Exhibit 24

## Summary of the Logo Requirements

[ http://msdn.microsoft.com/en-us/library/aa240996(VS.60).aspx ] | Overview
[ http://msdn.microsoft.com/en-us/library/aa271855(VS.60).aspx ] | How Do I
[ http://msdn.microsoft.com/en-us/library/aa271856(VS.60).aspx ]
There are five basic requirements your application must meet to qualify for the Windows 95 logo, no matter what type of application it is:
*Win32 executable

Your application must be a Win32 executable file using the PE (Portable Executable) format. This requirement is satisfied automatically, because Visual C++ for the Intel platform always produces executable files in the PE format.

If you have a 16 -bit application that you need to port, see Porting 16 -Bit Code to 32 -Bit Windows [ http://msdn.microsoft.com/en-us/library/aa294347(VS.60).aspx ].

- UPI and Shell support

Your application must follow the Windows 95 application setup guidelines, register large and small icons, and use system color and metrics. It's also recommended that your application provide context menus through the right mouse button, use the common dialogs and controls, and follow the user interface guidelines set forth in the Windows Interface Guidelines for Software Design on the MSDN Library CD.

The section Following UPI Recommendations [ http://msdn.microsoft.com/en-us/library/aa279020 (VS.60).aspx ] below discusses how to add this support using MFC.

* Windows NT compatibility

Your application must run successfully on both Windows 95 and Windows NT 3.5 (or greater). If your application uses functionality specific to either operating system, its behavior must degrade gracefully when run on the other operating system (that is, an execution error should not result).

Meeting this requirement depends on your use of API functions specific to Wirdows 95 or Windows NT. For a list of these APIs, see the paper "Diving into the Requirements for the Windows 95 Logo" on the MSDN Library CD.

* Long file name support

Your application must be able to accept and store long file names and display them in its title bar, in dialogs and controls, and so on.

The MFC library supports the use of long file names, so you can pass a long file name to any MFC function taking a file name as a parameter. You should also use the MFC common dialog class CFileDialog when requesting a file name from the user. Make sure that any file name-handling code of your own can handle file names longer than eight characters and file names containing spaces or other special characters. For more information on long file names, see Supporting Long File names on the MSDN Library CD.

* Plug and Play support

This is recommended but not required. Your application should respond appropriately to the
WM_DEVICECHANGE, WM_DISPLAYCHANGE, and WM_POWERBROADCAST messages, which signal changes in peripheral devices, the display resolution, or the system power status, respectively.

The MFC library responds to the WM_DISPLAYCHANGE message and resizes windows and toolbars according to the new system metrics. How an application should respond to the
WM_DEVICECHANGE and WM_POWERBROADCAST messages depends on the specific application; for an example of the most common response to these messages, see the Plug-and-Play awareness component in the Gallery.

There are also three other requirements your application must meet if it is file-based (that is, if your application's primary purpose is to create, edit, and save files):

* UNC path support

Your application must support Universal Naming Convention (UNC) paths. That means your application must be able to accept and store paths of the form "<br>server\share\directory" directly, without requiring the user to assign a drive letter to the server beforehand.

The MFC library supports the use of UNC paths, so you can pass a UNC path to any MFC function taking a path as a parameter. You should also use the MFC common dialog class CFileDialog when requesting a file name from the user. Make sure that any pathname-handling code that you write can handle double backslashes instead of a drive letter at the beginning of a path.

* OLE support

Your application must either be an OLE container or an OLE server, or both. In addition, if it's an OLE container, it must act as a target for drag-and-drop operations, and if it's an OLE server, it must act as a source for drag-and-drop operations. It's also recommended (thought not required) that you support Automation and provide summary information with your documents.

The section Adding OLE Support [ http://msdn.microsoft.com/en-us/library/aa269091(VS.60).aspx ] discusses how to add this functionality, using MFC.

* MAPI support

Your application must include a Send Mail command on the File menu to enable the user to send the current document as a piece of mail, using MAPI or the Common Messaging Call (CMC) API.

The section Adding MAPI Support [ http://msdn.microsoft.com/en-us/library/aa269089(VS.60).aspx ] discusses how to add this functionality, using MFC.

There are certain additions and exceptions to the previous requirements, depending on the type of application you're developing; these additions and exceptions are described in the "Logo Criteria" document.

The remainder of this section on Windows 95 logo compliance describes the changes made to the [ http://msdn.microsoft.com/en-us/library/9t94971d(VS.60).aspx ] sample application to make it meet the Windows 95 logo requirements, as well as certain user-interface recommendations; you can use this discussion as a guide for modifying your own MFC application.

