

From the November 2001 issue of MSDN Magazine.



New Graphical Interface: Enhance Your Programs with New Windows XP Shell Features

Dino Esposito

This article assumes you're familiar with GDI

Level of Difficulty 1 2 3

SUMMARY The Windows XP shell introduces many new features that both users and developers are sure to welcome. The interface supports a number of styles that will be new to users, and it also supports customization of those styles through a new concept called themes. There are more shell registry settings available to the user and developer, a facility for customizing infotips, and infotip shell extensions. In addition, folder views can be customized.

This article covers these shell changes and includes a discussion of a number of other Windows XP additions. These include fast user switching, which lets users log on and off quickly, and AutoPlay support for a variety of devices and file types not previously supported.

Like it or not, the new Windows® XP user interface (formerly codenamed "Luna") is something that most people have strong feelings about when they first see it. I confess that when I first completed the Windows XP installation and I was confronted with the new Windows XP visual style, I was tempted to drop it and return to the more classic and familiar Windows 2000 interface (which is an option with Windows XP). I wanted my old Windows back!

The Windows XP shell introduces a rationalization of the user interface services and features found in previous versions of Windows. As a result, a few things you could do with Windows 2000 are either unsupported under Windows XP or are supported in a more restrictive way. To make up for this, new common controls and, above all, new listview capabilities, make the Windows shell easier to use with richer programming support.

In this article, I'll delve into the Windows XP shell to cover shell settings, shell extensions, new registry entries, and folder customization. Finally, I'll summarize the key things you need to know about supporting Windows XP visual styles in your own apps through the new family of common controls. This article is based on the first Release Candidate of Windows XP, so it's remotely possible that some of this information might change when the product ships. However, if you follow along you'll find the kind of information you need to get ready for Windows XP now.

What's New in the Windows XP Shell?

Windows XP does not support any new shell extensions that aren't also supported in Windows 2000, with one exception I'll cover shortly. However, folder customization through hypertext templates (.htt files) has been deemphasized in Windows XP and important restrictions have been applied.

There are places in the Windows XP shell where the user interface is drawn in a completely new way. The underlying technology, though, is not documented or exposed at present.

There are now more shell registry settings, letting you better personalize more aspects of the shell

shell extension. Additional shell enhancements include common controls, fast user switching (the ability to have the system work with multiple users simultaneously connected), and Autoplay. The one new type of shell extension Windows XP *does* have that allows you to register new handlers for music, video, and pictures delivered through CD, DVD and other media. (See the "Automatically Detect and React to New Devices on a System" article in this issue.)

The contents of my Windows 2000 article "[Enhance Your User's Experience with New Infotip and Icon Overlay Shell Extensions](#)" (*MSDN Magazine*, March 2000) is still largely valid. However, I'm going to revisit some of the information in that article in light of Windows XP.

After playing with Windows XP for a couple of days, I got the distinct impression that the system tends to control what the user can and cannot do much better than earlier versions of the Windows operating system did. I've written at least three articles for *MSDN® Magazine* about the Windows 2000 shell and registry. In addition to the aforementioned piece about shell extensions, I covered advanced folder customization in "Extending Explorer Views by Customizing HyperText Template Files" in the June 2000 issue and the registry in "Latest Features and APIs Provide the Power to Customize and Extend Your Apps" that appeared in the November 2000 issue. All these articles (whose URLs are listed in this article summary) illustrate hidden gems in the Windows 2000 shell architecture and shed light on poorly documented, yet important topics and features that satisfy diverse needs and make developers more productive. For example, how do you change a folder's icon? Based on some early MSDN documentation, if you create a file called `desktop.ini` in a folder, and add the following content

```
[.ShellClassInfo]
IconFile=shell32.dll
IconIndex=41
```

then setting the readonly attribute for the folder forces Explorer to pay special attention to the `desktop.ini` file. This makes Explorer read the `.ini` file and display the specified icon when folders are listed. While this is not a particularly difficult task for developers and power users, it required knowledge of implementation details and familiarity with the system.

In Windows XP, any user who wants to mark a particular folder with a custom icon need only right-click on the folder, select the Customize tab and follow the instructions shown in **Figure 1**. What happens under the covers is similar to the process I just described, but the details are hidden from the user.

 Figure 1 Choose Icon

Figure 1 Choose Icon

Another interesting feature that was once the exclusive domain of power users, but which is now available to any Windows XP user, is the ability to associate actions with classes of files. For example, suppose you want to add the Register and Unregister items on the context menu of DLL files to register and unregister the file as a COM component. Before Windows XP, this required a Windows Script Host (WSH) script to directly manipulate the registry or some compiled code using the registry API. While this remains the only way to accomplish it programmatically, users can now access a sequence of dialogs to associate commands with classes of files. Just click on the Folder Options item from the Explorer Tools menu, open the File Types tab, select or add the particular file type, and click the Advanced button. What happens next is shown in **Figure 2**.

 Figure 2 Advanced Options

Figure 2 Advanced Options

Customizing the layout of a Windows folder view is yet another user interface feature that's changed significantly in Windows XP. No longer will a wizard guide you through various steps. Instead, there is a simple dropdown list in the Customize tab of the folder's properties. From there, you can

In general, Windows XP comes with a significant number of user interface improvements, especially for users and power users. In prior versions of Windows, some of these improvements were only useful to programmers. In Windows XP, many more people can take advantage of these UI improvements. From the programmer's perspective things have been left more or less unchanged, but the tasks are now more thoroughly documented in the latest Platform SDK.

Shell Registry Settings

Let's start this tour of the Windows XP shell by looking at some little-known yet useful registry tweaks that can help you to better configure an application and fuse it to the system shell.

A large number of the shell features are governed by settings stored in the system registry. By changing values and creating entries, you can tune up the overall behavior of the shell and its constituent elements. For example, you can control the way in which a file is processed (AlwaysShowExt, NoOpen, NoOpenWith, EditFlags settings), the way a system object presents itself through Explorer, and the way Infotips are formatted. A majority of the settings mentioned here also work with Windows 2000.

The AlwaysShowExt setting serves the purpose of overriding one of the folder options that hides the extension of known file types. A known file type is any file class whose extension is registered with the system, along with handlers for actions like open, print, and edit. Technically speaking, you know that a file class is properly registered with the operating system when you find the following registry key exists:

```
HKEY_CLASSES_ROOT
\.ext
```

Here, `.ext` represents the file extension. If the AlwaysShowExt value is present in the registry, the file extension is always displayed in Explorer, regardless of the setting of the "Hide extensions for known file types" folder option. [Figure 3](#) and **Figure 4** show the various settings for typical DLL files.

Figure 4 Folder Options

Figure 4 Folder Options

Another rather interesting attribute is NoOpen. Through not frequently used, it helps the system notify users that they're about to do something dangerous. If your program creates files that users should not modify, you can register the corresponding file type with a NoOpen attribute under the file type identifier key. A file type identifier key takes the form of extfile, so the identifier for DLLs would be dllfile.

If the user attempts to open any file from a class that has been marked with the NoOpen attribute, then the system displays a warning message. If you assign some text as the value of the NoOpen attribute, the system displays that text in the message box. However, bear in mind that despite the attribute's name, signing your files this way does not prevent users from opening the files; they only get a warning. In addition, notice that NoOpen does not work if any command action has been specified for the class. For example, by default a DLL file has no custom command registered. (Except on my system I added a custom command through the dialogs shown in **Figure 2**.) In [Figure 3](#), you also see that DLL files are marked as NoOpen. Try to double-click on a DLL file under these conditions and you see a message that's similar to those in **Figure 5**. The top window shows the default message, whereas the bottom window displays a custom message that is the text of the NoOpen registry entry.


Figure 5 Default and Custom Messages

Figure 5 Default and Custom Messages

Assigning attributes to a file class allows you to control some aspects of its behavior. It also allows

verbs, through the File Types tab of the Folder Options dialog. The attributes are defined as binary flags and are stored in the EditFlags entry. EditFlags must be a REG_BINARY entry. For DLL files, its registry path evaluates to

```
HKEY_CLASSES_ROOT
\dllfile
```

To assign the EditFlags entry of a file class, take the sum of any combination of the attributes listed in [Figure 6](#). For example, setting EditFlags to 0x00100100 combines the ability to preserve the description of the file class with the restriction that files of that type cannot be added to the Recent Documents folder after being opened.

Reading about this feature validated all of my conjectures and hunches a few years ago when I realized that not all files that I opened were listed in the Recent Documents menu. This is the perfect example of system features that work with Windows XP and are documented for Windows XP, but are also present in earlier versions of Windows.

Customized Infotips

Each file type can be associated with an infotip by creating a registry entry called Infotip under the following path:

```
HKEY_CLASSES_ROOT
\extfile
```

If you register infotips in this way, you cannot apply them on a per-file basis. The string that you specify will be displayed for every file of that file type.

To customize infotips, you can write a shell extension, as I'll show you shortly. As an alternative, Windows XP supports a huge number of property names, each of which points to a special piece of information about files. Some of the supported property names, which also apply to Windows 2000, are listed in [Figure 7](#). The Infotip registry entry is assigned with a semicolon-separated sequence of these properties. Then, when Explorer is called to display the infotip for that file, it retrieves the information specific to that file. For example, if you look at [Figure 3](#) you'll notice that DLLs have the following infotip structure:

```
prop:FileDescription;Company;FileVersion;Create;Size
```

Any infotip text that includes property names must begin with prop: as this is the prefix that tells Explorer to apply a special treatment to the text that follows. Some of the property names in the list are specific to Windows XP, namely FileDescription, Company, and FileVersion. As you may have guessed already, they refer to information that can be stored in the VersionInfo resource of Windows executables. The list of property names includes information that is typical of a certain category of files, but not all. When a certain property name does not have a corresponding value, then it is simply ignored. In [Figure 8](#) you can see the infotips for two different DLLs. Guess what? Only the bottom one has version information.


 Figure 8 Infotips

Figure 8 Infotips

Speaking of DLLs, another particular shell registry setting you may have noticed in [Figure 3](#) is TileInfo. This refers to a new, Windows XP-specific view mode called Tiles. Tiles mode works side by side with other popular views in Windows such as thumbnails, details, and large. The TileInfo registry entry defines the format that displayed information should have when the file name is rendered in tiling mode in a folder view. TileInfo follows the same syntax rules as InfoTip.

Figure 9 shows how the menu looks. Notice that, unlike the thumbnail view which is implemented internally by the system, tiles and groups (another new feature for arranging icons) are new functionalities of the Windows XP listview common control.

AutoPlay

AutoPlay is a feature that detects and properly handles special content such as pictures, music, or video files stored on removable media and devices. AutoPlay should not be confused with AutoRun. AutoRun, introduced with Windows 95, enables removable media like a CD to automatically launch an application when placed in the drive. Autorun is accomplished through the use of a file called autorun.inf, located in the root directory of the CD.

AutoPlay takes in the essence of AutoRun, but applies specifically to certain categories of files and provides a way for you to register handlers for those kinds of content. Put another way, AutoRun makes a CD self-installing, or at least self-running. It does not rely on files or information stored on the machine. AutoPlay addresses a different requirement. When a user connects a specialized peripheral device such as an MP3 player or digital photo reader, AutoPlay allows machine-specific handling of the device. For instance, when you attach a digital camera, you might be asked whether you want to display the camera as a drive, browse the photos, print them, and so on. For more information on Autoplay see the article on page 77 in this issue.

Revamped Shell Extensions

With the exception of IHWEventHandler, which is involved with Autoplay, you will not find any new shell extensions in Windows XP that weren't in Windows 2000. However, the documentation now uses slightly different terminology for a few of the better-known extensions. Context menu shell extensions are often referred to as shortcut menu extensions. The column handler shell extension is frequently called the details shell extension.

I selected three types of shell extensions that you should always use as companion code for your custom documents. They are the infotip shell extension a must-have if you have very structured and complex documents the column handler now the only way you have to display custom information about the selected file in the folder view, and finally the thumbnail viewer. A thumbnail viewer shell extension is a piece of software that provides a graphical preview of the content of a given file. In Windows 2000, the output of such a module was displayed in the standard template upon which the folder view was based. In Windows XP there is no longer the notion of a folder template, at least not one that is used for all the files throughout the system. For performance reasons, the Windows XP UI design team opted for a binary mechanism (a namespace extension) that allows very little customization or "reverse-engineering." I'll have more to say about this by the end of the article. When the folder view turns to the thumbnail mode (see [Figure 10](#)), then all the folder items are displayed in a larger size and, whenever possible, use a dynamically generated thumbnail. To do this, you must write a shell extension. As you may notice in [Figure 10](#), not all the documents of the same type provide a thumbnail. This has nothing to do with the shell extension itself, but relates to the way in which the Office 2000 thumbnail extractor really works. It relies on the preview image dynamically generated and stored in the document itself. This holds true for Word, Excel and PowerPoint® documents. If the document does not include such a preview image it will not be displayed by the thumbnail extractor. You can check the preview image for an Office 2000 document by popping up the Properties dialog. Let's take a quick tour of how to write these shell extensions.

Infotip Shell Extensions

A shell extension is a (relatively) small, simple COM object that implements one or two interfaces. Which interfaces are implemented and how depends on the specific tasks the component you're writing

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts



Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research



With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips



Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.