Hood

As shown in **Figure 7**, the code creates a new library file in the Libraries folder. A library in Windows 7 is stored as an XML definition file that has a file extension of .library-ms. The file name is the actual name of the library. For example, the Documents Library is represented by an XML file called Documents.library-ms. Library descriptions are saved on disk in the %appdata%\Microsoft\Windows\Libraries folder (also known as FOLDERID_Libraries).

Let's dig into the Documents Library definition file schema. The XML structure is pretty self-explanatory, but let's explain a few of its elements. As shown in **Figure 8**, at the top of the file we can find the Library "header" information:

Figure 8 Documents Library Definition File Schema

```
<libraryDescription xmlns="http://schemas.microsoft.com/windows/2009/ library">
  <name>@shell32.dll,-34575</name>
  <ownerSID>S-1-5-21-2127521184-1604012920-1887927527-4897363</ownerSID>
  <version>4</version>
  <isLibraryPinned>true</isLibraryPinned>
  <iconReference>imageres.dll,-1002</iconReference>
  <templateInfo>
    <folderType>{7D49D726-3C21-4F05-99AA-FDC2C9474656}</folderType>
  </templateInfo>
```

The root XML element is libraryDescription, which contains child elements that define the library, as follows:

<ownerSID> defines the Security ID of the user who created this library to isolate Libraries and protect user data from other users.

<isLibraryPinned> is a Boolean element that defines if the library is pinned to the left navigation pane in Windows Explorer and NOT to the Taskbar.

<version> defines the content version of this library, which reflects the number of times the library definition file has been changed.

<templateInfo> is an optional container element that lets the author specify the folder type (Documents, Pictures, Video) for controlling the arrangements of views in Windows Explorer.

<iconReference> defines an icon resource using the standard Windows Shell resource style. For example:

<iconReference> C:\Windows\system32\imageres.dll,-65 **</iconReference>**.

This icon presents the library in Windows Explorer.

Inside Windows 7

Introducing Libraries

By Yochay Kiriathy and Alon Fliess

Users can't change the default libraries icon via Windows Explorer, nor can they assign the icon to a new customized user-defined library. However, this can be done programmatically using the API, which we will cover in future parts of this series.

A key part of the XML is the list of locations that the Library represents, as shown in the following code:

```
<searchConnectorDescriptionList>
  <searchConnectorDescription publisher="Microsoft" product="Windows">
    <description>@shell32.dll,-34577</description>
    <isDefaultSaveLocation>true</isDefaultSaveLocation>
    <isSupported>true</isSupported>
    <simpleLocation>
      <url>knownfolder:{FDD39AD0-238F-46AF-ADB4-6C85480369C7}</url>
      <serialized>MBAAAE... </serialized>
    </simpleLocation>
  </searchConnectorDescription>
</searchConnectorDescriptionList>
```

`<searchConnectorDescriptionList>` contains one or more search connectors that map to physical locations included in the library.

`<searchConnectorDescription>` contains a `simpleLocation` element that describes one location included in the library.

`<url>` defines a URL for this location. For human-readability only, this cannot be used by developers since it is likely to be out of date.

`<serialized>` is the actual representation of a location in a library that is a serialized ShellLink object.

One final note: Applications should never attempt to access or edit Library description files. Instead, applications should always use the Shell programming model or the IShellLibrary API to consume and manipulate the Library content.

Support the Full Library Model

A user can add, remove, and reorder locations, as well as change the default save location through the Library Management UI. This is shown in **Figure 9**, which is available directly from Windows Explorer.

Inside Windows 7

Introducing Libraries

By Yochay Kiriaty and Alon Fliess

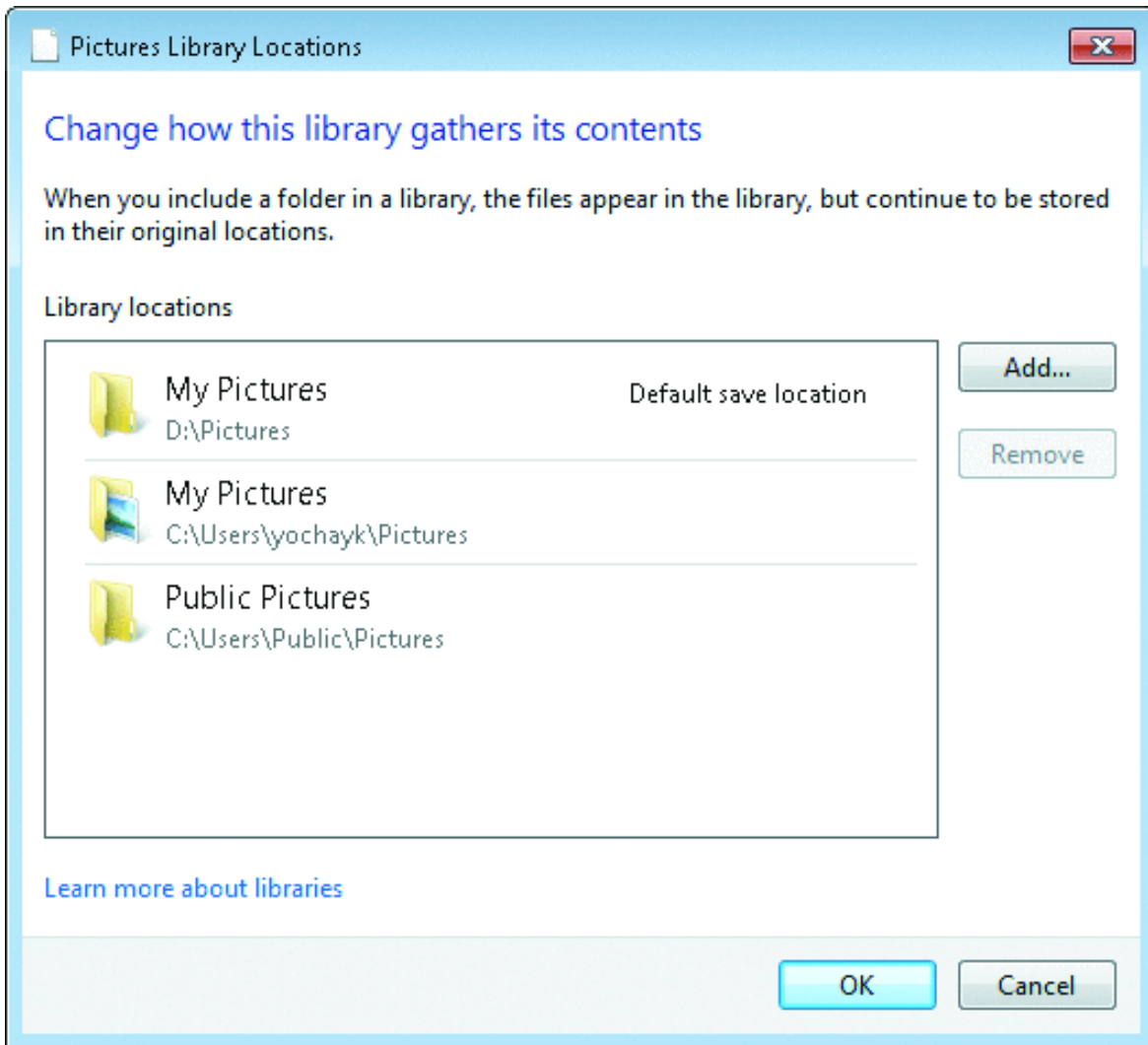


Figure 9
Library Management UI

In addition, library locations may be modified through the IShellLibrary interface. Any change to a library structure is reflected in the library definition file and persists directly to the underlying .library-ms file. You can be informed of such changes by monitoring any changes to the .library-ms file. Assume your application is relying on the content of a library, or your application is currently displaying the content of a given library. You would like to be notified once the content of the library changes. Applications that wish to be notified about changes to the library definition file can do so by using the native SHChangeNotifyRegister Shell helper function or by using the Managed FileSystemWatcher found in the System.IO namespace. Using these interfaces is out of the scope of this article since these are not new APIs and are well documented.

Another option to consider is the case in which your application needs to manage folders for users, like adding a new folder of pictures into the picture repository. If you are using the Pictures Library, you can use the library management dialog in your application to show the same dialog that Windows 7 offers users to manage their libraries. By doing so, you keep a consistent look and behavior that the user grows to appreciate. If you choose to use the library management dialog interface, changes to the Library will be made as if you were modifying the Library contents directly within Windows Explorer. This dialog will not

Inside Windows 7

Introducing Libraries

By Yochay Kiriaty and Alon Fliess

return any information to your application. In case you are showing the contents of a given library, you will need to register for notification to receive updates, as explained above.

Summary

In this article, you were introduced to the concept and programming model of Libraries in Windows 7. You reviewed the important role that libraries play as part of the Windows 7 user experience. Then you took a deep-dive into Libraries, understanding what Libraries are and exploring their supporting underlying architecture. Then you saw the different opportunities that developers have, to make their applications Library-aware. Finally, you went through a short tour of the different available programming models and APIs.