Designed for
Microsoft®
Windows NT®
Windows®95

Designed for Microsoft® Windows NT® and Windows® 95

Logo Handbook for Software Applications

Version 3.0b
March 3, 1998

Contents

1. Welcome

Welcome to the Logo Handbook. So, your first impression probably is that this is a *big* document. Don't be intimidated! It's not as hard as it looks & your products probably fulfill most of the Logo requirements already.

So, then why is this Handbook so large? There are two reasons:

First, we've gone to great lengths to explain & detail exactly what each requirement means, how to implement it, and why we believe it's important to defining a Designed for Windows application. Based on feedback from our Logo Partners, we've found that developers appreciate getting the most specific, detailed information possible about the requirements and recommendations. *So we've been thorough.*

Second, we've done a great deal of research into understanding what customers think of as an ideal application. We've worked with our Logo Partners, with end users, and with major corporations to develop the recommendations you will find in this Handbook. This document is a way for us to share this information with you, our current & future Logo Partners, to enable you to leverage not only the Windows brand, but also the valuable feedback that Microsoft hears everyday from our customers. *To this end, we've been explicit about the new recommendations, and that too made the Handbook a little longer.*

Additionally, to help make this Handbook a little easier to use, we've added a summary page - a quick overview of the logo requirements. Check it out - it is in section 5, just before the Detailed Specifications section. Do not forget this is not a substitute for careful reading of the specific requirements, but it will give you a good overview of the program.

Finally, we look forward to welcoming you to the Microsoft Windows Logo Partnership.

Sincerely,

The Windows Logo Team
Microsoft Corporation

2. What Does the "Designed for Windows NT and Windows 95" Logo Mean?

The Designed for Windows NT and Windows 95 Logo Program was developed by Microsoft to help end users easily identify desktop hardware and software products that were designed specifically for the Microsoft® Windows family of 32-bit operating systems, Windows NT Workstation (hereafter referred to as Windows NT) and Windows 95. Users can mix and match hardware and software products with the Designed for Windows NT and Windows 95 Logo and be assured that the products will take advantage of the new technologies integrated into both of these operating systems. As a software vendor, licensing the Logo enables you to use the Logo on your product packaging, advertising, collateral, and other marketing materials. This signals to customers that your product is:

- **"Tested"** - Your product has been tested and is fully functional on both Windows NT and Windows 95.
- **"Best User Experience"** - Your product has been designed to provide optimum usability and to ensure a consistent, accessible user experience.
- **"Works well with Others"** - You designed your product to take advantage of the latest software & hardware technologies available on Microsoft Windows NT & Windows 95.

Operating System Versions to Which This Logo Applies

In order to receive the Designed for Windows NT and Windows 95 Logo, all requirements will be fully tested in parallel and must pass on the latest releases of both Windows NT Workstation (version 4.0 at the time of publication) and Windows 95. If your application runs *only* on Windows 3.X, *only* on Windows 95, or *only* on Windows NT, do not submit your application for this Logo. If your application runs on Windows NT 3.51, but not on Windows NT 4.0, do not submit your application for this Logo.

There is no Microsoft Logo program for Windows NT alone for software.

Other Microsoft Logo Programs

The Designed for Windows NT and Windows 95 Logo is for commercially marketed desktop applications that run primarily on Windows NT Workstation and Windows 95. It is not for client/server applications or Windows NT Server-based applications. If your application is a client/server application or runs primarily on Windows NT Server, you should consider applying for the Designed for BackOffice™ Logo. Receiving the Designed for Windows NT and Windows 95 Logo *is not* a requirement for receiving the

Designed for BackOffice Logo. For additional information on the BackOffice Logo, see http://www.microsoft.com/backoffice/designed, or send an e-mail message to d4boinfo@microsoft.com.

Microsoft also offers the Microsoft Office Compatible Logo Program. The Designed for Windows NT and Windows 95 Logo *is* now a requirement for receiving the Microsoft Office Compatible Logo. For additional information on the Microsoft Office Compatible Logo program, see http://www.microsoft.com/office/compatible, or send an e-mail message to offcomp@microsoft.com.

Testing

Software applications are tested by an independent laboratory, VeriTest, Inc. (http://www.veritest.com/microsoft.htm or send an e-mail to logolab@veritest.com).

The Designed for Windows NT and Windows 95 Logo indicates that a software product provides all the features outlined in this handbook; it is not a "quality assurance" seal. Neither Microsoft nor the testing organization will be testing the quality of your product or ensuring that it is "bug-free." The testing organization's job is just to make sure that your product has full functionality, that the Logo features are present, and that your product is not generating frequent faults or system crashes.

Please note that the license agreement states:

You may only use the Logo as a symbol that your Product is compatible with Microsoft Windows NT and Windows 95. You may not explicitly state or imply that Microsoft or the testing organization in any way endorses your product. Also, the Windows NT and Windows 95 Logo program is not intended to be a "certification" program, that is, the Logo does not represent that Microsoft or the testing organization certifies your product(s) in any way.

3. How To Use This Handbook

Note: This document contains all of the technical requirements necessary to obtain the Designed for Windows NT® and Windows® 95 Logo. There are additional forms (Vendor Questionnaire, License Agreement, and Testing Agreement) you will need to complete in order to submit your application for testing. These are available at http://www.microsoft.com/windows/thirdparty/winlogo/

This handbook is divided into the following sections:

Welcome
A welcome letter from the Microsoft Designed for Windows Logo Team.

What Does The "Designed For Windows NT and Windows 95" Logo Mean?
This section provides an overview of the purpose of the Logo, why this Logo has value to your customers and how it differs from the Designed for Windows 95 Logo.

How to Obtain the Logo
This section describes the procedure you must follow in order to apply for and obtain the Logo.

The Logo Requirements: Overview
This section provides a high level summary of the Logo requirements.

The Logo Requirements: Detailed Specifications
This section details the complete requirements that your application must meet in order to obtain the Logo.

Legend for Detailed Specifications

Type	Definition
Required	The feature must be supported as described in order for the application to pass testing and receive the Logo.
Exception	These are situations where a requirement may not apply.
Recommendation	These features are recommended but not required at this time in order for the application to pass testing and receive the Logo. A recommendation will become a requirement in the future.
Verification	These are suggested ways that you can test whether your application meets a specific requirement or recommendation.
Tip	These are helpful suggestions. They are not recommendations or requirements.
Note	These are clarifications to the requirements/recommendations.

Exceptions, Exemptions, And Additional Requirements

This section describes exceptions, exemptions, and additional requirements for certain application categories, such as non-file-based applications, development tools, utilities, games and multimedia applications, and add-on products.

Pretesting & Verification of Compliance

This section provides tips on how to pretest your product to verify compliance with requirements before submitting for testing. We include this section to help reduce the need for product retests, and therefor speed up the testing process for you.

Other Resources

This section provides links to additional information to help you with your Windows-based application development.

Key Definitions

Term	Definition
AutoPlay	Denotes the behavior of automatically launching a program immediately after inserting a CD-ROM into the drive.
Autorun.inf	Text-based information file called during AutoPlay. Contains operating system instructions describing what executable to launch, etc.
Core Components	Core components are defined as DLLs that application developers have come to depend on for the proper functionality of their applications. Reverting to an older version or to an incorrectly implemented version may break many third party applications. In general terms, the core components are USER, GDI, and KERNEL. Core components are files which are installed by the operating system during a full installation, or which are considered operating system upgrades (for example, DirectX™, Direct3D™). This includes such things as DLLs which are installed by applets. This list is dynamic. For the most up to date list of core components for Windows NT and Windows 95, see http://www.microsoft.com/windows/thirdparty/winlogo/.
Degrade Gracefully	Does not crash the operating system (GPF or bluescreen) Dialog box or other visual and audio cue appears informing the user that the functionality is not available on *X* version of *Y* operating system User is not required to close the application, but may continue to use the other functionality .
File-Based Application	File-based applications have as their primary purpose the creation and editing of documents and include **Open**, **Save**, and **Close** commands, typically on the **File** menu of the application.

Term	Definition
Labeling Scheme	Common to games, accounting, and database programs. Users save "profiles" of reports, game states, etc. which have limited naming schemes. Labels may create actual file names, but the file naming behavior only vaguely represents the input of the user. Example: User may enter, "File Name Test", but the application would save "SAV001.EXT", or "FILAAAA.EXT" to the hard disk. These files are not intended to be accessed directly from Windows Explorer. A labeling scheme is not required to save long file names to the hard disk and is exempt from the long file name requirements.
Non-File-Based Application	A non-file-based application is one that is *not* primarily used to create, edit, and save files (although file operations may be common ancillary tasks).
Product Category	These categories are used to determine what behavior is required from the application to pass the Logo test. Includes: File-Based, Non-File-Based, Utility, Development Tool, Add-On. The final determination of what category applies to a product will be made by Microsoft.
Refcount	The act of incrementing and decrementing shareable components in the system registry under the ...\SharedDLLs key.
Shared Components	Any files which are installed by an application, but might be shared by multiple applications. Shared components are usually DLLs, but may be EXEs or any other file type. These must be "refcounted" unless they are "core components" or installed to the application's own directory.
Suite vs. Product	A suite is a collection of programs typically denoted by more than one shortcut on the Start Menu. A product can be a suite or a single executable. The final determination of whether or not a product is a suite will be made by Microsoft and VeriTest.
Telephony Centric	One of two categories of communications applications. An application that performs any telephony function beyond simple dialing of voice calls is considered a telephony *centric* application (including, for example, all applications that use a modem for data or fax calls). Examples of telephony centric include: dialers, answering machines, voicemail, interactive voice response, modem or fax communications, etc.
Telephony Enabled	One of two categories of communications applications. Telephone communications are not the primary purpose of the application. Examples of telephony enabled applications are: Personal Information Managers, address books, and databases

Term	Definition
Win32	Any 32-bit executable file described as "[PE_Win32]" by the Exedump program available on the Win32® SDK (Software Development Kit).

4. How to Obtain the Logo

4.1. Process Overview Steps

1. You must obtain and sign the following license agreements and submit them on paper with your product for testing.
   - The Microsoft Windows NT and Windows 95 Logo license, available by phone/fax service at (206) 635-2222, document #830
   - VeriTest's testing agreement, available by phone/fax service at (206) 635-2222, document #831
2. The online VeriTest Vendor Questionnaire must be filled out completely. The Vendor Questionnaire can be found on the VeriTest website at http://www.veritest.com/microsoft.htm. The more information you put in the Vendor Questionnaire, the better. This is where exceptions and requests for exemptions must be noted. Filling out the Vendor Questionnaire accurately and completely will ensure that your product is tested in the shortest amount of time possible. After filling out the Questionnaire, click the "Submit" button, and you will receive a Tracking Number from VeriTest. We recommend you write the tracking number on your product disk when you send it in for testing (see next step).
3. Submit your product to VeriTest. Your submission packet must include the following:
   - Your software product (on floppy disk, or CD-ROM)
   - The results of a pretest you have run on your product using the Installation Analyzer found at: http://www.veritest.com/installation_analyzer.htm For more information about this requirement, refer to section 4.2.3 of the Logo Requirements.
   - The VeriTest Testing Agreement (submit on paper – see above)
   - The Windows NT and Windows 95 Logo License Agreement (submit on paper – see above)

   License Agreement. Your signature on these contracts does not mean that you can start using the Designed for Windows NT and Windows 95 Logo. It means that you've agreed to the license terms and are awaiting your product's passage of the tests and Microsoft's signing of the license.

   Confidentiality Option. On the Testing Agreement with VeriTest, you are given the option to keep your testing and test results confidential from Microsoft. If you choose this option, upon passing you must sign a release allowing VeriTest to send the results to Microsoft. Only after you sign this release can Microsoft grant you license to use the Logo

1. You will receive test results from VeriTest within eight (8) business days from when your complete application package is received. VeriTest will send both you and Microsoft a copy of the test results.
2. If your product passes testing, VeriTest will then send to Microsoft the license agreement that you previously signed and submitted with your product testing to VeriTest.
3. Please read the Windows and Windows NT trademark guidelines (available at http://www.microsoft.com/windows/thirdparty/winlogo/) before using the Logo
4. Please see http://www.veritest.com/microsoft.htm for current fee schedule. All testing fees are paid to VeriTest, *not* Microsoft.
5. If your product box is printed before you obtain the license to use the Designed for Windows NT and Windows 95 Logo, you may order Logo sticker from Special Effects, Seattle, WA - an outside vendor – at specialeffex@msn.com The stickers will be delivered when you receive the license to use the Logo.
6. Microsoft will sign the license agreement and send it to the person designated as the marketing contact in the Vendor Questionnaire you submitted. At this point, you may download an electronic copy of the Logo artwork kit from a secure web server using the password provided to you by VeriTest. This contains electronic copies of the Logo and Logo usage guidelines.
7. If your company already has product(s) licensed to use the Designed for Windows NT and Windows 95 Logo, Microsoft will send you an update to the license after we receive your positive test results. This update will reflect the addition of the new product(s) to the license. Your signature on this update is required in order to complete the process.
8. The license allows you to use the Logo on packaging, advertising, your website, and other promotional materials.

4.2. Important Notes on International and Localized Versions

Because of the differences in core components of Windows NT and Windows 95 when localized to a language other than English, the Logo can only be used for localized versions of a product which are in the same language group (see below) as the one which was tested.

Double Byte Language Group - Far East: Japanese, Korean, Traditional Chinese (Taiwan), Simplified Chinese (PRC).

Single Byte Language Group - Western European: English, German, French, Spanish, Swedish, Italian, Dutch, Portuguese/Brazilian, Portuguese/Iberian, Catalan, Italian.

Single Byte Language Group - Eastern European: Finnish, Russian, Czech, Slovenian, Greek, Hungarian, Polish, Turkish, and Slovak.

Single Byte Language Group - Other: Arabic, Hebrew, Thai, and Vietnamese.

An application must be submitted in the primary language version in which it is to be marketed (where the most units are expected to be sold). VeriTest will test the application using the matching localized versions of Windows NT and Windows 95. If the application passes testing, the Logo will be licensed for use with that language version of the application only and other languages within the same language group only. For example, if an application is submitted in English and passes in English, the Logo may be used on the packaging and advertising for the English version and also any Western European language listed above. The Logo may not be used on the Japanese version for example, (unless the application is also submitted and passes in Japanese or another Double Byte language.)

If the Logo is desired for a version of the application which is in another language group, then a version of the application localized to a language in the intended group must be re-submitted and a full testing fee (plus possible additional charges) will be applied to this additional testing. If the application passes testing in this new version, the Logo is licensed for use with that language version as well as other languages within the same language group.

You do not need to notify Microsoft if distributing the product internationally, unless distribution includes the PRC, Taiwan, or Korea. If you are distributing to these countries, please send the Windows Logo Department (winlogo@microsoft.com) a note with your company name, title(s) being distributed, and countries where they are being distributed. Please note the following language from the License Agreement:

The license right set forth in Section 2(a) shall not extend to the Republic of China ("Taiwan"), South Koreá ("Korea"), or the People's Republic of China ("PRC"), unless and until COMPANY provides MS with written notice of COMPANY's intent to distribute Product in these countries. COMPANY agrees not to use the Logo in such countries and shall not be licensed pursuant to this Logo Agreement to do so until COMPANY has provided MS with such written notice.

5. The Logo Requirements: Overview

This section provides a brief overview of the Logo requirements. This list is not all-inclusive and all Logo testing will be performed against the requirements as detailed in section 6 of this Handbook. *Review of this list is not a substitute for a careful reading of the requirements.*

Tested & Reliable

Basics: Applications must be functional & stable on both Windows NT 4.0 and Windows 95. Applications & components must be 32-bit. Each product in a bundle of apps must pass Logo requirements for the bundle to carry the Logo.

Best User Experience

Installation: Applications must provide graphical install, properly register all components, auto-detect the OS onto which they're installing and default installation to the "drive:\Program Files" directory. Applications must pretest using Install Analyzer tool.

Uninstall: Applications must completely uninstall from a system by means of an automated, registered uninstaller. Applications must not remove core components or shared components necessary to other applications.

UI/Shell: Applications must use system metrics, must support the high contrast viewing option, must provide keyboard access** to all features, and expose keyboard focus** to enable accessibility aids such as screen magnifier accessories.

UNC/LFN: Applications must support Universal Naming Conventions and Long File Names. (Note LFN support for leading and trailing periods is no longer required.)

*** Testing Posponed until Logo 4.0 -- for more information refer to http://microsoft.com/windows/thirdparty/updates.htm*

Works well with others

Internet: ActiveX controls must be signed.

OLE/COM: Applications must be an OLE container and, or, an OLE object server. Applications must support Drag & Drop.

Communications: All telephony functionality must use TAPI. An application performing simple outbound dialing must either use the *full* TAPI interface for call control or pass requests to a telephony centric application that does (like Phone Dialer.) Telephony centric applications must use the full TAPI interface.

Exemptions, Exceptions & other requirements

All categories of applications listed here are required to meet all Logo requirements but have some exemptions and some additional requirements specific to the category. Be sure to read these sections carefully to determine whether or not they apply to your application

Non file-based Applications: Are exempt from OLE and UNC requirements.

Development Tools: must be capable of creating apps that can pass the Logo requirements, must submit sample apps.

Utilities: Must provide meaningful functionality for both Windows NT & Windows 95.

Games & Multimedia: Service Pack 3 for Windows NT has eliminated many compatibility issues for game developers; because of this, products using Direct3D or DirectSound or DirectX 5.0 must now apply for the Designed for Windows NT and Windows 95 logo. For these games and multimedia applications, the installation application must query the operating system and degrade gracefully (i.e., does not allow loading on the target system and displays a meaningful message to the user) on Windows NT systems without Service Pack 3.

Java Applications: Must run and re-distribute the Microsoft Win32 virtual machine. Java applications are exempt from OLE/COM, UI/Shell requirements.

Add-ons: Must be used by a 32-bit application capable of earning the Logo and must be tested both separately and with target suite.

This list is *not* all-inclusive and is *not* a substitute for a detailed reading of Logo requirements in section 6 of this Handbook. Be sure to read each section in the detailed requirements carefully.

6. The Logo Requirements: Detailed Specification

6.1. Basic Requirements

6.1.1. 32-bitness

Required:
An application must be a Win32 application programming interface executable file compiled with a 32-bit compiler that generates an executable file of the PE (Portable Executable) format ("[PE_Win32]" as reported by the Exedump program on the Win32 SDK). This means that all the major program files (DLLs and EXEs) must be 32-bit with the exceptions stated below.

Note:. On Windows NT, thunks allow applications to call from 16-bit code to 32-bit code. However, calling from 32-bit to 16-bit on Windows NT is not supported; you need 32-bit equivalent code for Windows NT. On Windows 95, thunks allow applications to call in both directions. Fully 32-bit code allows applications to ship a single binary.

Exception:
If your application is not represented in PE format (interpreted code, for example), then the "run-time engine" must be a Win32-based executable file in the PE format. For example, if you develop an application in Microsoft Access, your application is an .mdb file, not an .exe, but Access.exe would need to be a Win32-based executable file in the PE format.

Exception:
Under the following circumstances, certain 16-bit code elements may be included in your product:

? Some minor, subsidiary 16-bit executable files and dynamic-link libraries (DLLs) may be used in order to provide backward compatibility and links to 16-bit products. However, your main program files (including DLLs) must be 32-bit.

? Other minor, subsidiary 16-bit executable files and DLLs may be used for convenience in a very limited fashion, however, only if there is no existing 32-bit version of the DLL you need for your application. These files cannot constitute a significant portion of the product.

? You must fully explain any and all instances of 16-bit code when you submit your product for testing. This must be done in the electronic Vendor Questionnaire you submit with your application.

Required:
The application must support multitasking. It must support either Alt-Tab or Ctrl-Esc to switch focus from full screen to the desktop.

6.1.2. Stability and Functionality

Required:
The product must be fully functional and stable on the most recent versions of both the Windows NT and Windows 95 operating systems. We will do vigorous stability testing to confirm that your product does not adversely affect the overall stability of Windows NT or Windows 95. See section 8.1, Pre-testing & Verification of Compliance for specific stability tests.

6.1.3. Product Distribution

Required:
In order for a bundled package of applications (commonly known as a "suite") to use the Logo, each individual product in that suite must be tested and pass the Logo requirements. In order for each individual product to use the Logo when shipped separately, it must be retested for compliance with Logo installation & uninstallation requirements only.

Required:
The Logo program is for 32-bit Windows-based applications. However, vendors may distribute 16-bit versions of their product on the same distribution media as the 32-bit Logo tested product provided the installer clearly distinguish the difference between the 16-bit product and the 32-bit Designed for Windows NT and Windows 95 products. A message must be included such as "Do you also wish to install Product Y? This is a 16-bit product that is not designed for Microsoft Windows NT and Windows 95 but must operate in this environment."

Required:
Windows NT runs on multiple hardware platforms. Windows 95 is a single-platform operating system. You must provide a version of your application that runs on both Windows NT 4.0 and Windows 95. You may wish to ship multiple platform versions (different binaries that work on platforms such as Intel® x86, , or Alpha AXP™) of your application in the same package. All multiple platform versions may be submitted simultaneously for testing.

? If your product can be installed on multiple platforms from a single distribution media (for example, a CD-ROM), the installer must automatically detect the platform. It is not acceptable to ask the user which platform to install.

? If you ship your application in a multiple-platform package on which you are using the Logo, you must include in that package a version of your application that runs on Windows 95. Similarly, any advertising or marketing materials on which you use the Logo must refer to a package that includes a version of the product that runs on Windows 95.

6.2. Installation

6.2.1. User Experience

Required:
The product must have an installation program with a graphical Setup program that executes in a 32-bit Microsoft Windows operating system.

Required:
The installer must either lead the user through the process in a user attended mode, (not just instruct the user to copy or decompress files) or provide a facility for a 'silent' or unattended install requiring no user input. Preferably, (though not required) the application will provide both options.

Required:
The installer must automatically detect the version of Microsoft Windows NT and/or Windows 95 running and install the correct version of your product automatically. This must be a seamless experience for the user.

Required:
All shortcuts created by the installation must be viable, that is, not result in "file not found" or other error conditions when activated.

Required:
Products distributed on CD-ROM must utilize the AutoPlay feature to begin setup or launch the program itself the first time the application is run. It is up to the vendor whether AutoPlay is enabled on subsequent insertions of the CD-ROM. In the case of products distributed on multiple CD-ROMs, subsequent CD-ROMs must either utilize the AutoPlay feature or behave during install like a subsequent floppy disk (continue install without prompting user for action.) It is not acceptable to require the end user to Start, Run the CD-ROM during installation.

Required:
The installer for media other than CD-ROM must provide the ability to launch the installer through **Add/Remove Programs** in the Control Panel.

Required:
The application must default installation to a directory (or directories) under "drive:\Program Files". The installer must query the registry for this directory path to ensure appropriate install if, for example, a user has renamed this directory on their machine, or if the application is being installed onto a non-English language, localized version of Windows.

Required:
The application's installer must not install application DLLs or executables under the system root.

Required:
The installer must check **before** starting any operation to ensure that it can complete that operation. The following are the requirements based on this principle.

- Installers must fail gracefully in the case where a system component cannot be replaced because file system security prevents an existing file from being overwritten. Installers that need to overwrite system files, for example, should check up-front if the current user is a member of the administrators local group. This will avoid confusion later by preventing the user from getting file copy errors from which there is no recovery.
- Applications must be tested on Windows NT under non-administrator accounts. An application must run under a user account but that user must not be able to change the application setup configuration.
- When installing the application, the installer must check for the user privilege level. If the user is not an administrator and the application will work but with limited functionality, the installer must warn the user that only limited functionality will be available since they do not have admin privileges, and the installer must allow them to discontinue the installation. This may be accomplished by using the Sample Code below or the KB article PSS ID# Q118626.

Sample Code:

```
//------------------------------------------------------------//
//
// IsAdmin() - tests to see if the current user is an admin
//
//------------------------------------------------------------//

BOOL IsAdmin() {

  SC_HANDLE hSC;

  //
  // Try an Admin Privileged API - if it works return
  // TRUE - else FALSE
  //
      hSC = OpenSCManager(
              NULL,
              NULL,
              GENERIC_READ | GENERIC_WRITE | GENERIC_EXECUTE
              );

    if( hSC == NULL ) {
        return FALSE;
    }

  CloseServiceHandle( hSC );
```

```
    return TRUE;
}
```

Recommended:
Applications should not assume that directory names will be in the English language, or unchanged by the user (for example, "Program Files".) Applications should query the registry directly to obtain the proper, language specific directory names. Vendors should be aware that English versions of their application might be installed onto a non-English language version of Windows. In this instance a search for the English language directory string "Program Files" would not be successful & would result in a failed install. For other folders the proper paths can be queried via the **SHGetSpecialFolderLocation()** function with the following CSIDL flags:

```
#define CSIDL_PROGRAMS                   0x0002
#define CSIDL_PERSONAL                   0x0005
#define CSIDL_FAVORITES                  0x0006
#define CSIDL_STARTUP                    0x0007
#define CSIDL_RECENT                     0x0008
#define CSIDL_SENDTO                     0x0009
#define CSIDL_STARTMENU                  0x000b
#define CSIDL_DESKTOPDIRECTORY           0x0010
#define CSIDL_NETHOOD                    0x0013
#define CSIDL_TEMPLATES                  0x0015
#define CSIDL_COMMON_STARTMENU           0x0016
#define CSIDL_COMMON_PROGRAMS            0X0017
#define CSIDL_COMMON_STARTUP             0x0018
#define CSIDL_COMMON_DESKTOPDIRECTORY    0x0019
#define CSIDL_APPDATA                    0x001a
#define CSIDL_PRINTHOOD                  0x001b
```

Recommendations to enable roaming users & unattended install:

Recommended:
User settings should be maintained across application version changes. It's also recommended that user settings be able to be moved from machine to machine. Examples of user settings include customized toolbars, any dynamic, user-generated lists or other user specific options. For information about supporting a Networked or Managed environment refer to section 6.2.4.

Recommended:
If the application can be sure it is installing on a single user machine, it should collect user name or Personal ID (PID) information the first time the application is run, not during install.

6.2.2. Using the Registry

Required:
Native data file types (if applicable) must be registered as follows:

```
[HKEY_CLASSES_ROOT]
.(file type extension)
        (Default) = REG_SZ:FileTypeID
```

See the "Windows Interface Guidelines for Software Design," chapter 10 (available from MS Press), for further details on registering file types and application data.

Required:
The application installer must register *all* shared components designed to be uninstalled under the following Registry key:

[HKEY_LOCAL_MACHINE]\SOFTWARE\Microsoft\Windows\Current Version\SharedDLLs

Required:
The installer must *not* register components that are self-registering, such as DirectX and Direct3D. The installers for these components will register their components as needed.

Required:
Core components must *not* be refcounted in the registry by application installers. Please see http://www.microsoft.com/windows/thirdparty/winlogo/ for the most up-to-date and complete list of Windows 95 and Windows NT core components.

Required:
The application must not add information to Win.ini or System.ini.

Exception:
Certain multimedia applications may require video codecs not supplied with Windows NT or 95. Products that must support links to existing 16-bit products may write to these files. Certain other products, such as screen savers, may not be supported in the Registry and must write to the .ini files. Writes to the embedded section of Win.ini may be unavoidable. It is okay to write to the font section of Win.ini and to leave behind fonts and font information after uninstallation.

Any writes to Win.ini or System.ini must be explained in the Vendor Questionnaire.

Recommended:
In addition to the above requirements, the following behaviors are strongly recommended with regard to registry usage:

? Applications should provide the following set of information in the registry. This information will be used by the Add/Remove Programs Control Panel to provide a central place for information about the application.

? The registry values in the table below should be written under the following key:

? HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\NameofApplication

Name of Value	Type	Contains
DisplayName	REG_SZ	Display Name of Application.
UninstallPath	REG_SZ*	Full path to the Application's uninstall program.
ModifyPath	REG_SZ*	Full path to the Application's setup/modify program.
InstallLocation	REG_SZ*	Full path where the Application is located (folder or *.exe).
InstallSource	REG_SZ*	Location where the Application was installed from.
DisplayVersion	REG_SZ	Displayed version number of Application.
VersionMajor	DWORD	Major Version Number of Application.
VersionMinor	DWORD	Minor Version of Application.
Publisher	REG_SZ	Publisher/Developer of Application.
ProductID	REG_SZ	Product ID of installed Application.
RegOwner	REG_SZ	Registered owner of Application.
RegCompany	REG_SZ	Registered company of Application.
HelpTelephone	REG_SZ	Telephone number for support/help.
HelpLink	REG_SZ*	Full path of Helpfile or Help URL for Application.
URLUpdateInfo	REG_SZ*	URL of Update Informaton for Application.
URLInfoAbout	REG_SZ*	URL that provides a link to the publisher's home page or the application's home page.

* For Windows NT 4.0, Windows95, and future versions of Windows 95, the REG_SZ notation should be used for all registry entries requiring string values. However, for Windows NT 5.0, the REG_EXPAND_SZ notation should be used whenever possible for all string entries with asterisks.

InstallSource: This is the path from which the application was installed. It could also be an HTTP site or a network UNC path, such as \\applications\sample. If the application were installed from floppy, then this value would be a:\, for the floppy drive letter. This value also supports multiple paths. To add additional paths, just separate each path with a semicolon. For example:
InstallSource = \\applications\sample; \\corpnet\sample

VersionMajor/VersionMinor: Future versions of Add/Remove Programs will use these two values to compute whether an upgrade is necessary. These values will not be displayed to the user. The numbering and ordering scheme of these numbers are up to the application. Note that newer versions of the application must have a larger VersionMinor and/or a larger VersionMajor field to indicate an update.

Note the following example of registration information for an NT 5.0 application:

Name of Value	Type	Contains
DisplayName	REG_SZ	Sample Application
UninstallPath	REG_EXPAND_SZ	%SYSTEMDRIVE%\Program Files\Sample\uninstall.exe
ModifyPath	REG_EXPAND_SZ	%SYSTEMDRIVE%\Program Files\Sample\modify.exe
InstallLocation	REG_EXPAND_SZ	%SYSTEMDRIVE%\Program Files\Sample
InstallSource	REG_EXPAND_SZ	A:\
DisplayVersion	REG_SZ	1.0a
VersionMajor	DWORD	1
VersionMinor	DWORD	1
Publisher	REG_SZ	Sample Corp
ProductID	REG_SZ	111-111-111
RegOwner	REG_SZ	Test User
RegCompany	REG_SZ	Test Company
HelpTelephone	REG_SZ	1-800-555-5555
HelpLink	REG_EXPAND_SZ	%SYSTEMDRIVE%\Program Files\Sample\sample.hlp
URLUpdateInfo	REG_SZ	http://sample.com/sampleapp/update.html
URLInfoAbout	REG_SZ	http://sample.com/sampleapp/default.html

- Installers and applications should store string data as separate REG_SZ, REG_MULTI_SZ or REG_EXPAND_SZ values instead of embedding such strings as REG_BINARY data in the registry. This helps resolve ANSI to Unicode conversion issues when upgrading a Windows 95-based machine to future versions of Windows NT.
- Installers and applications should make use of REG_EXPAND_SZ and the %SystemRoot%, %windir%, and %SystemDrive% environment variables. In particular, do not hardcode paths to the Windows system root or to the drive containing Windows in the registry since users may remap their drives under Windows NT. For example, under Windows NT:

```
MyPath : REG_SZ : "C:\Program Files\MyApp"      // bad

MyPath : REG_EXPAND_SZ :
    "%SystemDrive%\ProgramFiles\MyApp"     // good
```

- The Win32 API ExpandEnvironmentStrings can then be used when the string is retrieved to get the up-to-date path.
- If your program uses a file extension that is used by other applications (for example, .htm files), the installer program should check the registry and ask the user if they want to change the default program for that particular file extension.

6.2.3. System & Shared Components

Required:

Applications must not overwrite core components with older versions of the components. The vendor must check http://www.microsoft.com/windows/thirdparty/winlogo.htm for the most up-to-date and complete list of Windows 95 and Windows NT core components.

Required:
The vendor must run Pretest the application using VeriTest's Install Analyzer & submit the report results with the Vendor Questionnaire. Improper handling of core and shared components is the number one reason applications fail Logo testing on the first try and must retest. The Install Analyzer provides automated testing of compliance in this area. The Install Analyzer can be found at http://www.veritest.com/installation_analyzer.htm

Recommended:
It is strongly recommended that applications do not write anything to the \system32 directory. On Windows NT 5.0 the OS will support a mode in which this directory will be read-only and unavailable to applications & installers. This will be part of the Zero Administration Windows effort to reduce DLL conflicts and other shared component problems.

Recommended:
It is strongly recommended that if your application requires updated system DLLs or other core components you must ship a self-extracting EXE from Microsoft which will handle installation of these components. Applications are strongly discouraged from installing system components or install into Windows\system32 by any other means than Microsoft provided service packs. (*MS will distribute* ***all*** *new DLLs by means of service packs only. Vendors should distribute self-extracting EXEs from Microsoft rather than the latest DLLs.*)

Recommended:
System Reboots should be avoided (this is strongly recommended.) If the system must be rebooted (for example, after installing a service pack for the OS) setup should gracefully resume where it left off.

6.2.4. Install Support of a Networked or Managed Environment

As the world moves to an increasingly networked environment, applications will need to enable networked usage scenarios. These scenarios include roaming users, remote administration, automatic updates of software, etc. The goal of this section is to strongly encourage vendors to write all user specific information to user Profiles and expose relevant information for remote management to Policy. This gives an end user or administrator the option to store (and thus manage) user information on a remote, shared server. The benefits of this approach are to enable roaming users, remote install, and Policy based management. Note that both Windows 95 & Windows NT 4.0 support the use of remote User

Profiles today. For more information see the Windows 95 Resource Kit or the Microsoft Zero Administration Kit.

Recommended:
The following behaviors are recommended for Installation in a Networked or Managed environment:

For more information about how your applications can support a networked or managed environment refer to http://www.microsoft.com/windows/zaw

- Registry Use: It is recommended that HKCU settings that are application defaults or part of an administered profile should not be written to the registry until the user modifies them (through setting/option changes, etc.). Until such a time, the settings should reside within the application. Therefor any HKCU keys that are written to the registry should be different than those that reside in the application (defaults).
- Setup should support network installations that enable network administrators to add additional components to network installs. (for example, third-party tools, add-ons, etc)
- Setup should support creating a disk image that can be installed onto multiple machines. In this situation machine & user specific settings should be collected the first time the application is run.
- Setup should not install shared files in non-shared volumes. Files shared by multiple users or applications should reside in a location that is common and accessible.
- Applications should support silent installs & 'push' strategies in managed environments.
- Applications should be capable of running directly from source in the case of CD or network installs. This should be done with no local machine footprint. (Local use of the registry is considered 'zero-footprint' where unavoidable.)

6.2.5. Use of Policy in a Managed Environment

About Policy

This section provides an introduction to the use of system policy to help create a managed environment. It is by no means exhaustive and is included to introduce software vendors to development resources available to assist them in supporting this useful management technique.

Windows NT 4.0 provides administrators the ability, through a single interface, to customize aspects of the user's environment and restrict the actions a given user can perform. This enables an administrator to reduce the end-user "futz-factor" and thereby greatly reduce application support costs. The configuration that the administrator develops for users and machines is referred to as a system policy. The policy actually takes the form of a file, which at logon is parsed, and the registry settings contained within that file are applied to the registry on the client machine to create a specifically defined environment for the user. When a system policy is applied, the existing user-specific or machine-specific registry settings can be overwritten with the settings the administrator has established in the policy file. This gives the administrator the ability to set restrictions on the client machine and end-user. These policy settings can be applied individually to a specific user account, or computer, or broadly to the group membership of a set of users, for example. With a properly implemented policy, regardless of where a user logs on, that user's environment can be customized to the administrator's specification.

Recommended:
It is recommended that applications include a system policy *template* to enable administrators to remotely manage the roll-out and use of the application. By utilizing a system policy template, global changes to registry parameters can be applied to all users, or a subset of users at the next interactive logon. It is recommended that the application offer the administrator a template of registry settings to choose from in customizing that application's rollout or behavior on user's machines. This template is referred to as an ADM file. The administrator (or software developer) can mandate a value, remove a value, or disregard the current state of a particular value at each interactive user logon. For example, a vendor can provide a template to administrators that can set the default server locations in each user's personal preferences.

Recommended considerations for developing applications to support use of system policy:

? It is recommended that vendors include an ADM file with their application or make such a file available for download from the vendor's web site.

? If the application utilizes the Run or Find dialog, support should be added for the scenario when a system policy is in effect that disables the user's capability to use these features.

? If the application is a shell extension, it should support the system policy NoViewContextMenu, which disables the context menu on the desktop or within the Explorer interface.

? If the application stores paths in the registry, the application should utilize ExpandEnvironmentStrings to support variables as part of the path, such as %USERPROFILE%. Hard-coded path names should not be used.

? When spawning an application, use the Win32 API ShellExecute instead of CreateProcess. As a component of system policies, administrators can define which applications cannot be started from the Explorer interface. Utilizing ShellExecute allows this check to occur before starting the application.

- Policy keys added to the registry should be created under either HKLM\Policy or HKCU\Policy and should follow the "software" key style, for example: HKLM\Policy\companyname\NameofApplication\<values>. If these keys do not already exist, they should be created following this format.

For details on Windows NT 4.0 System Policies, a whitepaper will be available in May 1997 on Microsoft's Web site titled "Windows NT 4.0 Profiles and Policies" at http://www.microsoft.com/windows/zaw. Additionally, more information on generating ADM files for use with System Policy Editor can be found in the Win32 SDK.

6.2.6. Supporting Migration from Windows 95 to Windows NT

Recommended:
An application should fully operable after the operating system is upgraded (for example, from Wind^ws 95 to Windows NT). To do so, applications should comply with the following:

- When installing on one 32-bit Microsoft Windows operating system, the installer should provide an option to the user whether to install all additional binaries that would be required to make the program operate on the other operating system. The application should determine at run time which components to use. For example, if the application requires a different DLL on Windows NT than on Windows 95, the application should install both and pick the one to actually load and use at application startup time. If the application requires a different .exe for each operating system, then write a stub .exe that calls the correct .exe based on a run-time version check (GetVersionEx() Win32 API).

6.2.7. Win95 to Win NT 5.0 Application Migration DLL

With the coming launch of Windows NT 5.0, some Windows 95 users will migrate to Windows NT. The purpose of this section is to provide recommendations that will ensure that an application installed on a Windows 95 PC will run smoothly without the need to be re-installed after an upgrade to Windows NT 5.0.

Many of the recommendations listed below will become requirements in the next revision of the Designed for Windows Logo program so please review this section closely. (Microsoft is preparing a Windows 95 to Windows NT 5.0 Migration Kit to assist vendors in the creation of this DLL. This will be available with Windows NT 5.0 beta 1.)

Recommended:
Review your product to ensure that it:

- Does not write to different registry locations on Windows 95 and Windows NT (registry values may also be different)
- Does not install different files (DLLs, EXEs, etc.) on the two operating systems (or additional files on Windows 95)
- Does not install different versions of files that are common to both Windows 95 and Windows NT
- Does not make operating system specific calls on Windows 95 that are not available on Windows NT

Future versions of the Designed for Windows Logo will **require** that the Windows 95 installation of the product is functional after a user upgrades their Windows 95 machine to Windows NT 5.0 or later. To accomplish this, vendors should ensure that the differences between the Windows 95 and the Windows NT versions of their applications are minimal.

Recommended:
If an application needs to move or modify registry entries, move files or system settings in order to get the Windows 95 version of that application to run on Windows NT, vendors should write a migration DLL as described in the Windows 95 to Windows NT 5.0 Migration Kit. The migration DLL should perform all tasks necessary to make the Windows 95 version of the application functional on Windows NT 5.0. This should include modifying or adding new registry entries and copying new files to their proper locations on Windows NT 5.0.

Recommended:
During an upgrade it is strongly recommended that the migration DLL not require user interaction or display a User Interface to the end-user.

For new applications, it is recommended that vendors create a migration DLL and place it on their product media so that users can find them in case they need to upgrade their computers from Windows 95 to Windows NT 5.0 or later.

This upgrade DLL should be tested by installing the application on Window 95 and then upgrading it to Windows NT 5.0 or later using the rules described in the Windows 95 to Windows NT Migration Kit (to be available with Windows NT 5.0 beta 1).

Watch for more information on Windows 95 to Windows NT 5.0 migration on: http://www.microsoft.com/

6.3. Uninstall

6.3.1. User Experience - Uninstall

Required:
The product must provide a fully automated uninstaller that removes the program files, folders, and Registry entries for the 32-bit Microsoft Windows 95 and Windows NT operating systems.

? The uninstaller must be accessible through Add/Remove Programs on the Control Panel and must operate properly from the Control Panel.

? The uninstaller must be properly registered and must appear under Add/Remove Programs in the Control Panel. The method for this registration is:

```
[HKEY_LOCAL_MACHINE]\SOFTWARE\Microsoft\Windows\Current
Version\Uninstall\YourProductName

DisplayName=REG_SZ:<your product name and version number>

UninstallString=REG_SZ: c:\apps\myapp\uninstll.log /h
```

Required:
All files and folders copied onto the hard disk must be removed, with certain exceptions.

Exception:
User data files including the following should remain on the hard disk:

? Resources that other programs might use, such as sharable DLLs, sharable fonts, and sharable Registry entries.

? It is better to err on the side of safety regarding other applications. If you are not sure whether removing a DLL might harm other applications, it is better to leave it behind. However, you must explain everything you leave behind when you submit your application for testing. The proper place to do this is in the Vendor Questionnaire.

Tip: Remember to remove .gid files created by your Help. Create a zero-length (0) file with the same name at install time.

Required:
The uninstaller must remove all advertisements and shortcuts placed anywhere in the **Start Menus** by its installer that are associated with the component that is being removed.

Recommended:
On Windows NT a user must not be able to uninstall an application if that user does not have adequate security permissions to complete the

uninstall. In this situation the application must provide an appropriate error message such as "You do not have adequate security permissions to successfully uninstall this application. Please contact your system administrator." A system administrator should be able to uninstall all applications that were installed (by a user or administrator) using the default installation.

6.3.2. Registry, System & Shared Components - Uninstall

Required:
The uninstaller must remove all Registry entries (with the exception of keys that might be shared by other programs).

Required:
The uninstaller must *not* decrement or remove any core component, in particular Microsoft Foundation Class Library (MFC) DLLs, as well as ODBC & DAO DLLs. Please see http://www.microsoft.com/windows/thirdparty/winlogo/ for the most up-to-date and complete listing of all Windows NT and Windows 95 core components. The core component list is updated regularly. It is the vendor's responsibility to check the web for the most up to date list.

Required:
The uninstaller must accurately decrement the count on all components your application uses that are installed as shared components.

Required:
If the Reference Count is at zero (0) and you will not be removing the component, you must leave the Reference Count at 0. This requirement prevents installers from inaccurately incrementing the Shared Count if the component was on the system, but was not registered.

Recommended:
If your installer finds a shared component already on the system and not registered, the SharedDLL (see section 6.2.2) count should be incremented by 1. plus the number of clients being installed. For example, if you install your application with 3 clients using a shared component, your installer will bump the SharedDLL count by 3. But if the shared component was already on the system and no SharedDLL exists for it (that is, the previous installer did not create the refcount), then set the SharedDLL count to 4. That way, when your application is uninstalled, it leaves the shared component on the system with a refcount of 1.

Required:
The uninstaller must remove itself

6.3.3. Uninstall Support of a Networked or Managed environment

Recommended:
Administrators should be able to uninstall and unadvertise (remove shortcuts) the application by means of a centrally managed Group Policy Template or through changes to user Profiles. Refer to section 6.2.5 for more information on the use of Policy & Policy Templates.

6.3.4. Managed Installation Service - coming with NT 5.0

Microsoft is working on new technologies to greatly simplify installation and uninstallation processes by the use of an operating system installation manager. This new installation manager will act as a gateway to the registry, handling all ref counting, component tracking, installation and uninstallation. Microsoft is working with major installation vendors to release tools to enable vendors to create install packages to work with this new system. Use of this technology will become a requirement in the NT 5.0 timeframe. Today's installation requirements and the recommended changes in registry usage reflect incremental steps towards a model in which registry entries will be managed by a central OS service.

Applications that are developed using the OS-provided installation service will be compliant by default with the majority of the Logo installation & uninstallation requirements.

Recommended:
The application should make appropriate use of user Profiles. In the NT 5.0 timeframe this means:

- Maintain strict separation of user data from application bits whenever possible.
- On Windows NT, applications should use the document storage folder in the user's Profile to store user data. The Profile path can be queried via the SHGetSpecialFolderLocation() function with the CSIDL_PERSONAL flag.
- On Windows NT applications should use the ...\winnt\Profiles\username\Application Data and ...\winnt\Profiles\username\Personal folders.
- On Windows 95 applications should use ...\windows\Profiles\username and should create \Application Data and \Personal subfolders. (Applications will need to create these subfolders on Windows 95)
- User data should by default be stored in the user Profile
- If the application settings are no longer part of a Profile, the application should recover gracefully and regenerate the needed settings based on the application's defaults.

6.4. UI/Shell

Required:

The application must be compatible with the following system color, size, font, sound, and input settings. This provides the user with a consistent user interface and allows them to customize the system to meet their needs and preferences. These settings are queried using the GetSystemMetrics, SystemParametersInfo, or GetSysColors functions. Color settings are required only when the High Contrast option is on. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

Tip: Standard window classes and controls automatically support all of these settings. Applications only need to be aware of these when creating custom window classes or controls.

Required Size Settings	
SM_CYFIXEDFRAME	SM_CXFIXEDFRAME
SM_CYDLGFRAME	SM_CXDLGFRAME
SM_CYMENUSIZE	SM_CXMENUSIZE
SM_CYSIZEFRAME	SM_CXSIZEFRAME
SM_CYFRAME	SM_CXFRAME
SM_CYVSCROLL	SM_CXVSCROLL
SM_CYMENU	SM_CYCAPTION
SPI_GETICONTITLELOGFONT	SM_CYSMCAPTION
SPI_GETNONCLIENTMETRICS	SPI_GETBORDER
SPI_GETWORKAREA	SM_CXSCREEN
SM_CYSCREEN	SM_CXFULLSCREEN
SM_CYFULLSCREEN	SM_CXMAXIMIZED
SM_CYMAXIMIZED	SM_CYTRACK
SM_CXTRACK	

Required Color Settings	
SPI_GETHIGHCONTRAST	All GetSysColor settings are required when SPI_GETHIGHCONTRAST is TRUE

Required Input Settings	
SM_CYDOUBLECLK	SM_CXDOUBLECLK
SM_CYDRAG	SM_CXDRAG
SPI_GETKEYBOARDPREF	

Required Sound Settings	
SM_SHOWSOUNDS	SPI_GETSHOWSOUNDS

Required:

Applications must be compatible with the High Contrast option, which indicates that the user wants a high degree of legibility. When this option is set the application must use only system colors selected through Control Panel or other colors that the user can customize. Because documents are typically shared, a user must not be required to alter a document in order to adjust the colors drawn on the screen. System colors must be used in their proper foreground/background combinations. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

Tip: To get the space left over after the tray takes up the screen, use either GetSystemMetrics to query SM_CXFULLSCREEN and SM_CYFULLSCREEN or use SystemParamtersInfo to query SPI_GETWORKAREA.

Exception:
Palettes where the user selects a color may continue to display their full range of fixed colors. Games and certain multimedia applications may have aesthetic and design considerations, such as custom dialog boxes and user interface elements, where system settings are inappropriate. Games, certain multimedia applications, and inherently graphical applications such as image editing tools may also be excepted from color requirements. These will be judged on a case-by-case basis. You must detail these exceptions in your Vendor Questionnaire.

Recommended:
In addition to the required system settings listed above, applications should be compatible with the following recommended system settings.

Recommended Size Settings	
SM_CXICONSPACING	SM_CYICONSPACING
SM_CXMENUCHECK	SM_CYMENUCHECK
SPI_ICONHORIZONTALSPACING	SPI_ICONVERTICALSPACING
SM_CXICON	SM_CYICON
SM_CXSIZE	SM_CYSIZE
SM_CXSMSIZE	SM_CYSMSIZE
SPI_GETICONTITLEWRAP	SPI_GETICONMETRICS

Recommended Sound Settings
SPI_GETBEEP

Recommended Input Settings	
SPI_GETMOUSEHOVERHEIGHT	SPI_GETMOUSEHOVERTIME
SPI_GETMOUSEHOVERWIDTH	SM_SWAPBUTTON

Other Recommended Settings	
SPI_GETSCREENREADER	SPI_GETMENUDROPALIGNMENT

Required:
Testing Posponed until Logo 4.0 -- for more information refer to http://microsoft.com/windows/thirdparty/updates.htm

The application must provide keyboard access to all features. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

Exception:
Exceptions may be made in the following cases:

- Games that are dependent upon the use of a mouse or joystick. It is recommended, but not required, that games provide keyboard support where feasible. They should not require a pointing device except where it is unavoidable due to the nature of the game.
- CAD and similar applications that are dependent upon graphing tablets and other input devices.
- Applications that are entirely graphical, such as image editors, may rely on the MouseKeys feature built into the 32-bit Windows operating systems to allow users to move the mouse pointer. However, this is not acceptable for drawing applications that allow the user to independently manipulate separate text and graphic objects.

Exceptions may be made for minor features that are not required for operation of the program, when the major features in the application have keyboard access. All major features or functional areas without keyboard access should be explained in the Vendor Questionnaire.

Required:

Testing Posponed until Logo 4.0 -- for more information refer to http://microsoft.com/windows/thirdparty/updates.htm

The keyboard user interface must be documented.

Exception:
It is not necessary to document elements that simply follow the Windows conventions for elements such as standard menus and controls.

Tip: This information may be provided in your standard documentation that is included with the product, or it may be orderable separately. In the latter case, it is strongly recommended that the standard documentation inform the user that this additional documentation is available.

Required:

Testing Posponed until Logo 4.0 -- for more information refer to http://microsoft.com/windows/thirdparty/updates.htm

The application must notify other software of the locations of the keyboard focus, either by moving the system caret or using Active Accessibility. This enables the use of accessibility aids such as the screen magnification accessory included with the Active Accessibility SDK. Note that this accessory will be included in future versions of the Windows operating systems. Be sure to check the Pretesting &

Verification section to help test your product. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

Recommended:
In addition to the above, the following behaviors are recommended for the User Interface. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

? The application should allow other software to identify and manipulate all screen elements that the user interacts with. This should be accomplished by supporting Microsoft Active Accessibility. Standard window classes and controls automatically support this mechanism. Applications only need to add additional support when creating custom window classes or controls, or drawing content in their window client area. Note: this recommendation will become a requirement in the future and may require significant work to accomplish. It is strongly recommended that vendors begin planning for this as soon as possible. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

? The application should not convey any important information by sound alone. If sound alone is the default method for conveying the information, the application should provide an option to convey this information by other means. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

? The application should not convey important information by color alone. If color alone is the default method for conveying the information, the application should provide an option to convey this information by other means.

? We strongly recommend that the product avoid hard coding any font sizes smaller than 10 points. Small fonts can cause eyestrain and are difficult for many people to read.

? Allow the user to customize all user interface timings that are not based on standard system settings. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

? The application should be compatible with changes to the system font size and changes to the number of pixels per logical inch. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

? Allow the user to choose font names and font sizes in order to make the document or window contents more legible.

? The application should not trigger unexpected side effects based on changes in pointer or keyboard focus locations. For more information refer to the accessibility guidelines for application developers on http://microsoft.com/enable/.

6.5. Support for the Internet

Required:

Authors of ActiveX™ Controls must digitally sign their controls, as specified in the ActiveX SDK. Controls should also be marked "safe for persistence" as appropriate. For more information, see the ActiveX SDK at http://www.microsoft.com/IntDev/sdk.

Recommended:

In addition to the above requirements, the following behaviors are recommended:

? Authors of ActiveX Controls should provide support for automatic code download. If the control is a single file, then no additional work is required. If multiple files are required, then a .cab, .inf, or a self-extracting executable should be provided. Refer to the Internet Component Download spec in the ActiveX SDK.

? Any code intended to be downloaded from the Internet should support Authenticode signing. Commercial publishers should obtain a digital signature from http://www.microsoft.com/workshop/prog/security/authcode/certs-f.htm Executables should be signed to allow users to identify provenance, using procedures and technology found at http://www.microsoft.com/workshop/prog/security/authcode/codesign-f.htm

? Applications should register an Internet update site for the application when installing. This site will provide information about upgrades available for the application. Upgrades should be made available for user-initiated download. See the Installation section for more information about how to include this URL in the registry.

? Applications should also register a general product information site when installing. See the Installation section for more information about how to include this URL in the registry.

? Applications, tools, controls, or any other software that installs or sets up DLLs should honor the ModuleUsage registry section described in the appendix of the ActiveX SDK.

? Your application should support WinSock 2. Note that support for the Broadcast PC capabilities of future versions of Windows will require WinSock 2. For more information on writing WinSock 2 applications refer to the Win32 SDK online documentation.

? Your application should expose the Help files as HTML. Over time, Microsoft will phase out support for the WinHelp engine, and replace it with Web-based help.

? Your application should have an FTP save option, enabling a user to save to an FTP site in the same way as they can save to the local drive or to a network drive.

? Your application should produce documents that can be viewed from a browser. You can do this either by implementing a "Save As HTML" feature, or by adding ActiveX Document (docobj) support to your application. See the ActiveX SDK for more information about adding ActiveX Documents.

6.6. Support for OLE/COM

Required:
The application must be an OLE container or object server. Note that although an application can be both a server and a container, it does not have to be.

Tip: If you want to host Java™ or ActiveX controls in your application, you should choose to be an OLE container.

Recommended:
It is recommended that developers read the book "Inside COM" published by Microsoft Press. In addition to the above requirements, the following behaviors are recommended for application utilizing OLE/COM:

? It is strongly recommended that the system provided property set implementation be used to read and write all property sets supported by an application.

? Applications should expose all functionality through Automation & should use a dual interface to do so. A dual interface combines an IDispatch interface with a VTBL interface. For more information refer to the Automation Programmer's reference, Exposing ActiveX objects, in the Platform SDK. This facilitates scripting your application from scripting languages.

? Applications that expose a public Automation interface should provide a type library.

? Applications should use component categories. (In particular, containers should indicate which standard component categories they support. Also, component categories should be used in preference to adding a key such as "Control" under a component's CLSID registry key.) This recommendation allows an object to publish its capabilities to other objects. For more information refer to the Platforms SDK, Control & Container guidelines.

? Applications should support Active Accessibility. For more information refer to information for application developers on http://microsoft.com/enable/ .

Exemption:
If the product falls into one of the following categories it may be exempt

from OLE support: (Be sure to review section 7 of this Handbook for a complete description of this exemption.)

? Non-file based applications
? Java Applications
? Multimedia Applications such as a game or reference title
? Utilities

Object Server Requirements:

Required:
The user must be able to drag the object to any container, including other internal documents, the desktop, or documents in another OLE-supporting application.

Required:
Object servers must pass basic OLE server functionality testing. You can test this by dragging an object created by your product into an applicat:on such as Microsoft Word.

Recommended:
In addition to the above requirements, the following behaviors are recommended for Object servers:

? All object servers should be ActiveX Document servers.
? Object servers should close automatically after delivering their object.

Container Requirements

Required:
Your product must pass basic OLE container functionality testing. You can test this by creating an object in any standard OLE server, such as Microsoft Word, and dragging it into your product.

Required:
Containers must provide an Object command on the Insert menu.

Required:
Containers must implement the IDropTarget and/or IDropSource interfaces for drag-and-drop functionality. This means that the application can accept drops from any IDropSource() or can drag into any IDropTarget(), or both.

Tip: If you use the Windows 95 and Windows NT common controls, or the RichEdit control for your edit controls, your application already has support for Drag and Drop.

Recommended:
In addition to the above requirements, the following behaviors are recommended for containers:

? All containers should be ActiveX Document containers.

? Containers should support ActiveX Accessibility.

? Containers should provide support for modifier keys.

? Containers should providing linking and in-place activation.

? Containers should support the structured storage compound file format. Summary information should be usable and complete as presented by the 32-bit Microsoft Windows operating system shell.

6.7. Communications: TAPI 2.x

Telephony *enabled* applications Vs. Telephony *centric* applications

A telephony *enabled* application is one capable of simple outbound dialing. If an application performs more than simple outbound dialing, it must implement the *full* TAPI interface and meet the requirements for telephony *centric* applications.

For more information on writing TAPI applications refer to the Win32 SDK online documentation. For more information on writing TAPI service providers refer to the TAPI Service Provider documentation.

Required:
Communications applications that dial, answer, and otherwise control telephone calls on equipment such as modems, ISDN adapters, and telephones must control those calls using the Windows Telephony API (TAPI). This includes online registration of an application. All telephony *enabled* applications must either use the *full* TAPI interface or must pass requests to a telephony *centric* application that does.

Telephony *enabled* applications

Required:
Telephony *enabled* applications must, at minimum, use the Assisted Telephony interface and pass requests for dialing of calls to a telephony *centric* application such as *Phone Dialer* that uses the full TAPI interface for call control. When using the Assisted Telephony interface, the following additional behaviors are recommended.

Recommended:
In addition to the above requirements, the following behaviors are strongly recommended for all telephony *enabled* applications:

? In telephony enabled applications, phone number entry fields for country code and area code should use the Dialing Properties current location settings as defaults. These defaults can be obtained by calling tapiGetLocationInfo.

- A telephony enabled application should pass the number dialed to tapiRequestMakeCall in canonical/international form. This is done so that the call manager application placing the call can properly apply the user's Dialing Properties settings.

Telephony *centric* applications

If an application performs more than simple outbound dialing, it must implement the *full* TAPI interface and meet the requirements for telephony *centric* applications.

Required:
Telephony *centric* applications must use the full Windows Telephony API (TAPI) for telephony-related functions. They must not control telephony-related equipment through proprietary interfaces, drivers, or APIs other than TAPI. In particular, the application must not access a modem by directly opening the serial port using CreateFile ("COMx"), although a "Direct to COMx" mode may be available for accessing non-modem devices.

Required:
Telephony *centric* applications must apply the user's Dialing Properties settings to numbers to be dialed. This can be accomplished by passing the number to the **lineTranslateAddress** function before the call is placed. This number must be in canonical/international form unless the number to be dialed is an internal extension or the user has otherwise explicitly instructed the application to dial the digits "as is" without applying Dialing Properties (they should pass to **lineTranslateAddress** so that tone/pulse dialing settings can be applied).

Required:
Telephony *centric* applications must allow the user to select the device to use for calls. The device name can be obtained from the **LINEDEVCAPS** structure returned by the TAPI **lineGetDevCaps** function.

Required:
If a telephony *centric* application is performing extended control functions on a serial device that is also accessible through TAPI (such as a fax application issuing Class 1 or 2 fax AT commands to a fax modem), it must use **LINEBEARERMODE_PASSTHROUGH** as defined in TAPI rather than opening the serial device directly, so that the device is properly shared with other TAPI apps when not in use. The application must not open the serial device directly using **CreateFile**.

Required:
If the application uses TAPI 2.x functions, it must either explicitly link to TAPI32.dll (by calling GetProcAddress) or be linked to theTAPI32.lib *explicit* link library so that it will load on both Windows 95 (TAPI 1.4)

and Windows NT 4.0 (TAPI 2.0). The application should gracefully degrade its features so that it works as well as possible within the limitations of TAPI 1.4 when running on Windows 95.

Recommended:
In addition to the above requirements, the following behaviors are strongly recommended for all telephony *centric* applications:

- User entry of phone numbers should be in separate fields for country code, area/city code, and local number. The country code and area code fields should default to the user's current location country code and area code obtained from lineGetTranslateCaps function. The country code selected may consist of a drop-down list of countries obtained from the lineGetCountry function.
- The application should properly handle the LINE_CREATE message so those telephony devices that are added by the user while the application is active become immediately available for use.
- The application should properly respond to LINEDEVSTATE_REINIT and PHONESTATE_REINIT messages by shutting down the application's use of TAPI at the earliest opportunity so that TAPI can be restarted.
- As soon as TAPI 2.1 is available, it is STRONGLY recommended that vendors develop telephony applications utilizing TAPI 2.1 for the Windows 95, Windows NT 4.0 Workstation and the Windows NT 4.0 Server platforms. Because TAPI version 2.1 is the only version that runs on all three platforms, this will become a Logo requirement in the future.
- Client server telephony applications are recommended to use the TAPI 2.1 Remote Service Provider.
- Client server telephony applications are recommended to use the TAPI 2.1 client management APIs to enable value-added client management and control form the server.

6.8. UNC/LFN Support

Required:
Your application must support the Universal Naming Convention (UNC). UNC paths allow logical connections to network devices without the need to specifically reference a network drive letter. Your application does not need to be network-aware per se, but it does need to work seamlessly in a network environment. The system must be able to locate the network server and path with the UNC name even over a modem connection. Supporting UNC does *not* mean that the application is disallowed from presenting network drive letters to users. It merely means that the user must have the option of using only the UNC path name.

Note: An LFN is 260 characters, which in general includes 3 bytes for "**<driveletter>:**", 255 bytes for the filename+extension, and 2 bytes for the null terminator. A UNC path has 2 bytes for "\\" instead of 3 bytes for "**<driveletter>:**", and the path may not include an extension (filetype).

Required:
If your application saves files that are exposed to the user, your application must support LFNs with all of the following required features:

? Users must be able to enter names of 255 characters, including all uppercase and lowercase standard characters, embedded spaces, embedded periods, etc.

? Leading and trailing spaces must be stripped by the Save As command. To test this, enter a name with leading and trailing spaces, such as "####This is a test###", retrieve the file, and then make sure the spaces are deleted.

Note: Here and throughout this handbook, the number sign (#) indicates a spacebar space.

? Question marks anywhere in the file name must prevent the file from being saved. No error message needs to be displayed.

? The characters within the quotation marks " + , ; = [ ] " must be supported anywhere in the name, including leading and trailing. These should not cause any error conditions. (Note: support for leading and trailing periods has been dropped from the Logo requirements.)

Required:
If a file name is fully exposed in Windows Explorer, it must be a fully supported LFN. Applications are allowed to use "labeling schemes" in which a user is saving, for example, a report type or a game state, without actually creating a file that is exposed to the user in Windows Explorer.

Required:
The application must use LFNs for displaying all documents and data files in the shell, in title bars, in dialog boxes and controls, and with icons.

Required:
You must test your LFN functionality on Windows NT FAT, NTFS, and compressed NTFS partitions.

6.9. ACPI/OnNow Support

The OnNow design initiative is a set of design specifications which, when applied to system hardware and software applications, enable a PC to deliver the 'instant on' capabilities consumers have come to expect from TVs, VCRs, stereos and other appliances. This initiative and the recommendations below work to ensure that all elements of the system -- applications, devices, and user interface -- can take advantage of this PC power management technology.

Because applications have the most direct knowledge of the user's activity on a PC, they must participate in system-wide power management decision-making in order to ensure error-free handling of power down and power up scenarios. See http://www.microsoft.com/hwdev/onnow.htm for more information and technical specifications for the OnNow initiative.

Many of the recommendations listed below will become requirements in the next revision of the Designed for Windows Logo program so please review this section closely.

Verification suggestions for these recommendations can be found in the 'Pre-testing & Verification of Compliance' section.

Recommended:
Applications should handle system sleep/wake transitions by listening to WM_POWERBROADCAST messages. When applications receive sleep messages (WM_POWERBROADCAST / PBT_APMSUSPEND), they must:

? Flush any unsaved user data to non-volatile media. Applications must save this data to temporary file or backup file since the user has not given the command to save the original file.

? Close all open files.

? When applications receive wake messages (WM_POWERBROADCAST / PBT_APMRESUMESUSPEND), they must:

? Re-open any files that had been open.

? Verify the user's files and inform the user if there are any problems (for example: files can no longer be found, files have been modified by someone else, etc.)

Recommended:
Games should pause game play and stop playing sound when they receive WM_POWERBROADCAST sleep messages.

Recommended:
Applications should not ask the user any questions while it is processing a WM_POWERBROADCAST sleep or query message.

Exception:
An application may pop up a confirmation dialog only if the UI Allowed bit is set in the message and if putting the machine to sleep will cause user data loss or corruption.

Recommended:
Event handling applications such as phone answering machine

applications, fax servers, periodic maintenance or desktop management applications, etc must call the SetTreadExecutionState API before and after handling each event. This allows the operating system keep the machine awake while the event is being handled and to put the machine to sleep immediately after the event has been handled.

Recommended:
Presentation applications must call the SetTreadExecutionState API when switching into and out of slide-show mode. This allows the operating system keep the machine awake and display on while the event is being handled and to put the machine to sleep immediately after the event has been handled.

Recommended:
Applications should reduce background activities such as background pagination, background recalculation, auto-saves, auto-formatting, etc. when the computer is running on batteries. The hard disk and CPU are the highest power devices that applications have a direct impact on. The following can be done to reduce hard disk and CPU usage while on batteries:

? Performing non-essential disk activity at regular intervals may cause the hard disk to spin up and down frequently, thus using significantly more power than leaving the disk fully running. While on batteries, applications should delay all non-essential disk activity until essential disk activity is required (for example, in response to explicit user command). This allows the hard disk to remain off as much as possible. An application can detect if the hard disk is already running by calling GetDevicePowerState before accessing the disk.

? While on batteries, applications should not perform operations at idle time unless they are in response to an explicit user command. This allows the OS to temporarily stop the processor between time slices.

Applications can determine when the computer is running on batteries by calling the GetSystemPowerStatus API and listening for the WM_POWERBROADCAST / PBT_APMPOWERSTATUSCHANGED message.

Recommended:
Applications should not take any action when they receive a WM_POWERBROADCAST / PBT_APMRESUMEAUTOMATIC message. OnNow computers may be configured to automatically wake when no user is present. Since no user is present, most programs do not need to take action. If the operating system detects the user, it will send WM_POWERBROADCAST / PBT_APMRESUMESUSPEND.

Recommended:
With the introduction of OnNow applications will be run continuously (without restarting) for a much longer than those applications are typically

run today. For this reason, applications should take particular care to ensure they do not have any memory leaks. A memory leak is defined as a memory allocation that grows without bounds. Memory leaks occur when an application opens a system handle and neglects to close it. (A handle can be used to extend the functionality of your application by using features provided by the shell.) Lack of system resources, resulting from memory leaks, will cause a decline in system performance and eventual failure.

For more information on ACPI/OnNow please refer to the following web site: http://www.microsoft.com/hwdev/onnow.htm

6.10. Multi-monitor Support

Some applications may wish to take advantage of the multi-monitor capabilities coming in the next version of Windows 95.

Recommended:
The application should be tested with multiple display support to verify correct behavior when two or more displays are attached to the desktop. Behavioral anomalies to watch for include dialogs or windows positioned on a monitor other than the one expected, and windows that are prevented from moving to a secondary display.

Recommended:
The application should make use of the following system settings to be compatible with multi-monitor support.

Recommended Multi monitor System Settings	
SM_XVIRTUALSCREEN	SM_YVIRTUALSCREEN
SM_CXVIRTUALSCREEN	SM_CYVIRTUALSCREEN
SM_SAMEDISPLAYFORMAT	SM_CMONITORS

These system settings are queried using GetSystemMetrics function.

For more information on Multi-monitor support please refer to the Win32 SDK.

7. Exceptions, Exemptions, And Additional Requirements

7.1. Non-file-based Applications

The following points define a file-based application and a non-file-based application:

- A non-file-based application is one that is *not* primarily used to create, edit, and save files (although file operations may be common ancillary tasks).
  - Many accounting packages and Personal Information Manager (PIM) products are considered non-file-based applications. If you're uncertain whether your product is considered non-file-based, please explain your case in the Vendor Questionnaire.
  - Generally, applications that run exclusively in full-screen mode are not, for Logo purposes, considered file-based. An application that runs exclusively in full-screen mode is one that cannot, for example, be windowed or resized.
  - File-based applications have as their primary purpose the creation and editing of documents and include Open, Save, and Close commands, typically on the File menu of the application.
  - The primary purpose of applications such as Microsoft Word and Microsoft Excel is to allow the user to create, edit, and manipulate files. Therefore, they do need to support the requirements for file-based applications in order to be eligible to license the Designed for Windows NT and Windows 95.
  - Utilities are considered non-file-based applications.

Exception:
If your product is a non-file-based application, the requirements to support OLE and UNC pathnames do not apply. A direct example of this might be a utility, multimedia reference title, a game, or applications like Terminal and Clock, which for the most part do not save files in any way and, if they do, it's done as a facility for saving user profiles, for example.

7.2. Development Tools

Development tools are products such as languages and compilers that create executable files or interpreted code modules that mimic the action of executable files when used with a run-time engine of the developing application. These products must meet all the general requirements, with the following additions and exceptions.

Exception:
OLE drag-and-drop support is not required within the tool's design environment.

Required:
If Windows is one of the compiler's or development tool's target platforms, then the product must be capable of generating applications that can meet all the Designed for Windows NT and Windows 95 Logo requirements.

Exception:
Development tools which create Java applications do not need to meet the above requirement.

Exception:
Certain "interactive entertainment authoring tools" are not required to create applications that support OLE and UNC pathnames, as follows:

? When the interactive entertainment authoring tools are intended only for authoring non-file-based multimedia applications.

? Developers of this kind of tool must submit a sample application.

? Developers of this kind of tool must advise their customers that if they create a file-based application using this tool (whether or not they use other tools), in order for that application to be eligible for the Designed for Windows NT and Windows 95 Logo, it must meet the file-based requirements, including OLE support.

Exception:
Development tools are temporarily exempt from the requirement to expose the keyboard focus. Vendors should be aware that this exemption will be removed in the next update of the Logo Handbook.

Required:
The development tool must provide an easy point-and-shoot way (commonly known as wizards or experts) to create applications with OLE container or object server, or provide this functionality by default.

Required:
Development tools are required to submit a sample application in uncompiled form for testing. When compiled, this application must demonstrate the following:

? The Win32 PE format.

? Large and Small Icons for its executable file.

? The ability to save/retrieve files with LFNs.

? The ability to save/retrieve files with UNC file names.

? OLE Container and/or Server drag and drop functionality.

Exception:
Sample applications do not need to meet any of the other Windows Logo requirements, such as install/uninstall, etc.

Recommended:
Development tools should provide the ability to create applications that use multiple threads.

7.3. Utilities

Utilities are products such as shell extensions, disk optimizers, antivirus software, certain query engines, and accessibility aids for people with disabilities. They must meet all the general requirements contained in this document with the following additions and exceptions.

Required:
In order for a utility to receive the Logo, it must include in the same package (box) meaningful functionality for *both* Windows 95 and Windows NT environments.

Note:
Some products such as disk utilities implement operating-system-specific functionality that cannot be implemented on the other platform. For example, a set of utilities might include a disk defragmenter which operates on Windows NT but, not on Windows 95. Such products must still include meaningful functionality for the other operating system.

Exception:
Certain components of utilities may be 16-bit, such as those that must use the Exclusive Volume Locking API, soft interrupts, or components that must talk directly to 16-bit drivers. The user interface and other components of these applications must be 32-bit and use the Windows 95 thunking mechanism to access these 16-bit components.

Required:
Products that use the Exclusive Volume Locking API to access the 8.3 aliases directly and manipulate them, such as disk optimizers, file utilities, and antivirus software, must be tested extensively to ensure that these products function properly and do not damage the file allocation table (FAT) or the LFN structure. To verify, refer to the "Pre-testing & Verification of Compliance" section.

7.3.1. FAT32 Support

Recommended:
An application written for a Windows Operating System must exhibit its full functionality regardless of the filesystem of the partition it is installed upon.

For most applications, this will not be a problem (most applications are filesystem independent.) However, there is a class of applications and device drivers that rely on a partition type on the media that they run from; namely disk utilities and anti-virus utilities. For example, a disk defragmenter written for a FAT16 partition will not work for a FAT32 partition. The disk defragmenter should include support to defrag both FAT16 and FAT32 partitions.

Recommended:
Applications should not assume a 2GB hard drive size limit. On drives larger than 2GB, FAT32 is the only file system on Windows that will allow the user to format the disk as a single partition. Note that on Windows 95, the existing Win32 function **GetDiskFreeSpace** may obtain incorrect values for volumes that are larger than 2 GB. **GetDiskFreeSpaceEx** obtains correct values on *all* platforms for *all* volumes, including those that are larger than 2GB. New applications should use the **GetDiskFreeSpaceEx** function instead of the **GetDiskFreeSpace** function.

For more information on FAT32 refer to the Win32 SDK.

7.4. Games and Multimedia Applications

Required:
All products that use DirectX must fail gracefully on Windows NT 3.51, but function properly on Windows NT 4.0 and later versions.

Required:
All products that use Direct3D or DirectSound APIs must provide a comparable user experience on both Windows NT 4.0 and Windows 95 to be eligible for the Designed for Windows NT & Windows 95 Logo.

For products that use Direct3D and DirectSound APIs, the installation application must query the operating system and degrade gracefully (i.e., does not allow loading on the target system and displays a meaningful message to the user) on Windows NT systems without Service Pack 3.

Recommended:
Many of the recommendations listed below will become requirements in the next revision of the Designed for Windows Logo program so please review this section closely.

Future versions of the Logo will require Games and Multimedia applications with the following capabilities to support DirectX 5.0 (or later) *or* use the Win32 Multimedia subsystem APIs. It is recommended that applications with:

? 2D graphics output be produced using DirectDraw, in either windowed or full-screen mode.

? sound playback use DirectSound.

? game support for keyboard, mouse, and gaming devices (such as joysticks and gamepads) use DirectInput.

? hardware accelerated 3D graphics be accessed through Direct3D.

? advanced services for media integration, animation or streaming of 2-D, 3-D, video and audio support DirectX media (ActiveMovie 2.0).

For more information on DirectX 5.0 please refer to the following web site: http://www.microsoft.com/directx

7.5. Java Applications

A Java application, for the purposes of this Logo program, is defined as any application written in the Java programming language that requires a Java virtual machine to run. This section details the requirements and exemptions applicable to Java applications.

Required:
To qualify for the Logo, a Java application, when running on a Windows PC, must use and re-distribute the Microsoft Win32 virtual machine for Java. For more information about using and re-distributing the Microsoft virtual machine refer to http://www.microsoft.com/java or the Microsoft SDK for Java.

Exemption:
Java applications are not required to be an OLE server or container.

Exemption:
Java applications are temporarily exempt from providing support for high contrast, from supporting full functionality through the keyboard, and from exposing the keyboard focus as outlined in the UI section.

Popular class libraries such as AFC 4.0 will provide support for these accessibility features in the future. Thus vendors should expect this temporary exemption to be removed from future versions of the logo program.

Required:
Java applications must follow all other Logo requirements that apply to any functionality they provide.

Examples:

? Java applications must provide automated installers and uninstallers that meet the requirements of the Install and Uninstall sections of this handbook. (Note that the VM is considered a core component and must not be uninstalled.)

? Java applications that use native data file types must register them and their icons.

7.6. Add-On Products

Add-on products are accessory products, such as wizards, templates, and macros, which are not executable files. If a wizard, template, or macro is always packaged *only* with the 32-bit product it supports, it is considered part of that product for Logo testing. This section applies only to add-on products that are packaged and marketed *separately* from the product they support.

Exception:
An add-on product does not have to be an executable (.exe).

Required:
To qualify for the Logo, add-ons must be used by a 32-bit product that is compliant with Designed for Windows NT and Windows 95 Logo. (This does not *require* the host product to license the Logo.)

Required:
If the separately marketed add-on products are als6 included in a suite of applications, they must be tested separately and with the suite for the add-on to qualify for the Designed for Windows NT and Windows 95 Logo.

Required:
Add-on products must follow all other Logo requirements that apply to any functionality they provide.

Examples:

? Any executable code in the add-on products must be 32-bit code.

? Add-on products must provide automated installers and uninstallers that follow the user interface rules as defined in UI/Shell.

? Add-on products that use native data file types must register them and their icons.

? Add-on products that use the mouse or keyboard, or that draw on the screen must support the Accessibility requirements.

8. Pretesting & Verification of Compliance

This section includes suggestions to help pretest your product for compliance with the Logo requirements. Be sure to review these suggestions prior to submitting your product for testing in order to assure the best results and avoid retests. Also, don't forget to review the "Top Ten Reasons Why Applications Fail Logo Testing on the First Try" on the web: http://www.veritest.com/nt95top10.htm

8.1. Stability & Functionality

The following are some of the tests we will run to test the stability & functionality of your product. You can pretest your product prior to submitting for Logo testing using the following.

ON WINDOWS 95

1. Find the Stress application in the Win32 SDK (\mstools\bin\win95\stress.exe), and select the following options: User Heap, GDI Heap, Wind32 Heap, Menu32 Heap, GDI32 Heap. Then set the Level to 2. Now execute the Stress program.
2. With Stress running, start your product from its icon.
3. Exercise all of your product's functions, and make sure they are stable even when the system is under stress.
4. Switch video modes "on the fly," and make sure your product is still operational.
5. Test printing and all other input/output functions supported.
6. Close Stress, and then start one popular 16-bit application and one popular 32-bit application to see whether your product will still run without problems while these are active.

ON WINDOWS NT

1. Start the Performance Monitor (Perfmon.exe).
2. Choose to view counters from "Memory," "Objects," and any other object-types that are relevant to your application.
3. Run your application for extended periods of time, exercising all of your product's functions, and ensure that all systematic changes in system resources are accounted for.
4. Switch video modes "on the fly," and make sure your product is still operational (not supported on Windows NT prior to version 4.0).

8.2. UI/Shell

Verification of High Contrast Support:
You can easily verify most of the settings related to size, color, and fonts. Click the **Display** tab on the Accessibility Options box in Control Panel. Choose Settings and verify that the "White on Black" option is selected. Apply these settings. Then go back into your product and test the major screens, dialog boxes, and controls to ensure that these changes have been properly reflected. The application should still be usable. Screen elements should be displayed correctly and be compatible with the appearance scheme. For example, make sure that text contrasts with its background, that graphics on toolbar buttons are distinguishable, and that menus and scroll bars have the appropriate sizes. You can also experiment with appearance schemes by clicking the **Appearance** tab on the Display box in Control Panel.

Verification of Exposure of Keyboard Focus:
Use the screen magnification accessory that is included with the Active Accessibility SDK. The magnifier must accurately track the location of the keyboard focus when the focus is moved. Check to ensure that it tracks the focus in both documents and toolbars. This will also be included with future versions of the operating system.

8.3. UNC & Long File Name

To test direct network browsing with UNC paths: Open a file, and use the **Save As** command to save it with a Long File Name (LFN) and UNC paths (for example, \\ServerName\MySubdirectory\MyLongFileName) to a standard server. It should be possible to save and retrieve files without specifically referencing a network drive letter.

Here is how to test whether your product properly handles LFNs. Basically, LFNs must do the following:

? Allow plus signs, commas, semicolons, equal signs, and square brackets anywhere.

? Not save leading or trailing spaces. (You can test for this by removing the name inserted into the Save As dialog box and typing "###test###" or similar text. The program should strip the spaces and add an extension, returning the file name "test.ext".)

? Not save question marks.

? Support 255 characters (including the path and extension).

? Save to a UNC path, such as \\Server\Directory\Filename.

? You should test each of the allowed file names in the following list. When your application saves each file, it should add an extension and save it to the hard disk.For example, "test." will save as "test..ext".

*() test

- *1 test#test#test#test
- *2 test#1234567890[on through 260]

? You should also test the following list of file names, which should save to the hard disk as indicated:

- *0 test (saved as "test.ext")
- *1 ###test (saved as "test.ext")
- *2 test### (saved as "test.ext")
- *3 test#;#+#,#=#[#] (saved as "test#;#+#,#=#[#].ext")
- *4 \\folder#one\folder#two\folder#three\folder#four\file

LFN Verification for Utilities:

Run the utility (that is, your product), and manipulate several LFN files. Open these files with their LFN and with their 8.3 aliases, and make sure they still relate to the same file.

Check overall LFN file structure to make sure its integrity has not been damaged.

8.4. ACPI/OnNow

Productivity applications can verify compliance with ACPI/OnNow *recommendations* with the following tests:

1. Open a file on a local hard drive, edit it (don't save), put the machine to sleep, wake the machine, and verify that the application continues to run, the file can be saved, etc.

2. Open a file on a hard drive in a docking station, edit it (don't save), put the machine to sleep, undock the machine, wake the machine, and verify that the application correctly informs the user of the problem and leads them through steps to correct the problem.

3. Open a file on the network, edit it (don't save), put the machine to sleep, wait a few minutes for the network connection to time out, wake the machine, and verify that the application continues to run, the file can be saved, etc.

Games can verify these recommendations with the following tests:

1. While in game play, put the computer to sleep via the Sleep button or Start Menu.

2. Verify the computer goes to sleep and sound stops playing.

3. Wake the computer.

4. Verify the game play remains paused. After resuming game play, the game continues to function normally (graphics, sound, and input devices continue to function as before).

Event handling applications can verify these recommendations with the following tests:

1. In the Power control panel, set the system idle time as low as possible.
2. Start the application and its event handling feature.
3. Put the computer to sleep using the computer's Sleep button or using the Start Menu.
4. Start an event that will take longer to process than the system idle time.
5. Verify that the computer does not go to sleep while the event is being processed, and after the event has been processed, the computer goes to sleep immediately.

Tip: Applications that need the computer to wake up quickly to respond to an event can RequestWakeupLatency to tell the OS to configure the computer to wake as quickly as the computer allows.

Tip: Applications that need to wake the computer at a certain time can do so using the CreateWaitableTimer and associated APIs.

Tip: Messaging applications can display the number of new messages on the computer's message waiting indicator panel by calling the SetMessageWaitingIndicator API.

Presentation applications can verify these recommendations with the following tests:

1. In the Power control panel, set the system idle time as low as possible.
2. Start the application and play a presentation.
3. Wait for a time longer than the system idle time and verify that the computer does not go to sleep and that the display does not blank during that time.