The Microsoft® Windows® Guidelines for Accessible Software Design

Creating Usable Applications for Everyone

Comment [MJH1]: 6/9/99 This version is refigured to address the needs of developers and testers. I feel we should have another version with a general audience focus.

Comment [MJH2]: 6/7/99 I removed the "Concise Summary of Recommendations" to keep it simple and un-presupposing. Again, this version is targeted solely for developers and testers.

Comment [MJH3]: 6/7/99 This tracks to our new slogan, seems less whiny and sounds more eloquent.

Please direct comments and suggestions to:
Martin Higgins, Writer/Editor - a-martih@microsoft.com

Microsoft Accessibility and Disabilities Group - *Technology for Everyone*
One Microsoft Way
Redmond, WA 98052-6399
http://microsoft.com/enable/

Contents

Section 1 - Introduction

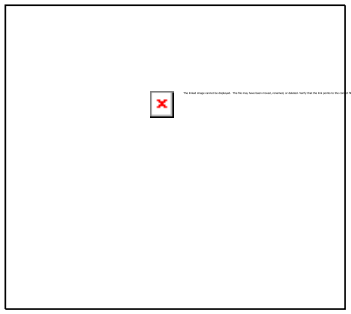

Figure 1 - Illustration of Computer user at the keyboard

Personal computers are powerful tools that help people work, create, and communicate in ways that would otherwise be difficult or impossible. But the goal of simplifying computers for everyone can only be realized when *all* people have access to the power of personal computing.

Microsoft® Windows® and Windows NT® have features and enhancements to make computers accessible to people with disabilities, but to make home, school, and workplace computers usable for everyone, applications must be designed to support these options.

This document explains:

- How to compensate for various limitations.
- How computers can be modified to meet accessibility requirements.
- How to design and develop accessible computer software.
- How to test, document, package, and support accessible products.

The Microsoft® Windows® Guidelines for Accessible Software Design was created to remove design flaws that prevent people from using applications – not as special considerations for special groups, but rather, "Technology for Everyone" which literally means *options for all*.

See http://microsoft.com/enable/ for more information.

Although prepared for software application designers, builders, testers, and documentation writers for the Windows 9*x*, Windows 2000, and Windows NT operating systems, this document provides valuable insights for all IT

- Current Accessibility Information - Disabilities, accessibility aids, and design sensibility
- Detailed Design Guidelines - How to put these guidelines into practice. Features and programming techniques that make applications more accessible and compatible with accessibility aids
- Summary of Requirements and Recommendations –Windows Logo requirements and design recommendations
- Testing Techniques – Guidelines for testing accessible products
- Quick reference appendices:

    A. Additional Resources - Accessibility aids and manufacturers

Section 2 Current Accessibility Information

This section discusses:

- Reasons for Supporting Accessibility
- Disability and Accessibility
- Disability Categories
- Types of Accessibility Aids
- How To Make Computers and Aplications Accessible

Reasons for Supporting Accessibility

Creating accessible applications is the right thing to do, from both a cogent, comprehensive design sensibility, to simply addressing the *entire* market. When everyone can live, work, study, play and create independently, we all benefit enormously from their contributions.

Accessible design is also important to the computer industry because:

- Accessible products are needed by over 30 million Americans with disabilities, plus their household members, friends and relatives.
- Accessible products are necessary for businesses, government agencies, schools, and federally funded organizations to comply with:
  - The **Americans with Disabilities Act (ADA)**
  - The **Rehabilitation Act** (sections 508 and 504)
  - The **Assistive Technology Act of 1998** (**ATA)**
  - The **Individuals with Disabilities Education Act of 1997 (IDEA'97)**
  - **Human Factors and Ergonomics Society (HFES)** and **American National Standards Institute (ANSI)** guidelines
  - The United Kingdom's **Disability Discrimination Act** and pending European Community **ISO 9000** legislation
- Accessible design enables applications to qualify for the "Designed for Windows and Windows NT" Logo. The logo handbook currently lists five requirements designed to make applications more accessible and includes strong recommendations, which will become requirements in the future.
- Accessible applications enable employers to retain valuable employees if they become temporarily or permanently disabled, or develop functional limitations as a natural part of aging.
- Accessibility aids require the same information used by sophisticated automation tools, such as:
  - Testing tools
  - Task automation tools (intelligent agents)
  - New input methods (voice input)
- The FCC is currently evaluating the **United States Telecommunications Act** to determine whether to mandate certain levels of accessibility for all commercial materials distributed by telephone and over the Internet.

Comment [GCL4]: Page: 3
12/29/98: Compare this with the sections from the LEA Chapter and the Logo handbook.

Disability Categories

Disabilities are divided into the following general categories:

- **Visual Impairments**
- **Hearing Impairments**
- **Mobility Impairments**

- **Speech Impairments**
- **Seizure Disorders**
- **Cognitive and Language Impairments**

These categories describe disability groups covering a wide range of people with distinctly different levels of individual need.

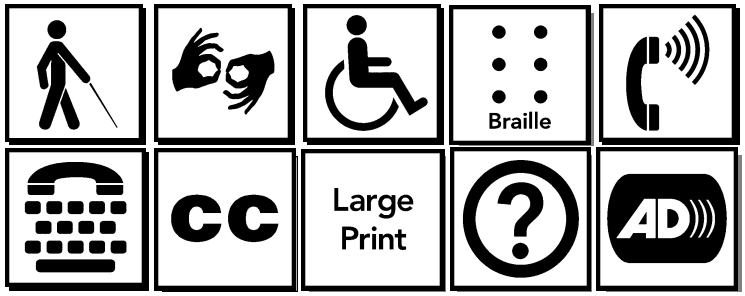

Figure 2 – Ten examples of accessibility graphic signage

Visual Impairments

Visual impairments range from minimally reduced visual acuity to total blindness. Millions of people have slightly impaired vision, finding it difficult to read small print or black text on a gray background or experience eyestrain during long computing sessions. These inconveniences are usually not considered disabilities, but applications that compensate for this type of user need makes computing less demanding and more accessible for all.

Corrected vision beyond 20/80 is considered low vision, a condition that requires larger than normal text and higher foreground text/background contrast. When corrected vision cannot rely exclusively on visual information — approximately 20/200 or higher — it is considered blindness, requiring all displayed information to be converted into articulated text or Braille.

Other visual impairments include:

- Reduced field of vision, where focus is limited to a small area
- Color blindness, where some color combinations are difficult or impossible to identify. This condition affects approximately 11% of the population.

Hearing Impairments

Users who are unable to hear, identify various sounds, or recognize spoken words require an alternative prompting method.

This problem includes people:

- Working in a *noisy* environment (busy office, vehicle, aircraft, factory)
- Working in a noise *sensitive* environment where sounds can disturb other people (library, classroom, courtroom, hospital)
- Using a computer with a broken or missing soundcard or loudspeaker.

Mobility Impairments

Some users are unable to perform manual tasks such as:

- Moving a mouse or other pointing device
- Pressing two keys simultaneously
- Pressing one key at a time
- Pressing and holding down a key
- Holding a book open

Many users need keyboards and mouse functions to be adapted to their requirements, or rely exclusively on a single input device.

Cognitive And Language Impairments

Cognitive and language impairments take several forms:

- Short- and long-term memory loss – Amnesia (caused by disease, trauma or intense emotional episodes)
- Perception anomalies – inability to recognize colors or images
- Developmental disabilities – Alzheimer's Disease, Down Syndrome, Aging
- Language impairments – dyslexia, aphasia, illiteracy or using software written in an unfamiliar language

Well-designed software can help people with mild cognitive or language impairments use computers and can also help children and illiterate adults learn to read.

Seizure Disorders

Epileptic seizures can be triggered by:

- Flashing lights or images – warnings, decorative elements
- Rapidly changing images – animations and reduced frame rate video
- Random or repetitive sounds – alert sounds and notifications

Speech Impairments

Computers aid people who have difficulty speaking by allowing them to communicate through electronic mail, postings and online chats, and by articulating typed or selected text with synthesized speech.

Given the current state of technology, speech difficulties do not usually affect a user's ability to operate a computer, but it may pose a problem when using telecommunications and voice menus. Speech limitations may also present a challenge when voice recognition becomes a prevalent input method.

Types of Accessibility Aids

The following sections describe common types of accessibility aids — utilities added to a computer to make it more accessible to people with certain disabilities:

- Screen enlargement utilities
- Screen review utilities
- Voice input utilities
- On-screen keyboards
- Keyboard filters

Not all people with disabilities use tools like these, but they are crucial for those who must. Utility effectiveness depends on compatible application software.

Many of these utilities are available as free demonstration software on the World Wide Web. See http://www.microsoft.com/enable/ for an accessibility aids catalog.

Screen enlargement utilities

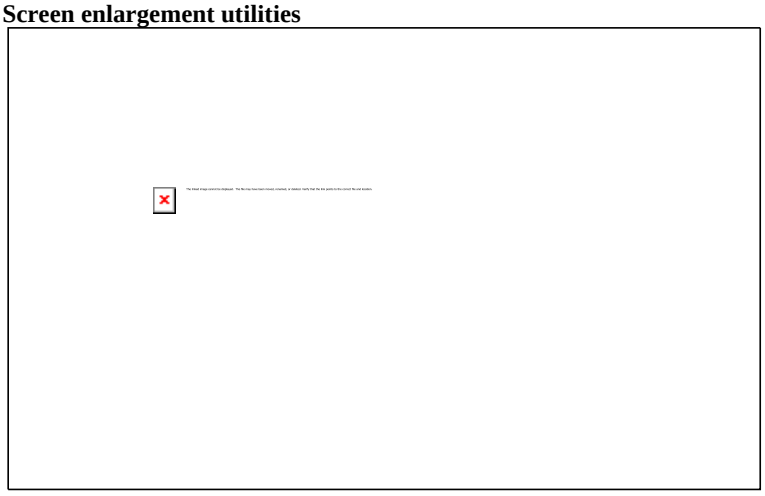

Figure 3 – Image of a window and the same window enlarged by the Windows Magnifier accessory.

Screen enlarger utilities – also known as *screen magnifiers* or *large print programs* – allow users to magnify a portion of the screen by converting the monitor into a window that shows an enlarged virtual display. Enlargers must track where the user is working to automatically display the active area. Users have the choice of moving the window with mouse or keyboard commands to view off-screen areas.

Comment [MJH5]:

Screen review utilities

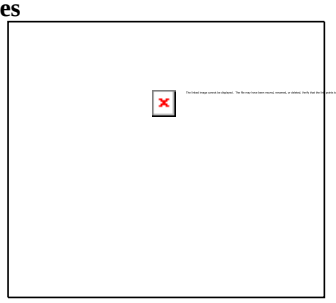

Figure 4 - Illustration of computer screen reader announcing, "Press enter to begin."

Users who are blind or have difficulty reading rely on screen review utilities – also known as blind-access utilities or screen readers. Screen review utilities present visually displayed screen information through alternative, non-visual media, such as synthesized speech or a refreshable Braille display. These methods are text driven, so the utility must render graphic screen elements as text. The utility must also track focus location and the current operation to describe important aspects of what is on the screen.

Blind-access utilities announce when the keyboard focus moves to a new word, control, or window and other on-screen changes automatically. Users can also choose to have the utility articulate other on-screen information.

Example: Keying a shortcut that prompts the utility to announce where the user is working on-screen, such as, "Cancel push button, Open dialog box, Excel".

Users with dyslexia or other reading difficulties use similar tools – also known as *reading aids* – to articulate individual words or documents.

Within Windows, these utilities collect information through the Win32® API and through the application, when provided, using Microsoft Active Accessibility. As a back-up verification system, some utilities infer contextual relationships between elements by monitoring all operations that draw information to the screen.

Voice input utilities

Users with profound mobility impairments often use voice input utilities – also referred to as *speech recognition programs* – to control the computer by voice command rather than with a mouse or keyboard. Most voice input users do not have disabilities per se, but use the utility for multitasking or when manual input is inconvenient.

Example: Hands-free text input while manually searching through print documents.

As with screen review utilities, voice input utilities must identify and manipulate on-screen objects, so users can activate commands by speaking a few commands.

On-screen keyboards

When keyboard entry is not an option, users often employ alternative input methods, such as switches or pointing devices. On-screen keyboard utilities display command lists and allow command execution through several input methods. These aids display a "point and click" key set that can choose commands, activate objects, or "type" text.

Some utilities also allow users to choose commands using switches. A single-switch system displays an on-screen keyboard that successively highlights groups of commands until the user selects a group by closing a switch. The utility then continues to highlight sub-groups until the user selects a single command. If the user can point but not click – as when using a head pointer – the command can be activated by pausing over the desired choice for a pre-set timing period.

(Brief description of a multiple switch system needed here?)

Comment [MJH6]: 6/8/99 We mention both then describe one in detail. Either we should drop the multiple switch reference, describe it or shorten the single switch description.

On-screen keyboards are commonly used to display all available command buttons. To display them appropriately, the utility must identify, name, and activate controls as a voice input utility would.

Keyboard filters

Users with impaired dexterity may have difficulty using an unmodified keyboard.

Comment [MJH7]: 6/8/99 Is this artwork necessary?

Windows operating system keyboard filters can compensate for erratic motion, tremors, slow response time, and related input issues, but these compensations are more difficult to assign to pointing devices (mouse, trackball). Users with impaired dexterity usually rely on keyboard input.

For more information about these options, see the Accessibility Options section in the Windows Control Panel.

Other types of keyboard filters include typing aids (word prediction, abbreviation or expansion utilities) and add-on spelling/grammar checkers. These utilities benefit a growing group of non-disability users by improving typing speed, maintaining word accuracy, and ensuring grammatical clarity to documents.

How To Make Computers and Applications Accessible

Accessibility is achieved by designing computer software to accommodate the widest range of users, *including* those with disabilities. A few simple options greatly increase the number of people who can use an application.

Special needs are addressed in the following ways:

- **New features** can be built into hardware and operating systems, making them more accessible for users with *and without* specialized needs. This is the preferred solution, ensuring that features will be available on all workstations and used with all accessibility-compliant applications.
- **Accessibility aids** can enhance the hardware and operating system, making them usable for more people. These aids work with most applications, but they must be moved or duplicated when the user works at a different computer and are often expensive. Unfortunately, many people who could benefit from such aids never acquire them.
- **Specialized applications**, such as word processors, can be optimized for people with specific disabilities. While these specialized applications are available, people with disabilities need to use *mainstream* applications to ensure system integration with co-workers and frequent feature upgrades.

  Example: Internet browsers designed to read pages for people who are blind and word processors that integrate voice and text for people with cognitive impairments or limited literacy skills.

- **Usability features** can be built into mainstream applications, making them easier to use for people with disabilities.

  Example: Customizable colors and access keys provided exclusively by the application.

These features also benefit people who *do not* have disabilities by offering usability options.

This document addresses two areas that determine application accessibility:

- **End-user features** to make an application more usable
- **Programming techniques** for application compatibility with accessibility aids

Section 3 - Design Guidelines

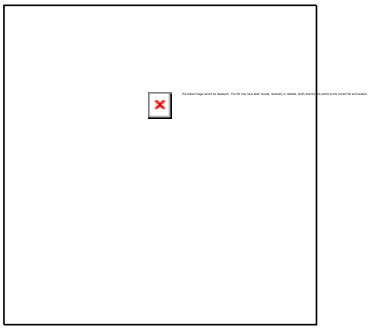

Figure 5 - Illustration of designer at drafting table

The following guideline discussions, requirements and recommendations are displayed with quick reference priority (★ = low, ★★★★ = high), Logo requirement status, and category headings:

Comment [MJH8]: 6/8/99 These decorative embellishments move the document toward a "user" presentation style rather than a crisp developer/tester format.

- Complete Keyboard Access
- Exposing the Keyboard Focus
- Exposing Screen Elements
- Color
- Size
- Sound
- Timings
- Unexpected Side Effects
- Mouse Input
- Customizable User Interface
- Layout

Complete Keyboard Access

Provide keyboard access to all features.	★★★★ **Logo Requirement** Blindness Dexterity

A well-designed keyboard interface is an important component of software accessibility because it serves users with a wide range of disabilities and those who simply prefer keyboard input. All applications should be designed so that they can be operated entirely by keyboard input. If mouse input is offered, it must be optional.

This section covers:

- Choose a Keyboard Interface
- Keyboard Access Design Exceptions

- Document the Keyboard Interface
- Underline Access Keys

Choose a Keyboard Interface

Effective keyboard interface design adapts a current, familiar model where interface keystrokes are intuitive where compatible with existing applications or controls.

The Microsoft Windows Keyboard Guide, available at: http://www.microsoft.com/enable/training/keyboard.htm , contains keyboard user interface (UI) standards and recommendations for Windows-based applications. This reference material identifies specific keystroke combinations and is a valuable resource when designing any keyboard UI.

In cases where dialog boxes become overly complex and unique access keys cannot be assigned, favor a "common sense" solution.

The following examples show how to move and resize objects using keyboard inputs, and navigational techniques that can be adopted or adapted to a new application:

- **Menus and Dialog Boxes – (WINAPP26)** Menus are one of the most common and standardized user interface elements. Putting commands on the menu is always a safe option, but avoid large and unwieldy menus by providing a default configuration that hides command menu items until the user requests them. commands to a menu is a simple accessibility solution, but designers should

  Menu items should be explicitly requested through an option in the application or when users set the Keyboard Preference flag in Control Panel. (Keyboard Preference is discussed later in this document.) Supporting this option can provide dialog boxes and property sheets with the functionality of mouse input.
- **Property Sheets –** A user can directly manipulate a control to adjust its size and location by using the objects property sheet. This preferable, non-modal method offers a tighter feedback loop, simplifying command adjustment. Microsoft's Visual Basic® program uses this method.

See: *The Microsoft Windows Keyboard Guide* for keyboard UI standards and tips.

GRAPHIC OF PROPERTY SHEET

- **Move and Size Commands.** Most system menus support Move and Size operations using keyboard commands. These commands and the UI can be used for application-specific objects on the normal menu bar or the object's context menu.

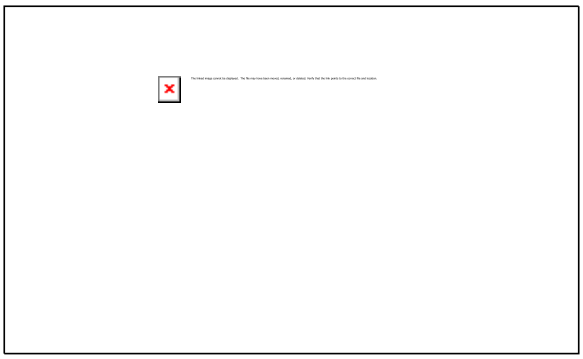

Figure 7 - Move and Size commands on control menu (detail)

- **Navigating Between Objects -** Windows Help enables the TAB key to move through a list of active regions or "hot spots", and the ENTER key to select an active region. The SHIFT+TAB key combination is used to move backwards through the list and the ARROW keys control directional movement (especially useful with two dimensional presentations). This application design allows users to move to the next active region by typing one or more label letters, similar to moving through standard list box entries.
- **Navigating Between Window Areas -** Some applications divide a window into two or more panes and allow users to shift between them with a simple keystroke. Windows Explorer can display three panes in a single window: tree view, folder, and tool bar. The TAB or the SHIFT+TAB key combination moves focus from pane to pane and the arrow keys move the focus within each pane.
- **Multiple Document Interface (MDI)** - The F6 key and the CTRL+F6 key combination moves focus between panes and child windows, and allows users to move through a list of windows using the Windows menu. Other applications use the CTRL+TAB or CTRL+PAGE DOWN key combination to shift through groups of panes or pages.

Comment [MJH9]: I am unsure what to do with the exercises. Perhaps there should be an appendix that contains exercises, test questions and a developer/tester "cheat sheet" containing a one-page list of critical design considerations.

Comment [MJH10]: 6/8/99 I'm not sure what Windows Help has to do with navigating between objects. Plus, I don't find a recommendation or suggestion to the designer here.

Comment [MJH11]: 6/8/99 No clear recommendation or suggestion. Either one of those plus an example would make all this much clearer.

Keyboard Access Design Exceptions

Design an efficient, comprehensive keyboard interface that provides access to all application features unless it is:

- Exceedingly difficult to use
- Infrequently used
- Too challenging for application designers to implement

Again, favor logic and common sense solutions when dialog boxes become overly complex and unique access keys cannot be assigned.

Users have the option of reverting to tools that simulate mouse input – using the keyboard or other input mechanisms – but these alternatives are not a substitute for good keyboard interface design.

Example: A simple drag and drop operation requiring 40 keystrokes might make an application accessible but not usable or user-friendly. Blind users (in fact, *all users*) would find performing such an inherently visual operation with keystrokes a difficult and frustrating process.

Document the Keyboard Interface

Fully document the keyboard UI. ▪ Windows conventions need no documentation.	★★★★ **Logo Requirement** Blindness / Dexterity

Without documentation, an application's keyboard commands and mechanisms are useless, so all logo applications must include complete, comprehensive documentation for the keyboard UI. If space is unavailable in primary documentation, keyboard interface information may be described elsewhere and cross-referenced.

Keyboard access should never be categorized as a niche or specialty feature because many users prefer it. It is unnecessary to document elements that follow accepted Windows conventions, such as standard menus and controls.

Comment [MJH12]: 6/9/99 Is there an accepted definition of standard when used in this context?

Underline Access Keys

Provide underlined access keys for all menus and controls. ▪ Required when Keyboard Preference option is True. ▪ Not required when names change dynamically or no unique characters remain unused.	★★ **Future Logo Requirement** Blindness / Dexterity

All menu items and controls should have underlined access letters. The following illustration shows access keys on a File menu.

Comment [MJH13]: 6/9/99 Are Access Keys preferable to Shortcut Keys? From a programming standpoint, there is a difference between Access Keys and Keyboard Accelerators (Shortcut Keys). An explanation of these differences might be valuable here.

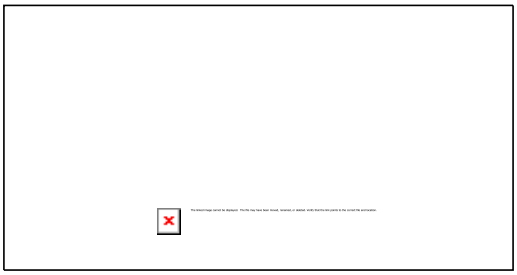

Figure 8 Calling out underlined access keys on a typical File menu.

Users are more likely to use access keys to speed up commonly performed operations when familiar with an application. This adaptation is even more prevalent among users who do not use a mouse, such as those who are blind or have low vision.

Example: Screen review utilities present the elements of the UI sequentially. Users must press the TAB key to read or hear each element before deciding whether to press the TAB key again. Access keys speed this process.

Comment [MJH14]: 6/9/99 Do all screen review utilities use the TAB key for this purpose?

Access keys may be omitted in two situations:

- **When there are no unused label characters.** Either rename the control or omit the access key if the command is not used often.

Example: The standard Sort Options dialog box has identical controls for three sort criteria, making it difficult to assign unique access keys.

- **When users cannot predict a dynamic command set.** The standard OK, Cancel, and Apply buttons found on all property sheets must not conflict with controls on a particular page.

  Example: A context menu whose commands might vary. Context menu commands can use non-conflicting access keys, but they may be omitted if they might conflict with each other or when users might expect a specific letter to be associated with a different command.

When providing access keys for dialog box controls, designers must ensure a static text control immediately precedes the object it labels in the tab order. This ordering helps screen review utilities determine the relationship between the control and its object.

Comment [MJH15]: 6/9/99 Shall we add a screenshot of the Sort Options dialog box here to drive this point home. Until I opened it myself, I didn't get the full impact of the recommendation.

Comment [MJH16]: 6/9/99 This whole paragraph is the most important point in the section. Perhaps we should highlight it, embolden it or box it. Also, does the ordering "help" or is it essential?

Exposing Keyboard Focus

Expose the keyboard focus location within a window ▪ Automatic for objects that are separate windows ▪ Within a window, use Active Accessibility or cover an object's bounding rectangle with the system caret.	★★★★ **Logo Requirement** Blindness / Low Vision Dexterity / Cognitive Language

This section covers:

- Using Standard Windows and Controls
- Using Active Accessibility to Expose the Focus
- Using the System Caret to Expose the Focus
- Examples of Keyboard Focus Location
- Verification

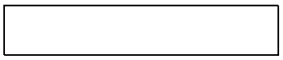

Figure 9 Yes and No command buttons with focus rectangle on Yes button

Many accessibility aids must identify the "focus point"– the area where the user is currently working on the screen.

Comment [MJH17]: 6/9/99 "Focus point" is such an important issue, there should be a more comprehensive definition here.

Example: Blind-access utilities describe active text or objects, screen-magnification utilities pan to include them, enlarging that portion of the screen. The utility rearranges multiple windows to avoid covering the focus point.

When the operating system indicates a focus move to a window, menu, or standard control, the accessibility aid can determine this location. But

when an application uses a different method of indicating the visual focus *within its window*, such as highlighting an icon, a windowless custom control or a cell in a spreadsheet, it is more difficult to determine the location.

To be accessible, an application must use Active Accessibility or move the system caret to make its focus location available to other programs in the system.

Active Accessibility and moving the system caret are described in detail below.

Using Standard Windows and Controls

The Windows operating system uses Active Accessibility, Windows hooks, and window messages to inform accessibility aids which window has the keyboard focus. No additional work is required for standard Windows controls or any object that occupies an entire window.

Comment [MJH18]: 6/9/99 Again, a main point that should be italicized, emboldened or boxed. Your choice.

Using Active Accessibility to Expose the Focus

Microsoft Active Accessibility is the preferred method of exposing the location of the keyboard focus for custom controls.

Comment [MJH19]: 6/10/99 What is involved in using AA to expose focus. Do we need an overview here?

More information about Active Accessibility is available at http://microsoft.com/enable/ or from the Microsoft Developers Network at http://msdn/

Applications must:

- Call **NotifyWinEvent** when the focus moves to an object that is not an entire window.
- Handle the **WM_GETOBJECT** message when used to query the focus object.
- For custom user interface elements that are exposed with **Iaccessible** interface, be sure to support the **Iaccessible::get_accFocus** property.

Using the System Caret to Expose the Focus

The system caret is the blinking vertical bar users see when editing text. It can be placed anywhere on the screen, in any shape or size, visible or invisible. When invisible, it can indicate the focus location to applications without obscuring onscreen information.

To make the system caret invisible, call the **CreateCaret** function to set the caret's size and shape and the **SetCaretPos** function to move it to the visual focus indicator (highlighted cell, icon, button, etc.). The caret is present, but invisible unless you explicitly make it visible.

Comment [MJH20]: 6/9/99 A brief description of how to make the caret visible or invisible needed here.

When the focus is moved to a different sized object, the application should call **DestroyCaret**, then **CreateCaret** to specify the new object size. The system caret must cover the screen element's bounding rectangle, so screen magnification tools can zoom-in on it.

Applications should display focus and selection indicators only when they are in the active window. When the window is no longer active, the application should remove the visual indicator and call the **DestroyCaret** function. While this step is not always necessary for Win32 applications, it is a good standard practice.

Comment [MJH21]: 6/9/99 Should we define or elaborate upon selection indicators here?

Comment [MJH22]: 6/9/99 As we found out a couple of weeks ago, DestroyCaret is triggered automatically and the problems encountered in the past have been fixed in subsequent versions. Is it still good practice? If so, why?

Examples of Keyboard Focus Location

Occasionally it may be difficult to determine the focus location. Extended and discontiguous selection often confuses this designation, but keyboard focus location should be considered independent of the selection, even though an application normally links them.

The following examples explain this distinction by identifying and indicating the correct keyboard focus location in an application:

- **Insertion bars in text –** When users move an insertion point within text, it is usually drawn with a visible system caret. When the application draws its own insertion point, it should still move the system caret invisibly, tracking the visible insertion bar's location and size.

Figure 10 - Text with a flashing insertion bar.

- **Extended text selection –** When a user makes an extended selection, one end of the selection is the "active" or moving end. This represents the definitive keyboard focus location.

  Example: To select three characters, start with the insertion bar in an edit control, hold the SHIFT key down and press the right arrow three times. The start point is the stationary end or the "anchor". To the right, the flashing caret marks the active end of the selection. By holding the SHIFT key and pressing another arrow key, the active end moves, indicating where the system caret should be placed.

  Always display a visible insertion bar at the selection's active end to provide users with its location.

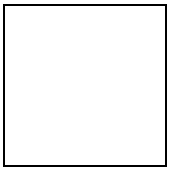

Figure 11 - Flashing insertion bar at active end of character selection (how is this shown?)

- **On graphic objects –** When users move the keyboard focus to a graphic object, such as an icon or bitmap, applications should position the system caret invisibly over that object so the caret's rectangle covers the entire image and its label.

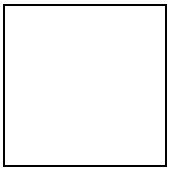

Figure 4 - Icon with keyboard focus and selected Icon with text labels.

- **Within graphic objects –** Occasionally, a single bitmap represents several objects, such as a group of graphical buttons. Applications indicate focus by highlighting a portion of the bitmap, drawing a dotted

rectangle over it or moving the mouse pointer. In addition to indicating focus, the application should place the system caret invisibly over the bitmap's "hot spot" or the object being referenced.

- **Simple custom controls –** When an application draws simple custom controls, such as a custom push button, the keyboard focus is associated with the entire control. The system caret should cover that control completely.

  Note: Explicitly setting the caret location is necessary for windowless custom controls only. Controls that are windows automatically take the keyboard focus.

- **Complex custom controls –** A complex or composite control, such as a list box, can position the focus on individual elements within a larger control. When doing this, an application should indicate the element's focus area with the system caret. Applications may recognize a collection of items as a single control, but each item should be considered a separate control element. The text for each element should be drawn using the standard Windows function calls – such as **ExtTextOut** – which can be seen by screen review utilities.

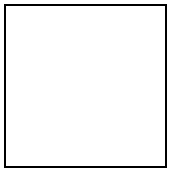

Figure 5 - Drop-down list box with dotted rectangle keyboard focus on one item.

- **Spreadsheets.** Within a spreadsheet, the focus is usually placed on a cell, rather than its content, as indicated by an emboldened cell border. Applications should place the system caret over the entire cell. When cell content editing begins, the application should indicate the content text or graphics focus appropriately.

- **Discontiguous selection.** Discontiguous selection is usually supported among discrete items, rather than in text. One item always has the keyboard focus or was most recently selected by a mouse click. This object should be covered by the system caret.

  Example: Select an item in a folder or in Windows Explorer, hold down the CTRL key and use the arrow keys to move the focus rectangle to a file that is not part of the selection.

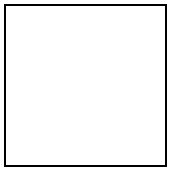

Figure 14 - Upper icon selected and highlighted with the focus color. Lower icon has the keyboard focus as indicated by dotted focus rectangle.

- **Mouse-only objects.** Although applications should provide keyboard access for all functionality, some objects can only be manipulated or selected using the mouse. In this case, regard an object, when selected, as if it had received the keyboard focus and position the system caret appropriately. Of course, when the actual keyboard focus moves, the application must follow it, because the user's attention has moved away from the mouse-only object.

Applications should:

- **Use standard Windows functions to copy or erase text and graphics –** These include **BitBlt**, **PatBlt**, and **StretchBlt**. Use these functions whether drawing to a screen or to an off-screen bitmap, because utilities track text and graphics from the time drawn until copied to the screen.
- **Avoid directly manipulating bitmaps –** If an application directly manipulates the memory associated with a device context (DC) and bypasses Windows functions, screen review utilities will not recognize bitmap changes.

  Example: If an application draws text into a bitmap using a Windows function call, then erases it by clearing the bitmap memory, the screen review utility will assume the text is still present. If the bitmap is used again for another operation, the text might be read even though it is no longer visible to the user. Similarly, information copied from one bitmap to another without using the Windows functions may be displayed visually but unseen by the screen review utility.
- **Avoid directly modifying the screen –** The Windows application programming interface (API) provides several tools for manipulating bitmap or display pixels directly, such as DirectDraw, Display Control Interface (DCI), WinG **(DOES THIS REFERENCE NEED AN UPDATE?)**, and the **CreateDIBSection** function.

Comment [MJH23]: 6/9/99 More importantly, this seems to say not to use these tools? Is that the case? My lack of tech background is showing here. This is amibiguous.

Verification

Use the Active Accessibility SDK Magnifier accessory to verify that your application responds to keyboard commands by displaying the focus area in the Magnifier window. The Magnifier will be a standard accessory in future versions of Windows and Windows NT.

The Active Accessibility SDK is available at http://microsoft.com/enable/.

Exposing Screen Elements

Allow other software to identify and manipulate all user interface elements. ▪ Standard window classes and controls support this requirement automatically, when used correctly. ▪ Add additional support only when creating custom window classes or controls, or drawing content in the window client area. ▪ Name every object, window, and graphic properly.	★★★★ **Future Logo Requirement** Blindness / Dexterity Cognitive

This section covers:

- Using Controls
- Naming Controls
- Naming Windows
- Identifying Windows By Function
- Identifying Images And Bitmapped Text
- Labeling Objects Clearly
- Drawing To The Screen
- Verification

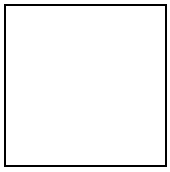

Figure 15 - The Inspect tool displaying the properties of a command button.

Many accessibility aids must identify and manipulate on-screen objects, discrete objects (windows, controls, and menus) and document elements (words, paragraphs, images, worksheets, and tables). Users who are blind rely on screen review utilities to describe on-screen objects verbally. These utilities articulate the name, type, and state of each control.

Comment [MJH24]: 6/9/99 We need a better differentiation between on-screen objects and discrete objects.

Example: When a user tabs to a check box, a screen review utility might respond with "Quick Printing check box, checked." Voice input utilities and on-screen keyboards have similar functions: to identify and name specific controls and to manipulate that control in response to user commands.

Accessibility aids collect information about a control by evaluating its window class. Standard controls support Active Accessibility, which exposes all relevant information through a single standard interface. Non-standard controls can render an application unusable for users who rely on accessibility aids unless the application supports active accessibility.

Screen review utilities must also assess document content before reading it aloud or describing where the user is working within the document.

Example: Sighted users can see spreadsheet headings (in rows and columns) to identify the active cell. But a blind-access utility would need to respond to "C5" because the cell functions as a custom control embedded in a table, which functions as a complex custom control.

Comment [MJH25]: What are we advising apps to do with doc content. MSAA is not very helpful (at least until MSAA is associated with doc content).

Using Controls

The following sections provide more information about working with these screen elements:

- **Standard Windows controls –** including Windows Common Controls
- **Superclassed or Subclassed standard controls –** a standard Windows control with customized behavior

Comment [MJH26]: 6/10/99 Is this a more correct or incorrect addition?

- **Owner-drawn controls –** operate like a standard control with customized appearance
- **Owner-drawn menus –** normal menu behavior with customized appearance
- **Custom controls –** implemented by an application without using the normal Windows mechanisms (Visual Basic, ActiveX, and Java controls)
- **Text control -** in controls and document areas

Standard Windows Controls

Use standard Windows controls and those provided by the Windows Common Controls library whenever possible. Both are fully compatible with accessibility aids.

Each standard Windows control is a separate window of a specific class, so the accessibility aid can be notified when the focus moves to a new control. The aid determines the control's window class and designates which additional control messages can be sent to query or alter its state. Aids can also identify all child controls contained within a parent window and also identify the parent window of any control.

Some Common Controls (list view, tree view controls, etc.) are flexible and can replace many custom and owner-drawn controls. Example: A list view can replace an owner-drawn list box to present a check box next to each item. The new, enhanced button control can display images and text, which was a common reason for using custom controls.

Comment [MJH27]: Does this refer to the checkbox in the list view?

Owner-Drawn Controls

Owner-drawn controls can be accessible if used properly. Although owner-drawn controls operate like standard controls, they may have a customized appearance. Some applications use custom controls to change its appearance, but owner-drawn controls are a more accessible solution.

Example: An application can use an owner-drawn control to display an X in a selected check box rather than a "checkmark" , or label a command button with a picture instead of a word. An owner-drawn control using a standard Windows control appears normal to accessibility aids and supports Active Accessibility, while presenting a customized appearance.

Comment [MJH28]: 6/10/99 "checkmark" and "X" seem to have been reversed/

Define all owner-drawn control labels, even when it will not be visible on the screen. Owner-drawn controls with a "unreadable" normal captions (graphic faced button) will be unidentifiable to the accessibility aid if its caption is a blank string. Ensure that owner-drawn controls can respond to all class-specific text retrieval messages, (CB_GETLBTEXT, LB_GETTEXT). Set the appropriate style bits (CBS_HASSTRINGS, LBS_HASSTRINGS) to indicate the owner-drawn control supports those messages.

Comment [MJH29]: 6/10/99 I don't find a CB_GET**KB**TEXT listed. Could this be actually be CB_GET**LB**TEXT?

Comment [MJH30]: 6/10/99 Add this here?

For more information about exposing owner-drawn controls, see the Active Accessibility documentation.

Comment [MJH31]: 6/10/99 Same question here as above.

Superclassed and Subclassed Controls

Some applications alter standard control behavior with techniques known as superclassing and subclassing. An application with superclassed and subclassed controls can add customized behaviors, but the underlying system code for the standard control type handles basic control functions, which supports Active Accessibility.

Superclassed and subclassed controls must respond to the normal messages for their class. As with owner-drawn controls, label each control properly, even if the label will not be visible on the screen. These simple steps enable the control to successfully support Active Accessibility.

Custom Controls

Custom controls include ActiveX controls, Java controls, and UI elements drawn directly on the screen without an associated window. Accessibility aids may have difficulty manipulating these custom controls when identifying them by name, role, state, location, or determining if they have the keyboard focus or selection.

Active Accessibility is based on the Component Object Model (COM), the Microsoft developed, industry standard that defines a common way for applications and operating systems to communicate. Active Accessibility provides dynamic-link libraries (DLLs) that are incorporated into the operating system as well as a COM interface and application programming interface (API) elements that provide reliable methods for exposing information about user interface elements.

Comment [MJH32]: 6/10/99 This, or a briefer version seems necessary here.

Active Accessibility is the only effective way to make custom controls compatible with accessibility aids. If an application cannot support Active Accessibility, the following steps can help make controls somewhat, but not fully, compatible with accessibility aids:

- **If the custom control has its own window -** Return a descriptive string when the control is queried using the WM_GETTEXT message.

  Example: A button control labeled Print could return the string “Print button” to identify both control type and label. The same string would be appropriate for a non-text labeled control graphic of a printer. Similarly, a custom control that behaves like a check box could return the caption string “Printing Enabled check box, checked.”
- **If the custom control has its own window -** Support the WM_GETDLGCODE message, identifying the keyboard input that is supported.
- **If the custom control has no window -** Move the system caret to communicate the focus location to accessibility aids as described in the previous section titled “Exposing Keyboard Focus.”

Owner-Drawn Menus

Owner-drawn menus are incompatible with most accessibility aids that must identify each menu item's name. There are two ways to overcome this limitation:

- Expose the menu item name using the **MSAAMENUINFO** structure. For more information, see the Active Accessibility documentation.
- Provide an option to replace graphic menus with standard, text menus when an accessibility aid is active. To test this, call: **SystemParametersInfo** with the **SPI_GETSCREENREADER** option.

People with cognitive disabilities can also benefit from an option to show text, menu graphics, or both.

Example: A line width menu item might display a line sample and text stating the width. This redundancy is also useful for people following precise specifications. Make text/graphic duplication the default, or allow users to select the text-menus option. This is the preferred choice of Microsoft Developer Studio.

Comment [MJH33]: 6/9/99 Is it the Studio's preferred choice, recommendation or guideline? Can we back this up? As it stands, I get no do/don't do instructions.

The following illustration contrasts an owner-drawn text and graphics menu with a standard text-only menu.

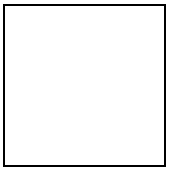

Figure 16 - *Left,* owner-drawn Start menu displaying icons and text - *Right,* the same controls rendered as a normal text only menu.

Using Text Controls

Screen review utilities must recognize text on the screen and in a document. Text is normally displayed using the Windows drawing functions, by copying images directly to the screen, or by using one of several text controls.

The most accessible text presentation uses the standard Windows edit, static, status, and HTML controls, all of which expose their text to accessibility aids using Active Accessibility. The HTML control is the best and easiest solution for displaying richly formatted text. All of the above can be used without a visible border, to fit seamlessly into a UI.

Comment [MJH34]: 6/9/99 Is this a verifiable fact or opinion?

Choosing the correct text control improves keyboard access to an application and assures compatibility with accessibility aids.

For more information, see the Keyboard section of this document.

If you use graphic images that contain text, see "Identifying Images and Bitmapped Text" in this document.

Naming Controls

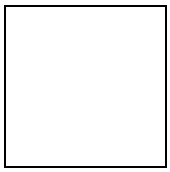

Figure 6 - Run dialog box, showing a static text control label for a drop-down list box. Static text ends with a colon.

When using Windows dialog boxes, you must label all controls so that accessibility aids can identify them. Labels are defined in two ways:

- The control's caption text
- A separate static text control

Accessibility aids can easily identify controls such as buttons that display defined caption text, but aids may have trouble identifying other controls (such as edit controls and list boxes) labeled with separate static text control. To ensure that these controls can be identified, provide meaningful caption text even though it will not be displayed.

Additionally, to expose the relationship between a control and its static text label, the static text must immediately precede the named control in the dialog box TAB order. Correct tab order also allows a static text control to define an underlined access key for the control it labels.

Naming Windows

Assign every window a user-friendly caption, whether visible or not on the screen, so accessibility aids can clearly identify the window to the user.

Every window can have a caption, whether it has a visible caption bar or not. Set this string in your resource script, when calling **CreateWindow** or related functions, or by calling **SetWindowText**. Accessibility aids are then able to query the caption using Active Accessibility or the **WM_GETTEXT** message.

This applies to all windows: top-level windows with visible frames, child windows such as floating palettes, custom controls, toolbars, and panes within a window when implemented as child windows.

Identifying Windows by Function

Identify each type of window by assigning unique window classes. This identification should not change with subsequent versions of the application or when translating to other languages.

Comment [MJH35]: 6/9/99 How? Are most properties localized?

Accessibility aids must have this identification to customize handling different windows within the same application. There may be separate instructions for handling these windows, such as announcing specific areas when the content changes. While humans can identify a window by its caption, accessibility aids must have an identifier that does not change with different documents or languages.

Identifying Images and Bitmapped Text

Screen review utilities must describe images on the screen to the user, especially images that convey important information or have bitmapped text. This is required only when a screen review utility is running and is determined by calling: **SystemParametersInfo** with the **SPI_GETSCREENREADER** value.

- Use Active Accessibility to expose the name and description of each image.
- Use the standard Windows ToolTip control to apply a name or brief description label to each image.
- Provide an option to replace the image with text.
- User manipulated images should be handled like controls.

Microsoft Internet Explorer supports the above mechanisms.

See: "Identifying Images and Bitmapped Text" in this document for more information.

Labeling Objects Clearly

Users with tunnel vision, or who require screen magnifiers, can see an object and perhaps some of its immediate surroundings. Users who are blind encountering the same object with Active Accessibility will have only its name, its type, the name of the window and any group box it is in, but none of the context provided by visual/spatial arrangements.

- **Label objects with names that make sense when taken out of context –** Labels do not have to be long or detailed, but should be logically descriptive.
- **Avoid assigning multiple objects the same name –** Example: Two similarly named buttons on the same dialog
- **Place objects in separate containers if duplicate name are unavoidable –** Example: Dialog box controls with the same name should be enclosed in separate group boxes with unique names.

An example of displaying multiple controls with the same name is the Keyboard page in the Accessibility Options section of Control Panel.

Comment [MJH36]: 6/9/99 Definition here, please.

Drawing to the Screen

Active Accessibility, or one the alternate methods described above, should be used to expose window content. When an application doesn't support the above methods, screen review utilities attempt to infer its status by monitoring calls to drawing functions and storing the locations of text and drawn graphics, and attributes such as font, size, color, and style. They also rely on the ability to monitor normal Windows drawing operations examining information copied from one location to another and being erased or overwritten by other text or graphics.

If the above methods cannot be supported, use the following recommendations to provide some degree of compatibility with screen review utilities or provide an option to do so.

Comment [MJH37]: 6/9/99 I can't locate these recommendations.

If an application uses any of these techniques when a screen review utility is running, it should also use conventional methods for performance. This can be determined by calling the **SystemParametersInfo** function with the **SPI_GETSCREENREADER** value.

Comment [MJH38]: 6/10/99 The logic of the first sentence seems reversed. Also, why would an app use "conventional methods" when the screen utility is running? And while we're at it - Does conventional mean not using the previously listed techniques?

Verification

The Inspect Objects tool is included in the Active Accessibility SDK. Hover the mouse pointer over portions of an application to determine if the tool displays the proper description. Compare this description with standard windows and controls to ensure they are the same.

Test applications by running accessibility aids to assure they work together properly. Many accessibility utilities are available at no charge or as trial versions.

More information about testing can be found in Appendix A of this document.

For a catalog of these tools, go to: http://microsoft.com/enable/.

Color

Using color appropriately can significantly enhance a UI, but colorblind users require that information is not conveyed by color alone. Color should be used to enhance, emphasize, or reiterate information shown by text, images or diagrams.

The most important aspect of using accessible color is using **SystemParametersInfo** to responding to the Accessibility Options High Contrast setting in Control Panel. This setting indicates that the screen appearance has been modified for enhanced legibility.

This section covers:

- High Contrast Mode
- Making Colors Customizable
- Information Conveyed by Color Alone
- Text Obscured By Background Design Or Related Color

High Contrast Mode

When the High Contrast option is set:

- Display only user-selected or user-customized system colors selected through Control Panel.
- Simplify document sharing by allowing users to adjust screen colors drawn without altering the document.
- Use correct foreground/background color combinations.
- Omit background images drawn behind text.

★★★★
Logo Requirement

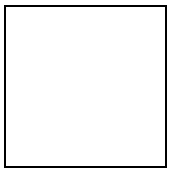

Figure 18 - High Contrast Black color scheme.

The screenshot above shows High Contrast text – light text on black background – and a portion that is harder to read – black text on gray background.

Basic color guidelines require conforming to standard system colors or allowing user-chosen colors not defined by the system. When this is unsuitable for the default configuration, users can choose high legibility by enabling the High Contrast Accessibility Option in the Control Panel.

Required steps

- Use only customizable colors through the Control Panel or the application's view options. Documents are often shared, so it is unacceptable to require document changes to display different text and background colors.
- Use correct foreground/background color combinations.
- Omit complex backgrounds or images behind text and controls.

Comment [MJH39]: 6/9/99 Are there "correct" colors or do we mean effective, appropriate or suitable?

Additional recommendations

- Supplement information usually conveyed exclusively by color with text, graphics, or patterns.
- Draw monochrome images with standard foreground/background color combinations rather than multiple color images.
- Replace application-specific colors with Control Panel color selections and use the fewest possible color combinations.

Exceptions

- Color selection palettes may continue to display the full range of fixed colors.
- Applications are not responsible for altering the appearance of third-party controls or embedded objects, unless shipped with the application.

Procedure

- Applications can verify the High Contrast setting by calling the **SystemParametersInfo** function with the **SPI_GETHIGHCONTRAST** value. This value should be queried during initialization and when processing **WM_COLORCHANGE** messages.
- Call **GetSysColors** to identify the colors chosen through Control Panel.

Making Colors Customizable

Color preference is a choice, but for low vision and color-blind users it is a *crucial* choice. Many users require high contrast between text and background or a particular scheme, such as white text on a black background, to prevent the background from "bleeding" over and obscuring foreground text. Some people find the default color scheme legible, but experience eyestrain after long viewing. In addition, nearly 23 million Americans (8% of male and 0.5% of female population), have

some form of color blindness that makes certain color combinations unreadable.

Some applications use fixed colors to prevent users from selecting an unattractive color scheme, but users may have perception problems with a fixed color scheme.

Using Control Panel Colors

Where possible, applications should conform to user-selected standard system colors available through the Control Panel. This is simple when an element in an application's window corresponds to a usage handled by Control Panel (window text, button face, dialog box text, etc.). User-chosen color combinations reduce the possibility that preset colors will make an application unusable, while ensuring its appearance and readability is pleasing to the user without providing a color-adjusting interface.

For a complete list of system colors, see the **GetSysColors** function description in the Microsoft Win32 Software Development Kit (SDK).

The color and use selected in Control Panel do not need to correspond exactly. Example: The user's choice of window text color and background is probably a safe combination to use for any purpose.

Using Private Colors

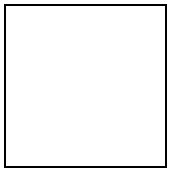

Figure 19 - Internet Explorer dialog box that allows link color adjustment.

If application elements use colors that do not correspond to Control Panel selected system colors, provide a method for color adjustment.

Example: When designing a calendar application with various background colors to indicate different events, allow the user to select the colors.

If an application cannot provide a UI for customizing colors, support a registry key that selects the colors and provide a .REG file users can edit to adjust these settings.

Using Proper Color Combinations

Applications should always use system colors in the correct foreground-background combination to provide contrast. Users would not logically choose the same button text (COLOR_BTNTEXT) and button face color (COLOR_BTNFACE, COLOR_3DFACE), so this combination will always be legible. However, a user might alter the color scheme so normally contrasting system colors, such as button text and window background (COLOR_WINDOW), are the same. If an application draws

using colors not specifically designed for combining, the information may be unreadable.

Applications should always draw foreground objects in foreground colors and fill backgrounds with background colors. Many users require specific high-contrast combinations, such as white text on a black background. Drawing these reversed, as black text on a white background, causes the background to "bleed" over the foreground and can make reading difficult, even painful, for some users.

The following list shows safe and unsafe onscreen combinations.

Combination	Status
Window Text on **Window Background**	**Safe**
Button Text on **Button Face**	**Safe**
Window Text on **Button Face**	**Unsafe** to mix unrelated colors
Button Text on **Window Background**	**Unsafe** to mix unrelated colors
Window Background on **Window Text**	**Unsafe** to reverse colors
Button Face on **Button Text**	**Unsafe** to reverse colors

These rules also apply when allocating user-select private colors in an application. Draw foreground objects in user-selected foreground colors and background objects in user-selected background colors.

Coloring Graphic Objects
Graphic objects present special challenges.

Example: Some application display buttons use pictures, text, or both. The following scenarios describe how Control Panel color-selection affects these variables:

Monochrome image - A button face should always be drawn in the standard system color, and the foreground image should be drawn in the standard button text color. If the image is drawn within a window rather than on a button, use COLOR_WINDOW and COLOR_WINDOWTEXT values instead of the button colors.

Multicolored image - The simplest solution is to include an equivalent monochrome image for use on monochrome displays, or that can be supplied when the user chooses a non-default button face color or has requests High Contrast Mode (described earlier in this document).

Comment [MJH40]: 6/9/99 I believe I understand this correctly, but maybe we can say it more clearly and succinctly.

If an application cannot include monochrome images, modify multicolored images by identifying light area as foreground and dark areas as background.

Example: A bitmap having a multicolored object on a white background can be mapped with all colors other than white in the appropriate system foreground color and white in the system background color. This relationship may be reversed for images designed to have a dark background.

Information Conveyed Exclusively by Color

Do not communicate important information by color exclusively – provide other options to receive this information.	☆☆ **Recommendation** Blindness / Low Vision

.

When an application conveys information by color exclusively, some users will not be aware of the message. Even including an option to customize colors is insufficient to remedy this situation if the user can only read white text on a black background or is using a monochrome display hand-held computer. To avoid this limitation, applications should make all displayed information available by other means, even when the user has an option to choose the colors.

One option is to provide patterns as an alternative to colors. In the above calendar /color example, users could be given a choice of patterns along with colors for each scheduled event. This would supply color combinations that "read", while offering additional background pattern information. Ensure that the patterned background does not interfere with the readability of the text.

Comment [MJH41]: I can't find this example among the original graphics.

Text Obscured by Background design or related color

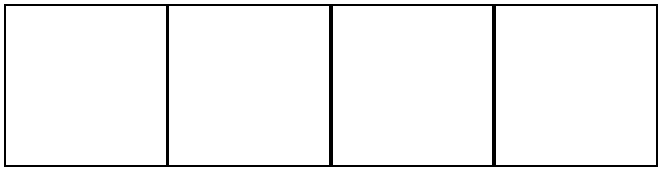

Figure 20 – An example of difficult to read text. Numbers formed by low contrast color dots on a spotted background of related low contrast colors.

Text is most legible when drawn against a plain contrasting color background. Text drawn over a variegated background, such as a wash of colors or bitmap, may be unreadable to many users. Always provide users with the option of omitting the image and reverting to a bare background.

- Provide a menu or dialog box option to omit complex backgrounds.
- Omit the background in response to the High Contrast setting as discussed earlier in this section.
- Omit background images if the foreground color changes or consists of mixed colors.

  Example: Text drawn over a very light bitmap image can appear legible in the default color scheme, but appear unreadable when a user chooses a light foreground text color.
- Use backgrounds that contrast well with text. Example: Many users find it hard to read black text drawn over gray, brown, or other dark backgrounds. This applies to all backgrounds, even when there is an option to omit them.

Strategies for addressing this guideline basically revolve around eliminating the requirement that a person see color to operate the device. This does not eliminate the use of color in any way as long as the information conveyed by the color is also conveyed in some other fashion. In addition, there are a number of things that can be done to allow even individuals with color anomalies to be able to take advantage

of the color-coded information. First, there are a number of common pairs of colors that are indistinguishable by people with color perception anomalies. **(And those pairs are?)**

Comment [MJH42]: Do you have a preference or shall I draw them from the Encarta reference?

Avoiding these color pairs avoids or reduces the problems for these individuals. In addition, as long as the colors have different hues and intensity, differently colored objects can be distinguished even on a black and white screen by their different appearance. Depending upon the product, the manufacturer may also be able to allow the user to adjust colors to match their preferences and visual abilities. It is generally a good idea to also avoid muted colors with a low luminance value (intensity).

Size

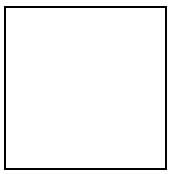

Figure 7 - Control Panel (detail) with extra large High Contrast Black option selected. Icons, icon titles, menu text, and title bar text are all larger than normal.

Some users prefer to display a lot of information on a single screen, but millions of users have difficulty reading small text or identifying and clicking on small objects.

Present several options when sizing objects on the screen.

System Size Settings

Applications must comply with user-chosen size and font system settings.	★★★★ **Logo Requirement** Dexterity / Low Vision

Screen element dimensions should reflect user-selected Control Panel size settings (scroll bars, custom menus, etc.). Example: A custom dialog box should use the system's user-selected dialog box font to display text.

This section covers:

- Avoiding Small Fixed Fonts
- Line Width Minimums
- Supporting Adjustable Fonts
- Scaling Non-document Regions
- Global Scaling
- Alternatives to WYSIWYG
- Adjusting Images
- Avoiding Font Dependencies

The following **GetSystemMetrics** and **SystemParametersInfo** descriptions are available in the Win32 SDK.

Applications must3 comply with the following font and size system settings:

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	

Applications should comply with the following recommended system size settings where pertinent:

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

Verification
Choose the "Windows Default (extra-large)" scheme on the Control Panel, Display, Appearance page to confirm that:

- The application adopts the size settings
- Onscreen information is not lost or unintelligible
- The application still *fully* usable

Avoid Small Fixed Fonts

Avoid hard coding any font sizes smaller than 10 points.	★★★ **Future Logo Requirement** Low Vision

Small fonts can cause eyestrain and are difficult for many people to read.

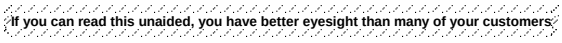

Figure 8 – Tiny typeface reading: "If you can read this unaided, you have better eyesight than many of your customers."

Line Width

Although many applications draw fixed lines that are one pixel wide, these disappear on high-resolution monitors and can be imperceptible to users with low vision. Rather than using fixed widths, determine the correct line thickness by calling the **GetSystemMetrics** function with the **SM_CXBORDER** and **SM_CYBORDER** values. These values are

defined with regard to the system's monitor resolution. Future Microsoft operating systems will allow users to adjust these settings appropriately for their vision.

Support Adjustable Fonts

Users who prefer small type or require larger type appreciate the ability to choose the typeface and size. This simple option makes applications much more useable.

The preferred selection method is to provide font choices through a menu option or property sheet. To display the standard Font dialog box, call **ChooseFont**. If text drawn with a user-requested font forces information to extend beyond the window's edge, make the window resizable and display scroll bars.

Another method is to resize fonts automatically when the window is resized, but this prevents large font selection in a small window with scroll bars.

Scale Non-document Regions

Many application windows contain a user-created document image and one or more application panes.

Fixed size buttons, rulers, toolbars, palettes, navigation bars, and status bars can present usability and accessibility problems. Where possible, allow users to independently select a size or zoom ratio for each non-document region.

Example: A “Toolbar Size” option would allow users to size all toolbar buttons.

Global Scaling

Allow for changes to system font size and pixels per logical inch. υ Avoid drawing with the MM_TEXT mode. υ Stretch, shrink, pad, or crop images to fit available space.	✯✯ **Future Logo** Requirement Dexterity / Low Vision

Windows provides a Custom Font Size feature in the Control Panel, Display section that allows users to scale all fonts and most other on-screen elements globally by changing the number of pixels per “logical inch.”

To be compatible, applications must avoid drawing in MM_TEXT mode, which bypasses logical scaling. Onscreen interface elements using MM_TEXT will be drawn out of proportion to non-MM_TEXT screen elements.

Note: The Custom Font Size feature does not automatically scale bitmaps. See: “Adjusting Images.”

Alternatives to WYSIWYG

Applications offering a WYSIWYG (What You See Is What You Get) document view, present onscreen text as it will appear when printed, but this may not always be appropriate or desired.

Comment [MJH43]: Is this true? We need to state which mapping mode (MM_ISOTROPIC) to use. Plus we need information on setting the scaling factors (with window + viewpost extents)

Comment [MJH44]: 6/9/99 **MM_TEXT** is the default mapping mode (and easiest to use) so which mapping should be used? I’m not seeing a choice here.

Example: Preparing to print small font text, but requiring a larger font display for editing. Draft mode, zoom features and wrap to window options are simple methods for users to size on-screen information.

- **Draft mode** provides the option of displaying all text in a single font and size. If possible, allow users to select these settings and employ underlining or similar denotation to indicate special formatted text, such as bold or italics. Draft mode also improves application performance on slow or reduced memory systems.
- **Zoom features -** scale all document elements to a user-selected ratio. Many applications, (including Microsoft Word and Microsoft Excel) offer this feature, benefiting many users who do not consider themselves disabled. Use TrueType scalable font technology to ensure clearly defined characters at virtually any size.
- **Wrap to window options -** allow users to ignore normal line breaks in text documents. Applications should not break onscreen lines to emulate the printed page.

Adjust Images

Occasionally a graphic image must conform to a different space than originally intended. If users adjust the Custom Font Size global scaling constant or if an application allows users to choose a differently sized font, the application drawn image could appear out of scale with the rest of the window. Solution: Change how the bitmap is drawn to the screen or use an alternative to bitmaps.

How to adapt a bitmap when changing font size:

- **Stretch or shrink –** Scale the bitmap "at run time" using the **StretchBlt** function to size it appropriately for its screen location.
- **Pad –** If stretching is not an option, draw the bitmap normal size and leave blank space around it. To prevent the bitmap's "blank frame" from displaying older, unrelated information, erase the area to the appropriate background color. Use a mapping mode other than MM_TEXT to specify the frame size so it will automatically adjust the global scaling factors.
- **Crop –** If shrinking is not an option, clip the bitmap to fit the surrounding rectangle.

  Example: When the Custom Font Size feature global scaling ratio is less than 100%, specify size using a mapping mode other than MM_TEXT to automatically adjust the global scaling factor.

Bitmap alternatives that adjust easily to different spaces:

- **Metafiles -** A convenient way to encapsulate images for easy playback, can be scaled automatically to fit a destination rectangle, and usually look good at any size.
- **Drawing on the fly –** For simple images that do not require encapsulation for storage and playback, create a routine that draws "on the fly" using Windows' graphic functions.
- **TrueType glyphs** - Although glyphs can represent an image, using them can be costly unless created in-house. Custom TrueType glyphs are not recognized by accessibility aids, so it is important to label the graphic properly, as outlined earlier. See: "Identifying Images and Bitmapped Text" in this document.

Comment [MJH45]: **MM_HNISOTROPIC** or **MM_ISOTROPIC**?

Comment [MJH46]: **MM_HNISOTROPIC** or **MM_ISOTROPIC**?

Avoiding Font Dependencies

Some application dialog boxes are so tightly spaced they look unsightly when users change the dialog box font. Leave enough white space in dialog box layouts to accommodate moderate font changes that make the dialog box easier to read. Extra space also makes an application easier to localize into other languages.

Some developers have expressed concern that changing the font can cause a dialog box to scale incorrectly, but this rarely occurs. Windows automatically positions dialog box controls based on the size of the dialog box font.

Changing fonts can be a problem when an application draws directly into elements of a dialog box.

Example: Applications that create a static control and draw over it to effect a custom design element. This element can appear incorrectly if the static control size is scaled to match a new dialog box font size. Avoid these problems by determining the proper location and size at run time.

Some applications include specific dialog box fonts rather than relying on the user-selected system dialog box font, but this can cause problems. Allow users to select this font or change its size to match the currently selected system font. See the "Color" section of this document.

Comment [MJH47]: 6/9/99 Examples? Or at the least, the *type* of problems.

Sound

This section covers:

- Sound Alternatives
- Making Sounds Customizable
- Reinforcing Sounds and Visuals
- Turning Sounds Off

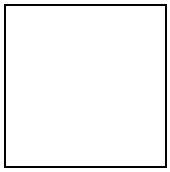

Figure 27 - Control Panel section displaying ShowSounds option.

Appropriate sound communications can benefit a wide range of people with disabilities, but not when used exclusively. Users may be:

- **Deaf or hard-of-hearing**
- **Working in a noisy environment -** airplane, workshop, crowded office, etc.
- **Working in a quiet environment where sounds are inappropriate -** office, conference, library, etc.

Sound Alternatives

Do not present important information	★★★★

exclusively by sound. υ Provide a visual presentation option υ Display information visually when the ShowSounds option is True. υ Do not disable sounds automatically. υ Provide closed captions or transcripts for all audio content.	**Future Logo Requirement** Hearing / Cognitive Language

Supporting the ShowSounds Option
By choosing the ShowSounds option in the Accessibility section of the Control Panel, users indicate that important information must be displayed visually - the equivalent of closed captions for sound cues.

Applications check the ShowSounds flag by calling the **SystemParametersInfo** function with the **SPI_GETSHOWSOUNDS** value.

Using the ShowSounds flag does preclude normal sound presentation; in fact, redundant sound and visual information increases an application's usability. Users should have the option of requesting visual or audible feedback independently.

The ShowSounds flag applies only to applications that usually present important information exclusively with sounds. Applications are responsible for determining how this information appears visually. The following examples outline specific situations and sound solutions:

Types of Sounds
Application sound cues can be divided into four general categories:

- **Important sounds** convey information not presented visually, but are important to the application's operation.

  Example: Audio wave files with narrative instructions or a "new mail" arrival notification.

- **Redundant alerts** accompany the visual presentation of information and attract a user's attention when not watching the computer screen.

  Example: An optional beep that accompanies a message box.

- **Redundant sounds** repeat information presented visually, but are not required to use the application properly.

  Example: An error sound or beep when a user moves beyond the end of a list box.

- **Decorative sounds** enhance an application's presentation or appearance, but are not required for operation.

  Example: Sounds that accompany minimizing a window or activating a menu and background sounds, thematic music or sound effects in multimedia games.

Users who cannot hear redundant or decorative sounds are not disadvantaged, but conveying important sounds and redundant alerts is essential.

For redundant alerts, users can select the Windows SoundSentry feature. This utility displays a generic visual indicator when it detects the computer producing a "signal sound." This is acceptable when the sound is simply a warning to attract the user's attention, but insufficient when an application uses various sounds to convey complex information.

Applications must adopt alternative methods of conveying important sounds and complex information.

Comment [MJH48]: Does this really work?

Playing Audible Alerts
To attract attention, as when new email arrives, an application should use one of the following techniques:

- **Flash the title bar by using the FlashWindow function.** If the window is not visible, the application's button on the taskbar will flash.
- **Display a message box that acquires the activation and keyboard focus.** This is a viable option unless the user is typing into another application at the time.
- **Display a status indicator on the notification area of the taskbar.** This indicator should flash when initially displayed to attract the user's attention.

Playing Redundant Sounds
Applications often produce an error status sound, such as when a user types an invalid character. In this case, an application can flash its title bar or application button on the taskbar, using the **FlashWindow** function.

Displaying Closed Captions for Video and Audio
Applications displaying multimedia animation, video clips or audio clips should support the ShowSounds flag with true closed captioning, if the clip has necessary information on the audio track.

Comment [MJH49]: 6/9/99 Are we talking about SAMI here?

There are four closed captioning options:

- **Microsoft DirectPlay** (formerly ActiveMovie 2.0) is an easy way to create and display closed captions for many types of digital media. See: http://microsoft.com/directx/.
- **Microsoft Video for Windows** is an older technology used to create a separate, synchronized caption data stream the application can then use for screen display.
- **Private mechanisms -** custom designed application technologies for presenting multimedia content.
- **Transcripts -** non-synchronized screen display of audio track content. While acceptable for short audio pieces, this is not suitable for long audio pieces or closely synchronized audio and video.

Making Sounds Customizable

The Control Panel allows users to customize the sound scheme to play more, fewer, or no sounds for specific events such as an error, warning, or menu item selection. Windows automatically conforms to these

preferences when carrying out the associated event. Applications should call **PlaySound** to trigger sound events.

The following guidelines apply to all audio events:

- **Always play sounds by specifying a registry entry.** (Use the **SND_ALIAS** value.) This allows users to customize sound associations through Control Panel. Accessibility aids can use the name provided in the registry to describe the event.
- **Trigger standard sound events when carrying out equivalent actions.**

  Example: When displaying the equivalent of an urgent message box, play the SND_ALIAS_SYSTEMEXCLAMATION event, consistent with the Windows environment.
- **Define as many application-specific sound events as possible -** This allows most events to be silent by default, but lets users desiring additional feedback to add appropriate sounds through the Control Panel. This option is especially useful for people with low vision or certain cognitive impairments.
- **Avoid specifying sounds by filename or resource -** These cannot be customized through Control Panel and accessibility aids are unable to determine the applicability of sounds specified by filename or resource.
- See: The Win32 Software Development Kit (SDK), "Playing Sounds Specified in the Registry"

Comment [GCL50]: Page: 37
7/21/97: Add code fragment.

Reinforcing Sounds and Visuals

Redundant sound and visual cues generally increase an application's usability.

Example: A status indicator on the screen should produce an audible cue when it changes. Conversely, appropriate words or graphic changes on the screen should accompany Web page start sound. Users should be able to request simultaneous or independent visual and audible feedback.

Turning Sounds Off

Users should have the option of turning off application sounds to avoid distracting people who are hard-of-hearing (mis-interpreting sounds for conversation), or when such sounds would be inappropriate (crowded offices, public environments, libraries, etc). This is especially true of non-essential or redundant sounds that are supplemental to on-screen information. People who are deaf or hard-of-hearing appreciate this option because it prevents unnecessarily disturbing people in their proximity.

Comment [MJH51]: 6/9/99 I think this is what you were suggesting.

To compensate for not providing an option to turn off sounds, check the **SM_BEEP** option using the **GetSystemMetrics** function. If FALSE, the user has chosen to disable the standard system beep, and implies that other sounds should also be turned off.

Sounds played by calling **PlaySound** and specifying a registry-based sound event do not require additional adaptation, because users can turn them off by changing the Control Panel sound scheme.

Timings

Allow user-customized interface timings. υ Provide an option to avoid timed out messages. υ Allow slowing or disabling of flashing screen updates.	✯✯ **Future Logo Requirement** Blindness / Low Vision Dexterity / Cognitive Language / Seizures

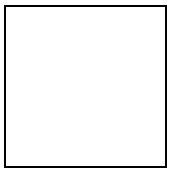

Figure 9 - Illustration of mouse and stopwatch

Some users have difficulty reading briefly displayed information, or encounter problems when performing tasks that require quick reflexes. All timed events should be user-adjustable.

This section covers:

- υ Input Timings
- υ Time Limits
- υ Rapid Flashing

Input Timings

Some users with relatively slow reaction times have difficulty using fixed-timing features such as:

- Automatic scrolling when users drag an object towards the edge of a window
- Holding down the mouse button while the pointer is over a scroll bar
- Applications that react quickly when the keyboard focus remains on an object, frustrating users who navigate slowly.

Several ways to adjust timing parameters are:

- υ **Use a system setting -** Applications can synchronize timed behavior to a user-specified Control Panel input timing setting.

  Example: Double-click speed and the amount of time focus remains on a cascading menu before the child menu is displayed.
- υ **Provide a timing option -** Applications can provide a menu or dialog box option to adjust the timing with settings stored in the registry's user-preferences section.
- υ **Provide a registry setting -** If a user-customized timings interface cannot be provided, support the option using a registry key and provide a .REG file that can edit to adjust this setting.

Time Limits

Avoid having messages time out or provide a time-out override option. There are several reasons why users may not notice a briefly displayed message:

- Users who read slowly because of lack of practice, cognitive disabilities, or when working in a second language
- Users who rely on screen magnification utilities have to locate information on the screen
- Users who respond slowly due to impaired dexterity

If a message is important, it should be displayed until the user dismisses it deliberately. Even if a message is unimportant, it is disconcerting to the user for it to disappear before it can be read, leaving a doubt that it may have been a necessary alert.

Rapid Flashing

Allow users to slow or disable rapid flashing or updates.	★★ **Recommendation** Seizures

If a user is prone to epileptic episodes, rapid visual changes can induce a seizure. Other stimuli include visual signals flashing on and off and different images being displayed repeatedly. Because individual susceptibility to this condition varies widely, it is impossible to eliminate the risk of triggering some seizures. The following precautions reduce the risk of triggering seizures:

- **Provide an option to slow or stop flashing.** Recommended solution.
- **Avoid very rapid flashing.** Flash rates of more than five times per second affect more people.
- **Avoid flashing large areas.** Small areas, such as the flashing insertion bar, are less likely to trigger seizures than larger flashing areas.
- **Avoid high contrast flashing.** Decrease image brightness ratio to reduce seizure risk.
- **Avoid multiple, simultaneous on-screen changes.** Several quickly changing, non-synchronized objects can produce the same effect as a single large image flashing much more rapidly.

Comment [MJH52]: How would this be accomplished?

Several methods allow the user to adjust object flash speed:

- **Use the Caret blink rate.** An easy solution is to synchronize object flash rate with the Control Panel's user-selected caret blink rate. To query this setting, call: **GetCaretBlinkTime**.
- **Offer a timing option.** Provide a menu or dialog box option to adjust the timing. These settings should be stored in the user-preferences section of the registry.
- **Provide a registry setting.** If a user-customized timing interface is not provided, support this option with a registry key and a .REG file to adjust the setting.

Unexpected Side Effects

Allow mouse movement and keyboard navigation without triggering unexpected side effects.	★★ **Future Logo Requirement** Blindness / Low Vision

Dexterity / Cognitive

Some users who rely on accessibility aids must move the mouse or keyboard focus to explore the information on the screen. These actions should not trigger unexpected side effects.

This section covers:

- Side Effects of Mouse Movement
- Side Effects of Keyboard Focus Movement

Side Effects of Mouse Movement

Avoid triggering events when the mouse pointer moves onto or off a special area. If triggering must be included, it must be made optional.

Comment [MJH53]: 6/9/99 Does this mean MSAA Win Events, other types of events or both?

Some accessibility aids require mouse pointer movement when information is explored on the screen.

Example: A screen review utility may move the mouse to follow words being read, or a user may need to move the mouse to enlarge specific text. If text appears when the mouse moves over an object and disappears when the mouse moves off it, the text virtually disappears as the user attempts to read it!

There are two situations where mouse pointer movement-triggered changes are acceptable. These exceptions are currently managed by accessibility aids and are handled appropriately:

- **Pointer shape -** It is acceptable to change the shape of the mouse pointer as it moves.

  Example: Pointer shape may change to indicate an object is a valid drop target.

- **ToolTips -** It is acceptable, and encouraged, to use ToolTip controls to display explanatory information when the pointer is paused over an object. However, this exception applies only if a standard ToolTip control is used.

Side Effects of Keyboard Focus Movement

Avoid triggering events when keyboard focus moves. If triggering is necessary, it must be made optional.

Comment [MJH54]: 6/9/99 I am unclear about the import of this statement. Optional, necessary events?

To allow users to read or explore the content of a window, support keyboard mechanisms that allow uninterrupted focus change to a control or an area.

Example: Blind users use the TAB key to move through all dialog box controls, exploring options before returning to the appropriate choice.

For accessibility aids to work appropriately, include standardized application behaviors and mechanisms in the following situations:

- **Explanatory text –** Applications may display explanatory text giving menu function details while processing menu messages. Although it is preferable to draw this text in a status bar for compatibility with other applications, any text drawn during menu processing will be assumed to serve this function.
- **Radio buttons -** Applications may automatically change option control, such as radio buttons, values and tab controls during

keyboard navigation. Although this behavior can cause problems for keyboard users, it is necessary for backward compatibility.

Comment [MJH55]: 6/9/99 Do we need to explain this backward compatibility further?

- **Providing alternatives -** Applications that react to focus movement should provide an alternative way to move the focus. This is commonly accomplished by using the CTRL key to modify a navigation key.

  Example: In Windows Explorer, or in list boxes that support discontiguous selection, users can move the focus and change the selection when navigating with an arrow key. However, users can move the focus without changing the selection by holding down the CTRL key when pressing an arrow key. This keyboard action is supported by radio buttons in Microsoft Office.

Comment [MJH56]: 6/9/99 Is keyboard action what we should be referring to here?

Mouse Input

Mouse input should not be required, but well-designed mouse support makes applications easier to use for many people.

This section covers:

- System Mouse Settings
- Well-designed Mouse Shortcuts

System Mouse Settings

Applications must be compatible with specified system mouse input settings.	☆☆ **Logo Requirement** Dexterity

Applications must comply with the following system mouse input settings:

Required Mouse Settings	
SM_CYDOUBLECLK, SPI_CXDOUBLECLK	Height and width of target rectangle for double-click
SM_CYDRAG, SM_CXDRAG	Height and width of target rectangle on a drag point
SPI_GETMOUSEHOVERHEIGHT	Height of mouse target rectangle
SPI_GETMOUSEHOVERTIME	Mouse hover time in target rectangle
SPI_GETMOUSEHOVERWIDTH	Width of mouse pointer target rectangle

It is recommended, but not required, that applications comply with the following system settings:

Recommended Mouse Settings	
SPI_GETMOUSE	Threshold values plus acceleration
SPI_GETMOUSEKEYS	MouseKeys accessibility feature
SPI_GETMOUSESPEED	Mouse speed
SPI_GETMOUSETRAILS	Mouse Trails feature enabled or disabled
SPI_GETACTIVEWINDOWTRACKING	Active window tracking (mouse activating)

SM_MOUSEPRESENT	Mouse installed
SM_MOUSEWHEELPRESENT	Mouse with wheel installed
SPI_GETWHEELSCROLLLINES	Lines scrolled / mouse wheel rotation
SM_SWAPBUTTON	Left/right mouse buttons swapped

Well-designed Mouse Shortcuts

The following steps make it easier for people to use your application:

- **Provide mouse shortcuts for commonly used features.** Providing good mouse support makes an application easier for most users, especially those who can only use a pointing device. Text can be entered using on-screen keyboard utilities, but is less efficient and convenient than using mouse shortcuts.
- **Support simple mouse operations.** Some users have difficulty double-clicking, so, where possible, require only single-clicks for common operations. Some users have trouble holding down the mouse button while moving it, making drag-and-drop operations difficult. Also, avoid requiring secondary mouse button input because some pointing devices and many alternative accessibility devices do not support it.
- **Provide customizable toolbars.** Allow user-created, mouse shortcut-activated commands that would normally require keyboard input or multiple mouse clicks.
- **Do not require mouse input for any feature.** See "Keyboard Access" in this document.

Customizable User Interface

Where possible, allow users and/or administrators to customize the application.	**Recommendation** Blindness / Dexterity Cognitive / Language

Applications can be adapted to the needs of a group or individual with customizable keyboard commands, menus, dialog boxes, and toolbars.

This section covers:

- Approaches to Customization
- Customization Areas

Approaches to Customization

The following user-preference options reduce application complexity and improve feature access:

- **Detailed customization –** When users can configure individual menu items and commands.

  Example: Hiding the toolbar or status bar.
- **Schemes –** When users can partially re-configure the interface.

  Example: Offering the choice of advanced, intermediate, or novice menus.
- **Macro capabilities –** When users can create custom dialog boxes to supplement or replace existing functionality.

Customization Areas

The following types of customization benefit many users:

- **Omitting extraneous graphics -** Provide an option to hide graphics that do not convey necessary information.

  Example: Option button icons can be hidden if the button function is described by accompanying text. While most users find graphical "decoration" useful, users with cognitive disabilities require a simpler interface with less visual distraction.

- **Customizing dialog boxes -** Commonly supported by macro or scripting language.
- **Customizable menus / Customizable keyboard shortcuts -** See "Keyboard Access" in this document.

Comment [MJH57]: Does accompanying text refer to a Tooltip or text label next to a button?

Layout

Several visual design and layout guidelines improve an application's usability and accessibility for people with cognitive or visual impairments.

This section covers:

- Naming Controls
- Labeling Icons
- Related Item Proximity

Naming Controls

Accessibility aids can identify buttons and other text labeled controls, but edit controls or graphical objects are typically labeled by a nearby static control.

To make controls easier to use:

- Place a text label above, or to the immediate left of each control.
- Keep a control and its label close together.
- Set the Tab order so that the static text control immediately precedes the control it labels.
- Do not place unlabeled controls beneath or to the left of a label.
- Every text label should end with a colon.
- Static text controls that do not label other controls should *not* end with a colon.
- Do not use punctuation to create lines, boxes, decorative embellishments, or ASCII "art".
- Do not include non-functional (decorative) buttons or unlabeled hotspots on the screen.

Labeling Icons

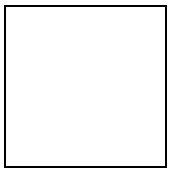

Figure 10 - Labels correctly positioned beneath large icons and to the right of small icons. A ToolTip is shown hovering over a graphical push button.

Icons, and graphics that represent an object or control, should always be displayed with a text label. This text/graphic convention is familiar to most users, helps new users learn application options quickly, and provides information to screen review utilities.

These are the standard guidelines:

- Place text labels directly beneath large icons and immediately to the right of small icons.
- Use Control Panel defined font, size, and color for title labels beneath icons.
- Expose text labels that cannot be displayed visibly, with ToolTips or other comparable techniques. See "Exposing Screen Elements" in this document.

Example: Start Windows Explorer from the Start menu and choose Large Icons from the View menu.

Related Item Proximity

Arrange related items close to each other. People with tunnel vision or low vision who use a screen magnification utility, see only a portion of the screen. Grouping related items in close proximity eliminates the need to scan for related items and makes these associations clearer for all users.

Section 4 Summary of Recommendations

The following is an overview of software techniques to make applications accessible to people with disabilities.

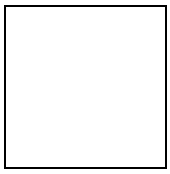

Figure 11 – Illustration of box with checkmark

Basic Principles

Follow these basic principles when designing an accessible application:

- **Flexibility –** Provide a flexible, customizable user interface (UI) that accommodates user needs and preferences.

  Example: Allow users to choose font sizes, reduce visual complexity, and customize the menu arrangement.

- **Choice of input methods –** Support input method choice by providing keyboard access to all features and simple mouse click access for common tasks.

- **Choice of output methods –** Provide a choice of output methods by supporting discrete and redundant combinations of sound, visuals, visual text, and graphics.

- **Consistency –** Design the application to interact consistently with other applications and system standards.

  Example: Support Control Panel settings for colors and sizes and use standard keyboard behavior.

- **Compatibility with accessibility aids -** Use programming techniques and user-interface elements compatible with accessibility aids (blind access, screen magnification, and voice-input utilities).

Requirement and recommendation ratings are, from low (✯) to high (✯✯✯✯), are based on:

- The number of people affected by the adaptation
- The impact an adaptation has on software usability
- The ease with which an application can handle the adaptation correctly

Keyboard Access

Providing an efficient keyboard user interface is crucial to accessible design.

- Provide keyboard access to all features. ✯✯✯✯ (Logo Requirement)
- Fully document the keyboard UI. ✯✯✯✯ (Logo Requirement)

Comment [GCL58]: Page: 47
12/29/98: I think these are revised elsewhere.

Comment [MJH59]: 6/9/99 Unclear what other system standards we are referring to.

Comment [GCL60]: Page: 47
12/30/98: Consider dividing these into subheadings for Required and Optional.

- Do not trigger unexpected side effects when the keyboard focus is moved. ✯✯✯
- Provide compatibility with the keyboard UI of analogous controls. ✯✯
- Provide underlined access keys for all menu items and controls. ✯✯
- Use logical keyboard navigation order. ✯✯
- Display normally hidden keyboard UI elements when Keyboard Preference flag is set. ✯✯
- Allow users to select and copy text with the keyboard everywhere it can be selected with a mouse. ✯✯
- Allow the user to select and copy text with the keyboard everywhere text is displayed. ✯
- Avoid using the **GetAsynchKeyState** function. ✯
- Where possible, provide customizable keyboard shortcuts. ✯
- Trigger standard error sound event whenever keyboard input is invalid or fails. ✯

Comment [GCL61]: Page: 48
12/29/98: Might be able to drop this one, GeorgA checking with the ISVs.

Keyboard Focus Location

Many accessibility aids need to be able to determine where the user is working.

- Expose the keyboard focus area within a window, either by moving the system caret or by using Active Accessibility. ✯✯✯✯ (Logo Requirement)
- Provide visual indication of the keyboard focus location. ✯✯✯
- Ensure onscreen indication of focus location is easy to see. ✯✯
- Ensure visual indication of the focus location is compatible with corresponding controls. ✯

Exposing Screen Elements

Many accessibility aids need to identify and manipulate objects on the screen.

- Use standard window classes and controls where possible because they automatically support Microsoft Active Accessibility. Implement Active Accessibility for custom UI elements. ✯✯✯✯
- Ensure all objects, windows, and graphics are properly named, define correct text labels for all controls, and give all windows a user-friendly caption, even if the text is not visible on the screen. ✯✯✯✯
- Expose names of owner-drawn menu items using Active Accessibility, or provide an alternative to any owner-drawn menus. ✯✯✯✯
- Expose names or descriptions for all images and bitmapped text. ✯✯✯✯
- Ensure dialog boxes define the correct tab order. ✯✯✯
- Assign unique identities to every type of window. ✯✯

Comment [GCL62]: Page: 48
12/29/98: Sort sub bullets by priority.

- Provide hints for screen elements not exposed using Active Accessibility. ★★
- If screen contents are not exposed in other ways, support standard drawing techniques that can be monitored and recorded. Provide alternatives to operations that manipulate bitmaps or screen pixels directly. ★★
- Display text using appropriate read-write edit, read-only edit, rich edit, status, static, or HTML controls. ★
- Assign objects unique labels within their context that are unambiguous when taken out of context. ★
- Provide access to screen elements using an application-specific object model when it provides more functionality than available through Active Accessibility. ★
- If not using Active Accessibility, support the WM_GETDLGCODE message in all custom controls that have a window to identify the control type and keyboard interface. ★

Comment [MJH63]: 6/11/99 Added *Rich Edit*.

Comment [GCL64]: Page: 49
12/29/98: Be sure that MSAA 2.0 handles this case better than 1.x.

Color

Color should be used to enhance, emphasize, or reiterate information.

- The application must conform when the High Contrast option is True. ★★★★ (Logo Requirement)
  - Menus and dialog boxes must be displayed using the current user-chosen Control Panel color scheme. This also applies to all UI elements required to adjust colors in the application's UI.
  - Users must be able to adjust the application's window display colors.
  - This option must override the application or document color display without altering document content or affecting other users.
  - The preferred method is to use the currently chosen Control Panel color scheme, but the application may also provide secondary display options.
  - When using Control Panel chosen colors or secondary display options, always draw foreground objects in foreground colors and fill backgrounds with the corresponding background color.
  - Provide all visual information by means other than, or in addition to, color.
  - Omit any images or patterns drawn behind text. All important information in such backgrounds must be available by another means.
- Always use colors in proper foreground/background combinations. ★★★
- Avoid conveying important information by color alone, or make it optional. ★★★
- Draw graphic objects to contrast with the current background color. ★★★
- Provide an option to omit complex or shaded backgrounds drawn behind text. ★★

Comment [GCL65]: Page: 49
12/29/98: Exceptions for small or temporary.

Comment [MJH66]: 6/9/99 Small or temporary colors?

- Where possible, allow users to customize all colors through Control Panel or application UI without setting the system-wide High Contrast option. ★★
- Where possible, draw monochrome images or textual equivalents that contrast with the background color. ★★
- When screen elements correspond with standard elements, use appropriate system colors chosen in Control Panel. ★

Sound

- Do not convey important information by sound exclusively, provide a visual presentation option. ★★★★
- Display important information visually when the ShowSounds option is True. ★★★
- Provide closed captions for all audio content exceeding three seconds. ★★
- Trigger standard sound events when performing equivalent actions. ★★
- Play audio as a user-customizable event through the Control Panel's Sounds section. ★
- Define many custom sound events, even if silent in the default sound scheme. ★
- If the application generates sounds, provide a way to turn them off. ★

Timings

- Allow users to customize all UI timings. ★★
  (a) Provide an option for preventing message time outs. ★★
  (b) Provide an option for slowing or disabling any rapid screen updates or flashing images. ★★

Size

The size of text and graphics affects usability as well as accessibility.

- Applications must be compatible with system settings for sizes and fonts. ★★★ (Logo Requirement)
- Avoid hard coding font sizes smaller than 10 points. ★★★
- Allow users to select font sizes for displayed information. ★★
- Ensure the application is compatible with changes to the system font size and number of pixels per logical inch. ★★
- If drawing lines, determine the appropriate width rather than using a fixed value. ★
- Allow users to adjust the size of non-document elements such as toolbars. ★
- Where possible, provide draft mode, zoom, and wrap to window features. ★
- Stretch, shrink, pad or crop images when space availability changes. ★

Comment [GCL67]: Page: 50
12/29/98: Clarify that several of these are only applicable when the High Contrast option is off.

Comment [MJH68]: 6/9/99 I will need to have ore info on which and how.

Comment [GCL69]: Page: 50
12/29/98: Is there a global mute system setting?

Comment [GCL70]: Page: 50
12/29/98: Move that to the separate Fonts section.

Comment [GCL71]: Page: 50
12/29/98: Note that this is very important for fixed-pixel flat-panel displays.

Comment [GCL72]: Page: 50
12/29/98: Need a new system metric for this.

Mouse Input

- Mouse movement should not trigger unexpected side effects. ✯✯
- Applications must be compatible with specified system settings for mouse input. ✯✯ (Logo Requirement)
- Provide mouse shortcuts for commonly used features. ✯
- Provide for customizable toolbars. ✯
- Emphasize simple, single click mouse operations. ✯

Comment [GCL73]: Page: 51
12/29/98: Moved here from older Unexpected Side Effects section.

Customizable User Interface

- Where possible, allow users or administrators to customize the application to meet specific needs. ✯
- Provide a method for users to automate tasks and assign them to simple inputs. ✯

Fonts

- Support user-selected font settings through the Control Panel. ✯✯
- Allow user-adjusted font face and style display without affecting shared documents or other user preferences. ✯

Layout

Efficient visual design and layout makes an application more usable and accessible for people with cognitive or visual impairments:

- Position the label so that it immediately precedes the control it labels in Tab order.
- Provide easily recognizable labels for each control or object. ✯✯
  (a) Place the text label above or immediately to the left of the control.
  (b) Keep the control and its label close together.
  (c) Do not place unlabeled controls beneath or to the left of a label.
  (d) Do not place a text label above one control and to the left of another.
  (e) All text labels should end with colons.
  (f) Static text controls that do not label other controls should not end in colons.
  (g) Follow conventions for positioning labels of icons and controls. ✯
- Position related objects close to each other. ✯
- Position objects on the screen or within a window according to Windows UI design conventions. ✯
- Do not use punctuation to create lines, boxes, decorative embellishments, ASCII "art" or emoticons. ✯
- Do not include non-functional (decorative) buttons or unlabeled hotspots. ✯

Documentation

- All application generated reports must be available in text file formts.

Comment [GCL74]: Page: 51
12/29/98: DickB owns updating this section.

- Provide documentation in an accessible format, such as ASCII text or HTML.
- Include descriptions of illustrations and tables in all accessible documentation.
- Use color and graphics redundantly with text. Do not convey important information by either color or graphics alone.
- Maintain high contrast between text and background.
- Do not use text smaller than 10 points.
- Where possible, bind printed documentation to lie flat.

Verifying Accessibility

Test the application against this checklist:

- Test with the High Contrast option and High Contrast appearance schemes.
- Test compatibility with extra-large appearance schemes.
- Ensure all features can be used without a mouse through documented methods.
- Test with the Inspect Objects tool and Windows Narrator accessory to verify that all screen elements are exposed and properly labeled.
- Test with the Microsoft Magnifier to verify that the keyboard focus location is properly exposed during navigation and editing. This accessory is included with Windows 98 and Windows 2000 operating systems and in the Active Accessibility Software Development kit..
- Test with commercial accessibility aids. See http://microsoft.com/enable/ for a list of these aids.
- Test with changes to the system font size and number of pixels per logical inch using the Control Panel's Display Settings option.
- Include people with disabilities and accessibility software vendors in beta tests.
- Include people with disabilities in application usability tests.
- Conduct surveys of application users who have disabilities.
- Distribute free evaluation copies of the application to individuals with disabilities, disability organizations, and accessibility software vendors.

Section 5 Testing Techniques

Prior planning alone will not assure an application is as accessible as it could be. Several techniques can determine an application's accessibility:

Testing Techniques

Test an application's accessibility by following these suggestions:

- **Use the recommendation list -** Have several disparate user/testers compare the application to this document's list of recommendations. All major features should be tested systematically, but informal testing can also detect many of the most obvious problems.
- **Large, high-contrast appearance schemes -** Choose the Accessibility Options section of Control Panel, choose the Display page, select the High Contrast option and choose the Settings dialog box "White on Black" scheme.
  - Do any parts of the application become invisible, hard to recognize, or difficult to use?
  - Do any areas continue to appear black on a white background?
  - Are any elements improperly sized or truncated?
- **Keyboard.** Disconnect the mouse and use keyboard input exclusively to operate all application features for a minimum of five workday-long sessions.
  - Are there operations that cannot be performed?
  - Are any operations awkward or difficult to perform?
  - Are keyboard mechanisms documented adequately?
  - Do all controls and menu items have underlined access keys?
- **Exposing screen elements -** Use the Active Accessibility SDK's Inspect Objects tool while hovering the mouse pointer over various application objects. Make sure the tool displays a proper description that matches standard windows and controls.
- **Exposing the keyboard focus location -** Use the Active Accessibility SDK's Magnifier accessory while working with the keyboard exclusively to make certain the work area appears magnified, in the Magnifier window. This aid will be a standard accessory with future versions of Windows and Windows NT.
- **Compatibility with accessibility aids -** Operate the application configured with available accessibility aids to determine if they work properly with the application. Many utilities are available at no charge or in trial versions. See http://microsoft.com/enable/. for a catalog of these tools.
- **Larger system font -** Increase the size of the system font using the Display section of Control Panel.
  - Does the application display correctly despite the changes?
  - Can all application fonts be adjusted to be at least as large as the system font?

- **Custom font sizes -** Change the display scaling ratio using the Custom Font Size feature in the Display section of Control Panel.
  - Does the application appearance remain consistent?
  - Do various elements of the UI appear disproportionately large or small?

Prioritizing Features

If you prioritize features to be tested, consider the following evaluation factors:

- **Number of users -** Assign higher priority to features that affect the largest number of users. Generally, document viewing features affect more people than document authoring features.
- **Frequency of use -** Assign higher priority to features chosen frequently by users who need them.
- **Urgency of use –** Assign higher priority to features that are integral, necessary parts of the product, rather than optional or "ease-of-use" features.

Getting Feedback from Customers

- To best determine if a product is truly usable by people with disabilities, actively solicit feedback, suggestions and complaints from users with disabilities.
- **Beta tests –** Include people with disabilities and accessibility aid developers in all field or beta product tests.
- **Usability tests –** When performing usability tests, include subjects who have disabilities. It is unnecessary to design special tests for these subjects. This unmodified test platform can provide valuable insight into the way they perform routine tasks - a very informative process - revealing the different ways people approach a task.
- **Surveys –** Request design input from users who have disabilities. This information may currently be available to application builders, but is "filtered" before reaching the development team. The Internet is a powerful resource for finding and consulting computer users with specific disabilities.
- **Evaluation programs –** Distribute free evaluation copies of the application to organizations that represent, work with or develop accessibility aids for people with disabilities and request feedback.

Include accessibility aid manufacturers in all the above processes to determine if their product is compatible with the application or in need of configuration files or other modifications. Industry affiliations also provide valuable technical suggestions for improving the application's accessibility.

See http://microsoft.com/enable/ for a list of accessibility aid manufacturers.

Appendix A - Additional Resources

For additional information about accessibility, a catalog of accessibility aids, and resources for people with disabilities, see Microsoft's Accessibility site on the World Wide Web: http://microsoft.com/enable/

Or contact:

Microsoft Sales Information Center
One Microsoft Way, Redmond, WA 98052-6393
World Wide Web: http://microsoft.com
Voice telephone: (800) 426-9400
Text telephone: (800) 892-5234

Additional Accessibility Guidelines

This document is based on general guidelines first proposed in the white paper "Making Software More Accessible for People with Disabilities." prepared by Gregg Vanderheiden of the Trace R&D Center of the University of Wisconsin. Research funding was provided by the Information Technology Foundation (formerly ADAPSO Foundation) and the National Institute for Disability and Rehabilitation Research (NIDRR) of the U.S. Department of Education. That paper and similar guidelines for other types of products are available on the *CO-NET CD* or in print from:

Trace R&D Center - University of Wisconsin-Madison
5901 Research Park Boulevard, Madison, WI 53719-1252
World Wide Web: http://trace.wisc.edu/
Fax: (608) 262-8848

The Windows Interface Guidelines for Software Design contains a section on accessible software design. It is available online at http://msdn.microsoft.com and in book form through Microsoft Press.

Appendix B - Accessible Documentation

Applications are not considered accessible if a user cannot learn to use it.

Documentation in Accessible Formats

Some users have difficulty reading or holding conventionally printed documentation, so documentation should also be provided in more accessible formats: online, multimedia, or CD-ROM tutorials.

Inform the user if online documentation is available that covers the same information as the print version. Users should be able to determine easily if online documentation is available, how complete it is, and how to obtain it. Of course, the application presenting the documentation itself must be accessible.

The following formats are acceptable for accessible documentation:

- **HTML Help.** Microsoft is currently developing HTML Help, the next-generation of help systems for the Windows operating systems. The HTML Help system will be completely compatible with accessibility aids and take advantage of accessibility features in Microsoft Internet Explorer. Documentation developed for this system is accessible to users with disabilities through WinHelp.
- **Text formats.** Vendors should offer customers documentation in electronic form, usually provided as formatted ASCII text files, or as HTML files designed for the World Wide Web.

These formats address a wide variety of needs.

Example: Customers who are blind or have low vision can read the files in their own word processor using screen review or screen enlarger utilities. Customers with mobility impairments can read them online without holding or turning the pages of a physical book. HTML files normally include special tags that identify the structure of the document (that is, tags for headings, footnotes, and so on) and are routinely included as part of all HTML files.

- **Specialized formats.** Documentation can also be provided in large print, Braille, or audiotapes. These options are not usually provided directly by the software company, but licensed as source files to users or organizations that create accessible versions in those formats.
- All of the above formats should provide illustration descriptions. All formats other than HTML or HTML Help should also provide descriptions of table structures.

- **Designing Accessible Documentation**
  Accessible documentation design follows the same rules as accessible software visual design:
  - **Do not convey information by color or graphics exclusively.** If printed documentation relies on color or graphics to convey important information, that information will be available to a limited number of users. Some users may rely on a variety of devices to enlarge a document or translate it into ASCII text, speech, or Braille and these devices are often unable to interpret and convey graphic or color-coded information.
  - **Color and graphics should be added redundantly to the text to improve documents.**

    Example: If a reference work contains a function call list and details important information about each one, some entries can be printed as blue, rather than black text, denoting instantly that they are not supported on all systems. In this case, all the entries that are shown in blue should also include a phrase, such as "platform specific" in their description. If space is limited, each can simply be marked with an asterisk and footnoted.

    - Redundant information display often makes documentation easier for everyone to use. A phrase or asterisk can also be used to call out certain paragraphs with a graphic in the margin. This modified text makes it easier to translate your documentation into alternative formats, such as Braille or an online document.
  - **Maintain high contrast between text and its background and avoid screened art behind text.**
  - **Do not use text smaller than 10 points in size**.
  - **Bind printed documents to lie flat.** Comb and spiral bindings are considered most accessible because they allow a document to lie flat. These bindings are useful for people with motion or visual impairments.

    Example: A person who is quadriplegic may open the book flat and turn the pages with a pencil. A person who is blind may run it through a flatbed scanner to use optical-character recognition for conversion to an online format. A person with low vision might use a closed-caption television system to enlarge the pages. People who want to type while reading also prefer flat bindings.

Appendix C - Accessible Packaging

This appendix describes accessible packaging for software:

- **Provide easy-to-identify diskettes.** All diskettes and CD-ROM disks should be given a unique volume label that easily identifies the specific product and disk number. People who are blind may not be able to read the printed disk label, but providing an appropriate volume label allows them to identify the diskette using the **dir** command at the command prompt.
- **Provide easy-to-open packaging.** Users with mobility impairments may have trouble opening some packaging. Evaluate existing packaging to determine if it could be made easier to use. Example: Shrink-wrapped packages can be easy to open if they are left unsealed along a seam where two layers overlap.
- **Don't require users to read the packaging.** If your setup procedure requires the user to read installation instructions, or a product identification code that is only available in print, it can be impossible for users with visual impairments to install the software without assistance.

Appendix D - Product Support and Customer Service

The following guidelines can help when providing effective technical support and product service for customers with disabilities.

- **Provide customer support through text telephone and modem.** Customers who are deaf or hard-of-hearing or who have speech impairments may not be able to use standard voice telephones to access customer information and support services. These services should, therefore, be made available through a text telephone (also known as TTY or TDD) and standard ASCII modems. Stand-alone text telephones are available with a wide range of features, and combination TTY/ASCII modems can also be attached to standard computers, although specialized software is normally used to get full answering-machine functionality.
- **Train your service representatives.** You should provide your customer service and technical support representatives with guidelines for interacting with people with disabilities. Having a basic familiarity with types of disabilities, the way in which people with disabilities use computers, and the very fact that they *do* use computers, can go a long way towards making customer interactions successful and positive.
- **Address customers with disabilities.** Some companies, including Microsoft, incorporate information about accessibility into every product, and into their sites on the World Wide Web. This can help establish a positive relationship with the disability community and make customers with disabilities feel more comfortable in dealing with your organization.

Appendix E – Benefits of Accessible Software

Appendix F - Exercises

Appendix G - Windows Version 3.*x* Guidelines

The following guidelines do not affect Win32®-based applications designed for Windows 95, Windows 98, Windows 2000 or Windows NT, but should be followed when developing 16-bit applications (also called Win16 applications) for Windows version 3.*x*.

Yielding Control to Background Applications

Windows-based 16-bit applications should yield control at all times so other programs, such as accessibility aids, can run in the background. If a program refuses to yield control, the user is unable to access the machine. You can avoid access problems by following these techniques:

- **Avoid using system modal dialog boxes or windows.** When a system-modal window is active, no background tasks are allowed to run. (This is true to a lesser extent for 16-bit applications running under Windows 95, although it is not a problem for those running under Windows NT.)
- **Avoid using the** PeekMessage **function in tight loops without yielding**.

Colors in Online Help

An author designing online help topics can specify foreground and background colors, or use the color scheme selected by the user in Control Panel. If the author specifies his or her own color scheme, the user running Windows version 3.*x* or Windows NT version 3.*x* has no way to override the scheme and use a different set of colors. As a consequence, some users who require high-contrast color schemes will not be able to make use of the help topic. (In Windows 95, the help system provides the user with the option to use only Control Panel colors. However, this option leads to having the topic appear different from the help author's preference.)

If you want your online help to be usable by as many customers as possible, you should generally allow the user to choose their own color scheme rather than specifying one of your own choosing.

Testing Accessibility Flags

Windows 95 introduced four new flags that notify applications when to adjust to accommodate users with disabilities. Although each can be tested using the **SystemParametersInfo** function, this capability is not supported in earlier versions of Windows or Windows NT. To make this behavior available in earlier operating systems, you can test for the flags as WIN.INI settings. Windows does not use these settings, but users who want specified disability options can set them manually. The following WIN.INI settings are recommended.

SystemParametersInfo	WIN.INI setting in earlier operating systems
SPI_SHOWSOUNDS	[Windows] ShowSounds=TRUE
SPI_KEYBOARDPREF	[Windows] KeyboardPref=TRUE
SPI_SCREENREADER	[Windows] ScreenReader=TRUE
SPI_HIGHCONTRAST	[Windows] HighContrast=TRUE