

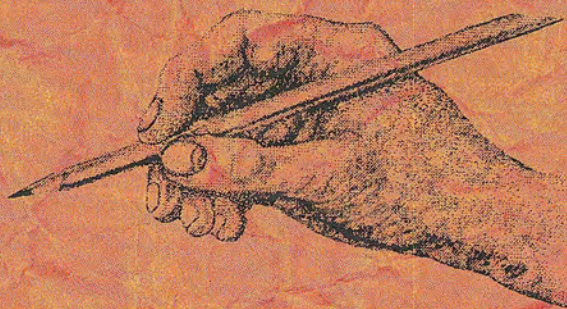


Microsoft® Professional Reference

Covers  
Windows 95 &  
Windows NT™!



# The Windows® Interface Guidelines for Software Design







# **The Windows<sup>®</sup> Interface Guidelines for Software Design**



PUBLISHED BY  
Microsoft Press  
A Division of Microsoft Corporation  
One Microsoft Way  
Redmond, Washington 98052-6399

Copyright © 1995 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Cataloging-in-Publication Data  
The Windows interface guidelines for software design.

p. cm.

Includes index.

ISBN 1-55615-679-0

1. Microsoft Windows (Computer file) 2. User interfaces (Computer systems) 3. Computer software--Development. I. Microsoft Corporation.

QA76.76.W56W553 1995

005.265--dc20

95-330

CIP

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QEQE 0 9 8 7 6 5

Distributed to the book trade in Canada by Macmillan of Canada, a division of Canada Publishing Corporation.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office. Or contact Microsoft Press International directly at fax (206) 936-7329.

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property rights.

Adobe, PostScript, and TIFF are trademarks of Adobe Systems, Inc. Apple and TrueType are registered trademarks of Apple Computer, Inc. Borland and Quattro are registered trademarks of Borland International, Inc. Frutiger is a registered trademark of Eltra Corporation. HP and LaserJet are registered trademarks of Hewlett-Packard Company. Backup was developed for Microsoft by Colorado Memory Systems, Inc., a division of Hewlett-Packard Company. HyperTerminal is a trademark of Hilgraeve, Inc. 1-2-3 and Lotus are registered trademarks of Lotus Development Corporation. Microsoft, Microsoft Press, Microsoft Press logo, MS, MS-DOS, PowerPoint, Visual Basic, Windows, Windows logo, and XENIX are registered trademarks and Windows NT is a trademark of Microsoft Corporation. Arial, Bodoni, Swing, and Times New Roman are registered trademarks of The Monotype Corporation PLC. Paintbrush is a trademark of Wordstar Atlanta Technology Center.



# Contents

## Introduction

What's New .....	xv
How to Use This Guide .....	xvi
How to Apply the Guidelines .....	xvii
Conventions Used in This Guide .....	xviii

## PART I FUNDAMENTALS OF DESIGNING USER INTERACTION

### Chapter 1 Design Principles and Methodology

User-Centered Design Principles .....	3
User in Control .....	3
Directness .....	4
Consistency .....	5
Forgiveness .....	6
Feedback .....	7
Aesthetics .....	7
Simplicity .....	8
Design Methodology .....	9
A Balanced Design Team .....	9
The Design Cycle .....	9
Usability Assessment in the Design Process .....	12
Understanding Users .....	15
Design Tradeoffs .....	16



**Chapter 2 Basic Concepts**

Data-Centered Design ..... 17

Objects as Metaphor ..... 18

    Object Characteristics ..... 18

    Relationships ..... 19

    Composition ..... 19

    Persistence ..... 20

Putting Theory into Practice ..... 20

**Chapter 3 The Windows Environment**

The Desktop ..... 23

The Taskbar ..... 24

    The Start Button ..... 25

    Window Buttons ..... 26

    The Status Area ..... 26

Icons ..... 26

Windows ..... 28

**Chapter 4 Input Basics**

Mouse Input ..... 29

    Mouse Pointers ..... 29

    Mouse Actions ..... 31

Keyboard Input ..... 32

    Text Keys ..... 33

    Access Keys ..... 33

    Mode Keys ..... 34

    Shortcut Keys ..... 35

Pen Input ..... 37

    Pen Pointers ..... 39

    Pen Gestures ..... 40

    Pen Recognition ..... 41

    Ink Input ..... 41

    Targeting ..... 42



**Chapter 5 General Interaction Techniques**

Navigation ..... 43

    Mouse and Pen Navigation ..... 44

    Keyboard Navigation ..... 44

Selection ..... 45

    Selection Feedback ..... 46

    Scope of Selection ..... 47

    Hierarchical Selection ..... 47

    Mouse Selection ..... 48

    Pen Selection ..... 55

    Keyboard Selection ..... 56

    Selection Shortcuts ..... 57

Common Conventions for Supporting Operations ..... 58

    Operations for a Multiple Selection ..... 58

    Default Operations and Shortcut Techniques ..... 59

    View Operations ..... 59

Editing Operations ..... 62

    Editing Text ..... 62

    Handles ..... 63

    Transactions ..... 64

    Properties ..... 65

    Pen-Specific Editing Techniques ..... 66

Transfer Operations ..... 72

    Command Method ..... 74

    Direct Manipulation Method ..... 77

    Transfer Feedback ..... 83

    Specialized Transfer Commands ..... 86

    Shortcut Keys for Transfer Operations ..... 87

Creation Operations ..... 87

    Copy Command ..... 87

    New Command ..... 88

    Insert Command ..... 88

    Using Controls ..... 88

    Using Templates ..... 88

Operations on Linked Objects ..... 89



**PART II WINDOWS INTERFACE COMPONENTS**

**Chapter 6 Windows**

Common Types of Windows ..... 95

Primary Window Components ..... 95

    Window Frames ..... 96

    Title Bars ..... 96

    Title Bar Icons ..... 97

    Title Text ..... 98

    Title Bar Buttons ..... 101

Basic Window Operations ..... 103

    Activating and Deactivating Windows ..... 103

    Opening and Closing Windows ..... 104

    Moving Windows ..... 106

    Resizing Windows ..... 106

    Scrolling Windows ..... 109

    Splitting Windows ..... 116

**Chapter 7 Menus, Controls, and Toolbars**

Menus ..... 121

    The Menu Bar and Drop-down Menus ..... 121

    Common Drop-down Menus ..... 124

    Pop-up Menus ..... 126

    Pop-up Menu Interaction ..... 128

    Common Pop-up Menus ..... 129

    Cascading Menus ..... 133

    Menu Titles ..... 134

    Menu Items ..... 135

Controls ..... 140

    Buttons ..... 141

    List Boxes ..... 149

    Text Fields ..... 157

    Other General Controls ..... 163

    Pen-Specific Controls ..... 169

Toolbars and Status Bars ..... 172

    Interaction with Controls in Toolbars and Status Bars ..... 173

    Support for User Options ..... 174

    Toolbar and Status Bar Controls ..... 175

    Common Toolbar Buttons ..... 176

**Chapter 8 Secondary Windows**

Characteristics of Secondary Windows ..... 179

    Appearance and Behavior ..... 179

    Window Placement ..... 183

    Modeless vs. Modal ..... 183

    Default Buttons ..... 184

    Navigation in Secondary Windows ..... 185

    Validation of Input ..... 187

Property Sheets and Inspectors ..... 187

    Property Sheet Interface ..... 188

    Property Sheet Commands ..... 190

    Closing a Property Sheet ..... 191

    Property Inspectors ..... 191

    Properties of a Multiple Selection ..... 192

    Properties of a Heterogeneous Selection ..... 193

    Properties of Grouped Items ..... 193

Dialog Boxes ..... 193

    Dialog Box Commands ..... 194

    Layout ..... 194

    Common Dialog Box Interfaces ..... 195

Palette Windows ..... 207

Message Boxes ..... 209

    Title Bar Text ..... 209

    Message Box Types ..... 209

    Command Buttons in Message Boxes ..... 212

    Message Box Text ..... 213

Pop-up Windows ..... 215



**PART III DESIGN SPECIFICATIONS AND GUIDELINES**

**Chapter 9 Window Management**

Single Document Window Interface ..... 219

Multiple Document Interface ..... 220

    Opening and Closing MDI Windows ..... 222

    Moving and Sizing MDI Windows ..... 223

    Switching Between MDI Child Windows ..... 225

MDI Alternatives ..... 225

    Workspaces ..... 227

    Workbooks ..... 229

    Projects ..... 231

Selecting a Window Model ..... 233

    Presentation of Object or Task ..... 233

    Display Layout ..... 234

    Data-Centered Design ..... 235

    Combination of Alternatives ..... 235

**Chapter 10 Integrating with the System**

The Registry ..... 237

    Registering Application State Information ..... 238

    Registering Application Path Information ..... 241

    Registering File Extensions ..... 242

    Supporting Creation ..... 249

    Registering Icons ..... 250

    Registering Commands ..... 251

    Enabling Printing ..... 253

    Registering OLE ..... 253

    Registering Shell Extensions ..... 254

    Supporting the Quick View Command ..... 256

    Registering Sound Events ..... 257

Installation ..... 257

    Copying Files ..... 257

    Providing Access to Your Application ..... 260

    Designing Your Installation Program ..... 260

Installing Fonts ..... 262  
 Installing Your Application on a Network ..... 262  
 Uninstalling Your Application ..... 263  
 Supporting AutoPlay ..... 264  
 System Naming Conventions ..... 266  
 Taskbar Integration ..... 268  
     Taskbar Window Buttons ..... 268  
     Status Notification ..... 269  
     Message Notification ..... 270  
 Application Desktop Toolbars ..... 271  
 Full-Screen Display ..... 272  
 Recycle Bin Integration ..... 273  
 Control Panel Integration ..... 273  
     Adding Control Panel Objects ..... 273  
     Adding to the Passwords Object ..... 273  
 Plug and Play Support ..... 275  
 System Settings and Notification ..... 275  
 Modeless Interaction ..... 276

**Chapter 11 Working with OLE Embedded and OLE Linked Objects**

The Interaction Model ..... 277  
 Creating OLE Embedded and OLE Linked Objects ..... 279  
     Transferring Objects ..... 279  
     Inserting New Objects ..... 285  
 Displaying Objects ..... 290  
 Selecting Objects ..... 293  
     Accessing Commands for Selected Objects ..... 295  
 Activating Objects ..... 297  
     Outside-in Activation ..... 297  
     Inside-out Activation ..... 297  
     Container Control of Activation ..... 298  
 OLE Visual Editing of OLE Embedded Objects ..... 300  
     The Active Hatched Border ..... 304



- Menu Integration ..... 305
- Keyboard Interface Integration ..... 308
- Toolbars, Frame Adornments, and Palette Windows ..... 310
- Opening OLE Embedded Objects ..... 313
- Editing an OLE Linked Object ..... 316
  - Automatic and Manual Updating ..... 318
  - Operations and Links ..... 319
  - Types and Links ..... 320
  - Link Management ..... 320
- Accessing Properties of OLE Objects ..... 321
  - The Properties Command ..... 321
  - The Links Command ..... 324
- Converting Types ..... 326
- Using Handles ..... 329
- Undo Operations for Active and Open Objects ..... 330
- Displaying Messages ..... 332
  - Object Application Messages ..... 332
  - OLE Linked Object Messages ..... 334
  - Status Line Messages ..... 336

## Chapter 12 User Assistance

- Contextual User Assistance ..... 339
  - Context-Sensitive Help ..... 339
  - Guidelines for Writing Context-Sensitive Help ..... 342
  - Tooltips ..... 343
  - Status Bar Messages ..... 344
  - Guidelines for Writing Status Bar Messages ..... 345
  - The Help Command Button ..... 346
- Task-Oriented Help ..... 347
  - Task Topic Windows ..... 347
  - Guidelines for Writing Task Help Topics ..... 348
  - Shortcut Buttons ..... 349

Reference Help ..... 350  
 The Reference Help Window ..... 351  
 Guidelines for Writing Reference Help ..... 352  
 The Help Topics Browser ..... 354  
 The Help Topics Tabs ..... 354  
 Wizards ..... 358  
 Guidelines for Designing Wizards ..... 359  
 Guidelines for Writing Text for Wizard Pages ..... 363

**Chapter 13 Visual Design**

Visual Communication ..... 365  
 Composition and Organization ..... 366  
 Color ..... 368  
 Fonts ..... 370  
 Dimensionality ..... 371  
 Design of Visual Elements ..... 372  
 Basic Border Styles ..... 372  
 Window Border Style ..... 373  
 Button Border Styles ..... 374  
 Field Border Style ..... 375  
 Status Field Border Style ..... 376  
 Grouping Border Style ..... 376  
 Visual States for Controls ..... 377  
 Layout ..... 384  
 Font and Size ..... 384  
 Capitalization ..... 387  
 Grouping and Spacing ..... 388  
 Alignment ..... 389  
 Placement ..... 389  
 Design of Graphic Images ..... 390  
 Icon Design ..... 391  
 Pointer Design ..... 394  
 Selection Appearance ..... 395  
 Highlighting ..... 396  
 Handles ..... 397  
 Transfer Appearance ..... 398  
 Open Appearance ..... 399  
 Animation ..... 400



**Chapter 14 Special Design Considerations**

Sound ..... 401

Accessibility ..... 403

    Types of Disabilities ..... 404

    Types of Accessibility Aids ..... 406

    Compatibility with Screen Review Utilities ..... 408

    The User’s Point of Focus ..... 411

    Timing and Navigational Interfaces ..... 411

    Color ..... 412

    Keyboard and Mouse Interface ..... 413

    Documentation, Packaging, and Support ..... 414

    Usability Testing ..... 414

Internationalization ..... 415

    Text ..... 416

    Graphics ..... 417

    Keyboards ..... 418

    Character Sets ..... 419

    Formats ..... 419

    Layout ..... 420

    References to Unsupported Features ..... 420

Network Computing ..... 421

    Leverage System Support ..... 421

    Client-Server Applications ..... 421

    Shared Data Files ..... 422

Record Processing ..... 422

Telephony ..... 423

Microsoft Exchange ..... 424

    Coexisting with Other Information Services ..... 424

    Adding Menu Items and Toolbar Buttons ..... 424

    Supporting Connections ..... 425

    Installing Information Services ..... 425

**PART IV APPENDIXES**

**Appendix A Mouse Interface Summary**

Interaction Guidelines for Common  
 Unmodified Mouse Actions ..... 429  
 Interaction Guidelines for Using the SHIFT Key  
 to Modify Mouse Actions ..... 431  
 Interaction Guidelines for Using the CTRL Key  
 to Modify Mouse Actions ..... 435

**Appendix B Keyboard Interface Summary**

Common Navigation Keys ..... 437  
 Common Shortcut Keys ..... 438  
 Windows Keys ..... 439  
 Accessibility Keys ..... 440  
 Access Key Assignments ..... 441

**Appendix C Guidelines Summary**

General Design ..... 443  
 Design Process ..... 444  
 Input and Interaction ..... 444  
 Windows ..... 445  
 Control Usage ..... 446  
 Integration ..... 447  
 User Assistance ..... 448  
 Visual Design ..... 448  
 Sound ..... 449  
 Accessibility ..... 449  
 International Users ..... 450  
 Network Users ..... 450



**Appendix D Supporting Specific Versions of Microsoft Windows**

Microsoft Windows 3.1 ..... 451  
Microsoft Windows NT 3.51 ..... 453

**Appendix E International Word Lists**

International Word Lists ..... 455

**Glossary** ..... 511

**Bibliography** ..... 523

**Index** ..... 527







# Introduction

Welcome to *The Windows Interface Guidelines for Software Design*, an indispensable guide to designing software that runs with the Microsoft® Windows® operating system. The design of your software's interface, more than anything else, affects how a user experiences your product. This guide promotes good interface design and visual and functional consistency within and across Windows-based applications.

## What's New

Continuing the direction set by Microsoft OLE, the enhancements in the Windows user interface provide a design evolution from the basic and graphical to the more object oriented — that is, from an application-centered interface to a more data-centered one. In response, developers and designers may need to rethink the interface of their software — the basic components and the respective operations and properties that apply to them. This is important because, from a user's perspective, applications have become less the primary focus and more the engines behind the objects in the interface. Users can now interact with data without having to think about applications, allowing them to better concentrate on their tasks.

When adapting your existing Windows-based software, make certain you consider the following important design aspects:

- Title bar text and icons
- Property sheets
- Transfer model (including drag and drop)



## Introduction

- Pop-up menus
- New controls
- Integration with the system
- Help interface
- OLE embedding and OLE linking
- Visual design of windows, controls, and icons
- Window management
- Presentation of minimized windows

These elements are covered in depth throughout this guide.

## How to Use This Guide

This guide is intended for those who are designing and developing Windows-based software. It may also be appropriate for those interested in a better understanding of the Windows environment and the human-computer interface principles it supports. The content of the guide covers the following areas:

- Basic design principles and process — fundamental design philosophy, assumptions about human behavior, design methodology, and concepts embodied in the interface.
- Interface elements — descriptive information about the various components in the interface as well as when and how to use them.
- Design details — specific information about the details of effective design and style when using the elements of the interface.
- Additional information — summary and quick reference information, a bibliography, a comprehensive word list in numerous languages to assist in product localization, and a glossary.

This guide focuses on the design and elements of an application's user interface. Although an occasional technical reference is included, this guide does not generally cover detailed information about technical implementation or application programming interfaces (APIs), because there are many different types of development tools that you can use to develop software for Windows. The documentation included with the Microsoft® Win32® Software Development Kit (SDK) is one source of information about specific APIs.

## How to Apply the Guidelines

This guide promotes visual and functional consistency within and across the Windows operating system. Although following these guidelines is encouraged, you are free to adopt the guidelines that best suit your software. However, by following these guidelines, you enable users to transfer their skills and experience from one task to the next and to learn new tasks easily. In addition, evolution toward data-centered design breaks down the lines between traditional application domains, making inconsistencies in the interface more obvious and distracting to users.

Conversely, adhering to the design guidelines does not guarantee usability. The guidelines are valuable tools, but they must be combined with other factors as part of an effective software design process, such as application of design principles, task analysis, prototyping, and usability evaluation.

You may extend these guidelines, provided that you do so in the spirit of the principles on which they are based, and maintain a reasonable level of consistency with the visual and behavioral aspects of the Windows interface. In general, avoid adding new elements or behaviors unless the interface does not otherwise support them. More importantly, avoid changing an existing behavior for common elements. A user builds up expectations about the workings of an interface. Inconsistencies not only confuse the user, they also add unnecessary complexity.





These guidelines supersede those issued for Windows version 3.1 and all previous releases and are specific to the development of applications designed for Microsoft® Windows®, Microsoft® Windows NT™ Workstation, and Microsoft® Windows NT Server. There is no direct relationship between these guidelines and those provided for other operating systems.

For more information about special considerations concerning developing applications for both Windows 95 and Windows NT operating system, see Appendix D, “Supporting Specific Versions of Windows.”

# Conventions Used in This Guide

The following conventions are used throughout this guide.

Convention	Indicates
	A reference to related topics in this guide or other books that provide more information about the topic.
	Additional or special information about the topic.
SMALL CAPITAL LETTERS	Names of keys on the keyboard — for example, SHIFT, CTRL, or ALT.
KEY+KEY	Key combinations for which the user must press and hold down one key and then press another — for example, CTRL+P or ALT+F4.
<i>Italic text</i>	New terms and variable expressions, such as parameters.
<b>Bold text</b>	Win32 API keywords and registry key entries.
Registry text	Examples of registry entries.
[ ]	Optional information.





# Part I

## Fundamentals of Designing User Interaction





# Design Principles and Methodology



A well-designed user interface is built on principles and a development process that centers on users and their tasks. This chapter summarizes the basic principles of the interface design for Microsoft Windows. It also includes techniques and methodologies employed in an effective human-computer interface design process.

## User-Centered Design Principles

The information in this section describes the design principles on which Windows and the guidelines in this book are based. You will find these principles valuable when designing software for Windows.

### User in Control

An important principle of user interface design is that the user should always feel in control of the software, rather than feeling controlled by the software. This principle has a number of implications.

The first implication is the operational assumption that the user initiates actions, not the computer or software — the user plays an active, rather than reactive, role. You can use techniques to automate tasks, but implement them in a way that allows the user to choose or control the automation.



The second implication is that users, because of their widely varying skills and preferences, must be able to personalize aspects of the interface. The system software provides user access to many of these aspects. Your software should reflect user settings for different system properties, such as color, fonts, or other options.

The final implication is that your software should be as interactive and responsive as possible. Avoid modes whenever possible. A *mode* is a state that excludes general interaction or otherwise limits the user to specific interactions. When a mode is the only or the best design alternative — for example, for selecting a particular tool in a drawing program — make certain the mode is obvious, visible, the result of an explicit user choice, and easy to cancel.

For information about applying the design principle of user in control, see Chapter 4, “Input Basics,” and Chapter 5, “General Interaction Techniques.” These chapters cover the basic forms of interaction your software should support.

## Directness

Design your software so that users can directly manipulate software representations of information. Whether dragging an object to relocate it or navigating to a location in a document, users should see how the actions they take affect the objects on the screen. Visibility of information and choices also reduce the user’s mental workload. Users can recognize a command easier than they can recall its syntax.

Familiar metaphors provide a direct and intuitive interface to user tasks. By allowing users to transfer their knowledge and experience, metaphors make it easier to predict and learn the behaviors of software-based representations.

When using metaphors, you need not limit a computer-based implementation to its “real world” counterpart. For example, unlike its paper-based counterpart, a folder on the Windows desktop can be used to organize a variety of objects such as printers, calculators, and other folders. Similarly, a Windows folder can be more easily re-sorted. The purpose of using metaphor in the interface is to provide a cognitive bridge; the metaphor is not an end in itself.

Metaphors support user recognition rather than recollection. Users remember a meaning associated with a familiar object more easily than they remember the name of a particular command.

For information about applying the principle of directness and metaphor, see Chapter 5, "General Interaction Techniques," and Chapter 13, "Visual Design." These chapters cover, respectively, the use of directness in the interface (including drag and drop) and the use of metaphors when designing icons or other graphical elements.

## Consistency

Consistency allows users to transfer existing knowledge to new tasks, learn new things more quickly, and focus more on tasks because they need not spend time trying to remember the differences in interaction. By providing a sense of stability, consistency makes the interface familiar and predictable.

Consistency is important through all aspects of the interface, including names of commands, visual presentation of information, and operational behavior. To design consistency into software, you must consider several aspects:

- Consistency within a product. Present common functions using a consistent set of commands and interfaces. For example, avoid implementing a Copy command that immediately carries out an operation in one situation but in another presents a dialog box that requires a user to type in a destination. As a corollary to this example, use the same command to carry out functions that seem similar to the user.
- Consistency within the operating environment. By maintaining a high level of consistency between the interaction and interface conventions provided by Windows, your software benefits from users' ability to apply interaction skills they have already learned.
- Consistency with metaphors. If a particular behavior is more characteristic of a different object than its metaphor implies, the user may have difficulty learning to associate that behavior with an object. For example, an incinerator communicates a different model than a wastebasket for the recoverability of objects placed in it.



Although applying the principle of consistency is the primary goal of this guide, the following chapters focus on the elements common to all Windows-based software: Chapter 6, “Windows,” Chapter 7, “Menus, Controls, and Toolbars,” and Chapter 8, “Secondary Windows.” For information about closely integrating your software with the Windows environment, see Chapter 10, “Integrating with the System,” and Chapter 11, “Working with OLE Embedded and OLE Linked Objects.”

## Forgiveness

Users like to explore an interface and often learn by trial and error. An effective interface allows for interactive discovery. It provides only appropriate sets of choices and warns users about potential situations where they may damage the system or data, or better, makes actions reversible or recoverable.

Even within the best designed interface, users can make mistakes. These mistakes can be both physical (accidentally pointing to the wrong command or data) and mental (making a wrong decision about which command or data to select). An effective design avoids situations that are likely to result in errors. It also accommodates potential user errors and makes it easy for the user to recover.

For information about applying the principle of forgiveness, see Chapter 12, “User Assistance,” which provides information about supporting discoverability in the interface through the use of contextual, task-oriented, and reference forms of user assistance. For information about designing for the widest range of users, see Chapter 14, “Special Design Considerations.”



## Feedback

Always provide feedback for a user's actions. Visual, and sometimes audio, cues should be presented with every user interaction to confirm that the software is responding to the user's input and to communicate details that distinguish the nature of the action.

Effective feedback is timely, and is presented as close to the point of the user's interaction as possible. Even when the computer is processing a particular task, provide the user with information regarding the state of the process and how to cancel that process if that is an option. Nothing is more disconcerting than a "dead" screen that is unresponsive to input. A typical user will tolerate only a few seconds of an unresponsive interface.

It is equally important that the type of feedback you use be appropriate to the task. Pointer changes or a status bar message can communicate simple information; more complex feedback may require the display of a message box.

For information about applying the principle of visual and audio feedback, see Chapter 13, "Visual Design," and Chapter 14, "Special Design Considerations."

## Aesthetics

The visual design is an important part of a software's interface. Visual attributes provide valuable impressions and communicate important cues to the interaction behavior of particular objects. At the same time, it is important to remember that every visual element that appears on the screen potentially competes for the user's attention. Provide a pleasant environment that clearly contributes to the user's understanding of the information presented. A graphics or visual designer may be invaluable with this aspect of the design.

For information and guidelines related to the aesthetics of your interface, see Chapter 13, "Visual Design." This chapter covers everything from individual element design to font use and window layout.

## Simplicity

An interface should be simple (not simplistic), easy to learn, and easy to use. It must also provide access to all functionality provided by an application. Maximizing functionality and maintaining simplicity work against each other in the interface. An effective design balances these objectives.

One way to support simplicity is to reduce the presentation of information to the minimum required to communicate adequately. For example, avoid wordy descriptions for command names or messages. Irrelevant or verbose phrases clutter your design, making it difficult for users to easily extract essential information. Another way to design a simple but useful interface is to use natural mappings and semantics. The arrangement and presentation of elements affects their meaning and association.

You can also help users manage complexity by using progressive disclosure. *Progressive disclosure* involves careful organization of information so that it is shown only at the appropriate time. By “hiding” information presented to the user, you reduce the amount of information to process. For example, clicking a menu displays its choices; the use of dialog boxes can reduce the number of menu options.

Progressive disclosure does not imply using unconventional techniques for revealing information, such as requiring a modifier key as the only way to access basic functions or forcing the user down a longer sequence of hierarchical interaction. This can make an interface more complex and cumbersome.

For information about applying the principle of simplicity, see Chapter 7, “Menus, Controls, and Toolbars.” This chapter discusses progressive disclosure in detail and describes how and when to use the standard (system-supplied) elements in your interface.

# Design Methodology

Effective interface design is more than just following a set of rules. It requires a user-centered attitude and design methodology. It also involves early planning of the interface and continued work through the software development process.

## A Balanced Design Team

An important consideration in the design of a product is the composition of the team that designs and builds it. Always try to balance disciplines and skills, including development, visual design, writing, human factors, and usability assessment. Rarely are these characteristics found in a single individual, so create a team of individuals who specialize in these areas and who can contribute uniquely to the final design.

Ensure that the design team can effectively work and communicate together. Locating them in close proximity or providing them with a common area to work out design details often fosters better communication and interaction.

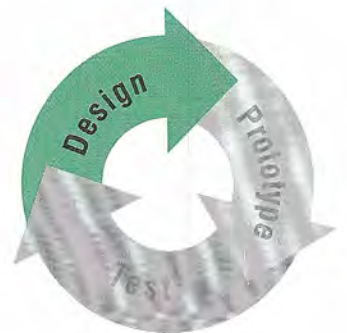
## The Design Cycle

An effective user-centered design process involves a number of important phases: designing, prototyping, testing, and iterating. The following sections describe these phases.

### Design

The initial work on a software's design can be the most critical because, during this phase, you decide the general shape of your product. If the foundation work is flawed, it is difficult to correct afterwards.

This part of the process involves not only defining the objectives and features for your product, but understanding who your users are and their tasks, intentions, and goals. This includes understanding factors such as their background — age, gender, expertise, experience level, physical limitations, and special needs; their work environment —





equipment, social and cultural influences, and physical surroundings; and their current task organization — the steps required, the dependencies, redundant activities, and the output objective. An order-entry system may have very different users and requirements than an information kiosk.

At this point, begin defining your conceptual framework to represent your product with the knowledge and experience of your target audience. Ideally, you want to create a design model that fits the user's conceptual view of the tasks to be performed. Consider the basic organization and different types of metaphors that can be employed. Observing users at their current tasks can provide ideas on effective metaphors to use.

Document your design. Committing your planned design to a written format not only provides a valuable reference point and form of communication, but often helps make the design more concrete and reveals issues and gaps.

## Prototype

After you have defined a design model, prototype some of the basic aspects of the design. This can be done with “pencil and paper” models — where you create illustrations of your interface to which other elements can be attached; storyboards — comic book-like sequences of sketches that illustrate specific processes; animation — movie-like simulations; or operational software using a prototyping tool or normal development tools.

A prototype is a valuable asset in many ways. First, it provides an effective tool for communicating the design. Second, it can help you define task flow and better visualize the design. Finally, it provides a low-cost vehicle for getting user input on a design. This is particularly useful early in the design process.

The type of prototype you build depends on your goal. Functionality, task flow, interface, operation, and documentation are just some of the different aspects of a product that need to be assessed. For example, pen and paper models or storyboards may work when defining task organization or conceptual ideas. Operational prototypes are usually best for the mechanics of user interaction.



Consider whether to focus your prototype on breadth or depth. The broader the prototype, the more features you should try to include to gain an understanding about how users react to concepts and organization. When your objective is focused more on detailed usage of a particular feature or area of the design, use depth-oriented prototypes that include more detail for a given feature or task.

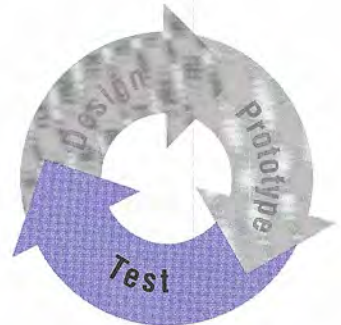
## Test

User-centered design involves the user in the design process. Usability testing a design, or a particular aspect of a design, provides valuable information and is a key part of a product's success. Usability testing is different than quality assurance testing in that, rather than find programming defects, you assess how well the interface fits user needs and expectations. Of course, defects can sometimes affect how well the interface will fit.

There can be different reasons for testing. You can use testing to look for potential problems in a proposed design. You can also focus on comparative studies of two or more designs to determine which is better, given a specific task or set of tasks.

Usability testing provides you not only with task efficiency and success-or-failure data, it also can provide you with information about the user's perceptions, satisfaction, questions, and problems, which may be just as significant as the ability to complete a particular task.

When testing, it is important to use participants who fit the profile of your target audience. Using fellow workers from down the hall might be a quick way to find participants, but software developers rarely have the same experience as their customers. The section, "Usability Assessment in the Design Process," provides details about conducting a usability test.





## Iterate

Because testing often uncovers design weaknesses, or at least provides additional information you will want to use, repeat the entire process, taking what you have learned and reworking your design or moving onto reprototyping and retesting. Continue this refining cycle through the development process until you are satisfied with the results.

During this iterative process, you can begin substituting the actual application for prototypes as the application code becomes available. However, avoid delaying your design cycle waiting for the application code to be complete enough; you can lose valuable time and input that you could have captured with a prototype. Moreover, by the time most applications are complete enough for testing, it is difficult to consider significant changes, because it becomes easier to ignore usability defects because of the time resources invested. In addition, changes at this point may affect the application's delivery schedule.



## Usability Assessment in the Design Process

As described in the previous section, usability testing is a key part of the design process, but testing design prototypes is only one part of the picture. Usability assessment should begin in the early stages of product development, where you can use it to gather data about how users do their work. You then roll your findings back into the design process. As the design progresses, usability assessment continues to provide valuable input for analyzing initial design concepts and, in the later stages of product development, can be used to test specific product tasks. Apply usability assessment early and often.

Consider the user's entire experience as part of a product's usability. The usability assessment should include all of a product's components. A software interface is more than just what shows up on the screen or in the documentation.



## Usability Testing Techniques

Usability testing involves a wide range of techniques and investment of resources, including trained specialists working in sound-proofed labs with one-way mirrors and sophisticated recording equipment. However, even the simplest investment of an office or conference room, tape recorder, stopwatch, and notepad can produce benefits. Similarly, all tests need not involve great numbers of subjects. More typically, quick, iterative tests with a small, well-targeted sample, 6-10 participants, can identify 80 to 90 percent of most design problems.

Like the design process itself, usability testing begins with defining the target audience and test goals. When designing a test, focus on tasks — not features. Even if your goal is testing specific features, remember that your customers will use them within the context of particular tasks. It is also a good idea to run a pilot test to work out the bugs of the tasks to be tested and make certain the task scenarios, prototype, and equipment work smoothly.

When conducting the usability test, provide an environment comparable to the target setting; usually a quiet location, free from distractions, is best. Make participants feel comfortable. Unless you have participated yourself, you may be surprised by the pressure many test participants feel. You can alleviate some pressure by explaining the testing process and equipment to the participants, and stating your objective in testing the software and not them; if they become confused or frustrated, it is not a reflection upon them.

Allow the user reasonable time to try and work through any difficult situations. Although it is generally best to not interrupt participants during a test, they may get stuck or end up in situations that require intervention. This need not necessarily disqualify the test data, as long as the test coordinator carefully guides or hints around a problem. Give general hints before moving to specific advice. For more difficult situations, you may need to stop the test and make adjustments. Keep in mind that less intervention usually yields better results. Always record the techniques and search patterns that users employ when attempting to work through a difficulty, and the number and type of hints you have to provide.

Ask subjects to think aloud as they work, so you can hear what assumptions and inferences they are making. As the participants work, record the time they take to perform a task as well as any problems they encounter. You may also want to follow up the session with a questionnaire that asks the participants to evaluate the product or tasks they performed.

Record the test results using a portable tape recorder, or better, a video camera. Since even the best observer can miss details, reviewing the data later will prove invaluable. Recorded data also allows more direct comparisons between multiple participants. It is usually risky to base conclusions on observing a single subject. Recorded data also allows all the design team to review and evaluate the results.

Whenever possible, involve *all* members of the design team in observing the test and reviewing the results. This ensures a common reference point and better design solutions as team members apply their own insights to what they observe. If direct observation is not possible, make the recorded results available to the entire team.

## Other Assessment Techniques

There are many techniques you can use to gather usability information. In addition to those already mentioned, “focus groups” are helpful for generating initial ideas or trying out ideas. A focus group requires a moderator who directs the discussion about aspects of a task or design, but allows participants to freely express their opinions. You can also conduct demonstrations, or “walkthroughs,” in which you take the user through a set of sample scenarios and ask about their impressions along the way. In a so-called “Wizard of Oz” technique, a testing specialist simulates the interaction of an interface. Although these latter techniques can be valuable, they often require a trained, experienced test coordinator.



# Understanding Users

The design and usability techniques described in the previous sections have been used in the development of Windows and in many of the guidelines included in this book. That process has yielded the following general characteristics about users. Consider these characteristics in the design of your software:

- Beginning Windows users often have difficulty with the mouse. For example, dragging and double-clicking are skills that may take time for beginning mouse users to master. Dragging can be difficult because it requires continued pressure on the mouse button and involves properly targeting the correct destination. Double-clicking is not the same as two separate clicks, so many beginning users have difficulty handling the timing necessary to distinguish these two actions, or they overgeneralize the behavior to assume that everything needs double-clicking. Design your interface so that double-clicking and dragging are not the only ways to perform basic tasks; allow the user to conduct those tasks using single click operations.
- Beginning users often have difficulty with window management. They do not always realize that overlapping windows represent a three-dimensional space. As a result, when a window is hidden by another, a user may assume it no longer exists.
- Beginning users often have difficulty with file management. The organization of files and folders nested more than two levels is more difficult to understand because it is not as obvious in the real world.
- Intermediate users may understand file hierarchies, but have difficulty with other aspects of file management — such as moving and copying files. This may be because most of their experience working with files is often from within an application.
- Advanced, or “power,” users want efficiency. The challenge in designing for advanced users is providing for efficiency without introducing complexity for less-experienced users. (Shortcut methods are often useful for supporting these users.) In addition, advanced users may be dependent upon particular interfaces, making it difficult for them to adapt to significant rearrangement or changes in an interface.



- To develop for the widest audience, consider international users and users with disabilities. Including these users as part of your planning and design cycle is the best way to ensure that you can accommodate them.

## Design Tradeoffs

A number of additional factors may affect the design of a product. For example, marketing considerations may require you to deliver a product with a minimal design process, or comparative evaluations may force you to consider additional features. Remember that shortcuts and additional features can affect the product. There is no simple equation to determine when a design tradeoff is appropriate. So in evaluating the impact, consider the following:

- Every additional feature potentially affects performance, complexity, stability, maintenance, and the support costs of an application.
- It is harder to fix a design problem after the release of a product because users may adapt, or even become dependent on, a peculiarity in the design.
- Simplicity is not the same as being simplistic. Making something simple to use often requires a good deal of work and code.
- Features implemented by a small extension in the application code do not necessarily have a proportional effect in a user interface. For example, if the primary task is selecting a single object, extending it to support selection of multiple objects could make the frequent, simple task more difficult to carry out.

# Basic Concepts



Microsoft Windows supports the evolution and design of software from a basic graphical user interface to a data-centered interface that is better focused on users and their tasks. This chapter outlines the fundamental concepts of data-centered design. It covers some of the basic definitions used throughout this guide and provides the fundamental model for how to define your interface to fit well within the Windows environment.

## Data-Centered Design

*Data-centered design* means that the design of the interface supports a model where a user can browse for data and edit it directly instead of having to first locate an appropriate editor or application. As a user interacts with data, the corresponding commands and tools to manipulate the data or the view of the data become available to the user automatically. This frees a user to focus on the information and tasks rather than on applications and how applications interact.

In this data-centered context, a *document* is a common unit of data used in tasks and exchanged between users. The use of the term is not limited to the output of a word-processing or spreadsheet application, but it emphasizes that the focus of design is on data, rather than the underlying application.

# Objects as Metaphor

A well-designed user interface provides an understandable, consistent framework in which users can work, without being confounded by the details of the underlying technology. To help accomplish this, the design model of the Windows user interface uses the metaphor of objects. This is a natural way we interpret and interact with the world around us. In the interface, *objects* not only describe files or icons, but any unit of information, including cells, paragraphs, characters, and circles, and the documents in which they reside.

## Object Characteristics

Objects, whether real-world or computer representations, have certain characteristics that help us understand what they are and how they behave. The following concepts describe the aspects and characteristics of computer representations:

- **Properties** — Objects have certain characteristics or attributes, called *properties*, that define their appearance or state — for example, color, size, and modification date. Properties are not limited to the external or visible traits of an object. They may reflect the internal or operational state of an object, such as an option in a spelling check utility that automatically suggests alternative spellings.
- **Operations** — Things that can be done with or to an object are considered its *operations*. Moving or copying an object are examples of operations. You can expose operations in the interface through a variety of mechanisms, including commands and direct manipulation.
- **Relationships** — Objects always exist within the context of other objects. The context, or *relationships*, that an object may have often affects the way the object appears or behaves. Common kinds of relationships include collections, constraints, and composites.



## Relationships

The simplest relationship is a *collection*, in which objects in a set share a common aspect. The results of a query or a multiple selection of objects are examples of a collection. The significance of a collection is that it enables operations to be applied to a set of objects.

A *constraint* is a stronger relationship between a set of objects in that changing an object in the set affects some other object in the set. The way a text box streams text, the way a drawing application layers its objects, and even the way a word-processing application organizes a document into pages are all examples of constraints.

When a relationship between objects becomes so significant that the aggregation can be identified as an object itself with its own set of properties and operations, the relationship is called a *composite*. A range of cells, a paragraph, and a grouped set of drawing objects are examples of composites.

Another common kind of relationship found in the interface is containment. A *container* is an object that is the place where other objects exist, such as text in a document or documents in a folder. A container often influences the behavior of its content. It may add or suppress certain properties or operations of an object placed in it. In addition, a container controls access to its content as well as what kind of object it will accept as its content. This may affect the results when transferring objects from one container to another.

All these aspects contribute to an object's *type*, a descriptive way of distinguishing or classifying objects. Objects of a common type have similar traits and behaviors.

## Composition

As in the natural world, the metaphor of objects implies a constructed environment. Objects are compositions of other objects. You can define most tasks supported by applications as a specialized combination or set of relationships between objects. A text document is a composition of text, paragraphs, footnotes, or other items. A table is a combination of cells; a chart is a particular organization of graphics. When you define user interaction with objects to be as consistent as possible at any level, you can produce complex constructions while maintaining a small, basic set of conventions. These

conventions can apply throughout the interface, increasing ease of use. In addition, using composition to model tasks encourages modular, component-oriented design. This allows objects to be adapted or recombined for other uses.

## Persistence

In the natural world, objects persist in their existing state unless changed or destroyed. When you use a pen to write a note, you need not invoke a command to ensure that the ink is preserved on the paper. The act of writing implicitly preserves the information. This is the long term direction for objects in the interface as well. Although it is still appropriate to design software that requires explicit user actions to preserve data, consider whether data can be preserved automatically. In addition, view state information, such as cursor position, scroll position, and window size and location, should be preserved so it can be restored when an object's view is reopened.

## Putting Theory into Practice

Using objects in an interface design does not guarantee usability. But applying object-based concepts does offer greater potential for a well-designed interface. As with any good user interface design, a good user-centered design process ensures the success and quality of the interface.

The first step to object-based design should begin as any good design with a thorough understanding of what users' objectives and tasks are. When doing the task analysis, identify the basic components or objects used in those tasks and the behavior and the characteristics that differentiate each kind of object, including the relationships of the objects to each other and to the user. Also identify the actions that are performed, the objects to which they apply, and the state information or attributes that each object in the task must preserve, display, and allow to be edited.

Once the analysis is complete, you can start identifying the user interfaces for the objects. Define how the objects you identified are to be presented, either as icons or data elements in a form. Use icons primarily for representing composite or container objects that need to be opened into their own windows. Attribute or state information

should typically be presented as properties of the associated object, most often using property sheets. Map behaviors and operations to specific kinds of interaction, such as menu commands, direct manipulation, or both. Make these accessible when the object is selected by the user. The information in this guide will help you define how to apply the interfaces provided by the system.

Redesigning an existing Windows 3.1-based application to a more data-centered interface need not require an immediate, complete overhaul. You can begin the evolution by adding contextual interfaces such as pop-up menus, property sheets, and OLE drag and drop and by following the recommendations for designing your window title bars and icons.





# The Windows Environment



This chapter provides a brief overview of some of the basic elements included in the Microsoft Windows operating system that allow the user to control the environment (sometimes collectively referred to as the *shell*). These elements provide not only the backdrop for a user's environment, but can be landmarks for the user's interaction with your application as well.

## The Desktop

The *desktop* represents a user's primary work area; it fills the screen and forms the visual background for all operations (as shown in Figure 3.1). However, the desktop is more than just a background. It can also be used as a convenient location to place objects that are stored in the file system. In addition, for a computer connected to a network, the desktop also serves as a private work area through which a user can still browse and access objects remotely located on the network.

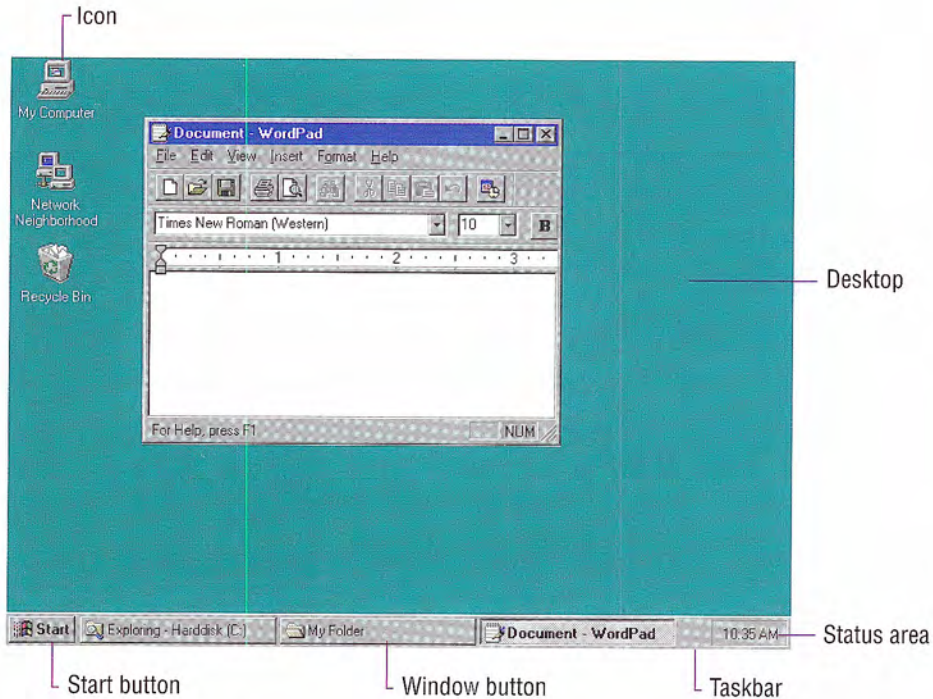



Figure 3.1 The desktop

## The Taskbar

The taskbar is a special component of the desktop that can be used to switch between open windows and to access global commands and other frequently used objects. As a result, it provides a home base — an operational anchor for the interface.

Like most toolbars, the taskbar can be configured. For example, a user can move the taskbar from its default location and relocate it along another edge of the screen (as shown in Figure 3.2). The user can also configure display options of the taskbar. The taskbar can

 For more information about integrating your application with the taskbar, see Chapter 10, “Integrating with the System.”



provide the user access to your application. It can also be used to provide status information even when your application is not active. Because the taskbar is an interface shared across applications, be sure to follow the conventions and guidelines covered in this guide.

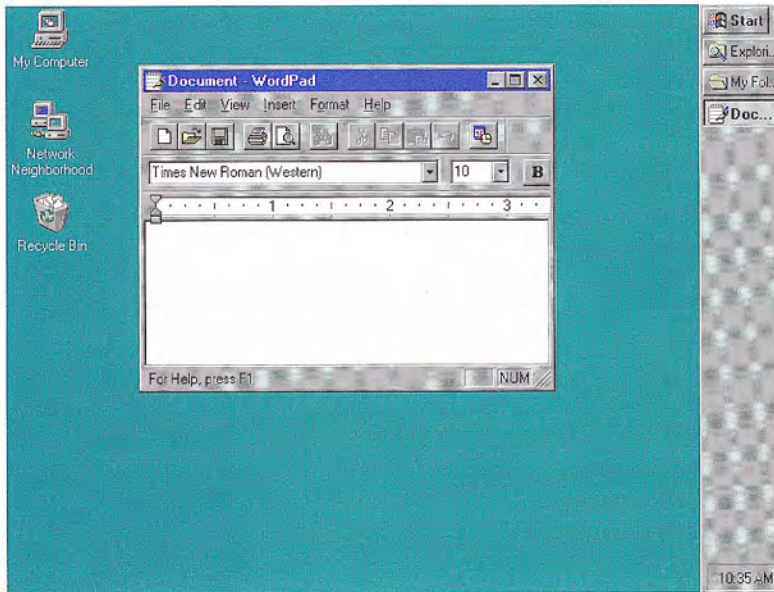


Fig 3.2 Showing the taskbar in another location

## The Start Button

The Start button at the left side of the taskbar displays a special menu that includes commands for opening or finding files. The Program menu entry automatically includes the Program Manager entries when the system is installed over Windows 3.1. When installing your Windows-based application, you also can include an entry for your application by placing a shortcut icon in the system's Programs folder.



## Window Buttons


Whenever the user opens a primary window, a button is placed on the taskbar for that window. This button provides the user access to the commands of that window and a convenient interface for switching to that window. The taskbar automatically adjusts the size of the buttons to accommodate as many buttons as possible. When the size of the button requires that the window's title be abbreviated, the taskbar also automatically supplies a small pop-up window (as shown in Figure 3.3) that displays the full title for the window.



**Figure 3.3 Pop-up window with full title**

When a window is minimized, the window's button remains on the taskbar, but is removed when the window is closed.

Taskbar buttons can also be used as drag and drop destinations. When the user drags over a taskbar button, the system activates the associated window, allowing the user to drop within that window.

 For more information about drag and drop, see Chapter 5, "General Interaction Techniques."

## The Status Area

On the opposite side of the taskbar from the Start menu is a special status area. Your application can place special status or notification indicators here, even when it is not active.

## Icons

Icons may appear on the desktop and in windows. *Icons* are pictorial representations of objects. This goes beyond the use of icons in Windows 3.1, which only represented minimized windows. Your software should provide and register icons for its application file and any of its associated document or data files.

Windows includes a number of icons that represent basic objects, such as the following.










 For more information about the use of icons, see Chapter 10, "Integrating with the System." For information about icon design, see Chapter 13, "Visual Design."


Table 3.1 Icons

Icon	Type	Function
 My Computer	System Folder	Provides access to a user's private storage.
 Network Neighborhood	System Folder	Provides access to the network.
 Folder	Folder	Provides organization of files and folders.
 Shortcut to My Favorite Folder	Shortcut	Provides access to other objects. A shortcut icon uses the icon of the type of file it is linked to, overlaid with the link symbol.
 All Files	Saved Search	Locates files or folders.
 Windows Explorer	Application	Allows browsing of the content of a user's computer or the network.
 Recycle Bin	System Folder	Stores deleted icons.
 Control Panel	System Folder	Provides access to properties of installed devices and resources (for example, fonts, displays, and keyboards).



# Windows

You can open icons into windows. Windows provides a means of viewing and editing information, and viewing the content and properties of objects. You can also use windows to display parameters to complete commands, palettes of controls, or messages informing a user of a particular situation. Figure 3.4 demonstrates some of the different uses for windows.

 For more information about windows, see Chapter 6, “Windows,” and Chapter 8, “Secondary Windows.”

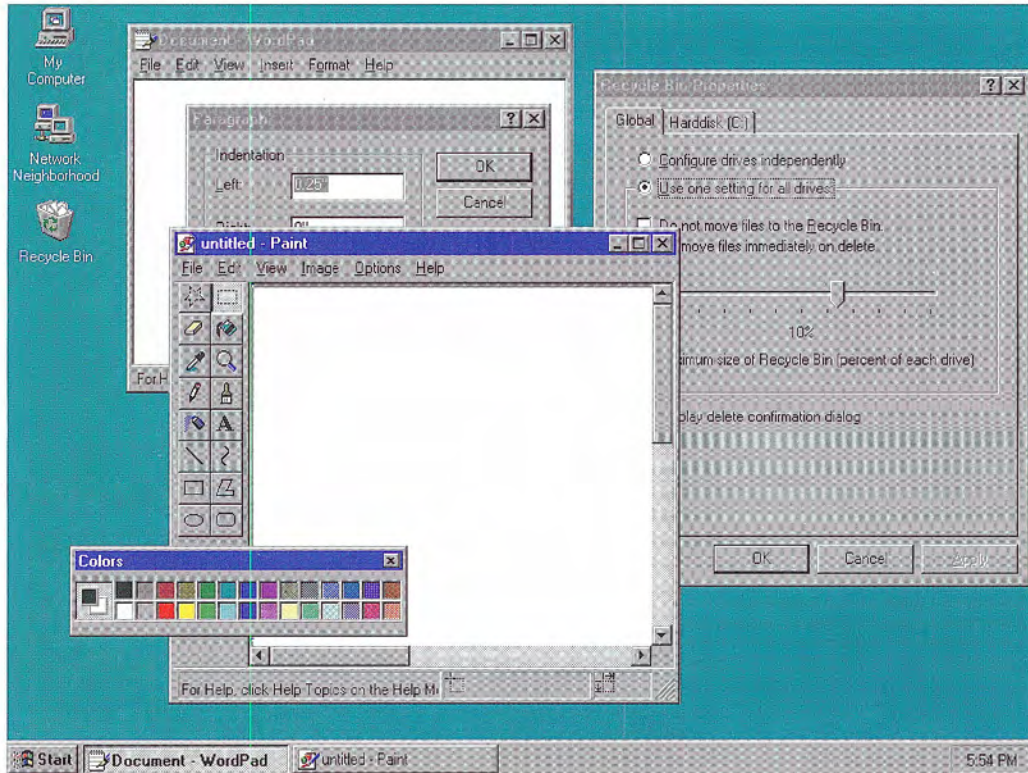


Figure 3.4 Different uses of windows

# Input Basics




A user can interact with objects in the interface using different types of input devices. The most common input devices are the mouse, the keyboard, and the pen. This chapter covers the basic behavior for these devices; it does not exclude other forms of input.

## Mouse Input



The mouse is a primary input device for interacting with objects in the Microsoft Windows interface. Other types of pointing devices that emulate a mouse, such as trackballs, fall under the general use of the term “mouse.”
















 For more information about interactive techniques such as navigation, selection, viewing, editing, transfer, and creating new objects, see Chapter 5, “General Interaction Techniques.”


## Mouse Pointers

The mouse is operationally linked with a graphic on the screen called the *pointer* (also referred to as the *cursor*). By positioning the pointer and clicking the buttons on the mouse, a user can select objects and their operations.

As a user moves the pointer across the screen, its appearance can change to provide feedback about a particular location, operation, or state. Table 4.1 lists some common pointer shapes and their uses.

Table 4.1 Common Pointers

Shape	Screen location	Available or current action
	Over most objects	Pointing, selecting, or moving.
	Over text	Selecting text.
	Over any object or location	Processing an operation.
	Over any screen location	Processing in the background (application loading), but the pointer is still interactive.
	Over most objects	Context-sensitive Help mode.
	Inside a window	Zooming a view.
	Over a sizable edge	Resizing an edge vertically.
	Over a sizable edge	Resizing an edge horizontally.
	Over a sizable edge	Resizing an edge diagonally.
	Over a sizable edge	Resizing an edge diagonally.
	Along column gridlines	Resizing a column.
	Along row gridlines	Resizing a row.
	Over split box in vertical scroll bar	Splitting a window (or adjusting a split) horizontally.
	Over split box in horizontal scroll bar	Splitting a window (or adjusting a split) vertically.
	Over any object	Not available as a drop target.

 The system does not provide all of these pointers. For more information about designing your own pointers, see Chapter 13, “Visual Design.”



Each pointer has a particular point — called a *hot spot* — that defines the exact screen location of the mouse. The hot spot determines what object is affected by mouse actions. Screen objects can additionally define a hot zone; the *hot zone* defines the area the hot spot must be within to be considered over the object. Typically, the hot zone coincides with the borders of an object, but it may be larger, or smaller, to make user interaction easier.

## Mouse Actions

Basic mouse actions in the interface use mouse button 1 or button 2. By default, button 1 is the leftmost mouse button and button 2 is the rightmost button. The system allows the user to swap the mapping of the buttons. Button 2 actions typically duplicate functions already accessible with button 1, but provide those functions more efficiently.



For a mouse with three buttons, button 2 is the *rightmost* button, not the center button.

The following are the common behaviors performed with the mouse.

Action	Description
Pointing	Positioning the pointer so it “points to” a particular object on the screen without using the mouse button. Pointing is usually part of preparing for some other interaction. Pointing is often an opportunity to provide visual cues or other feedback to a user.
Clicking	Positioning the pointer over an object and then pressing and releasing the mouse button. Generally, the mouse is not moved during the click, and the mouse button is quickly released after it is pressed. Clicking identifies (selects) or activates objects.
Double-clicking	Positioning the pointer over an object and pressing and releasing the mouse button twice in rapid succession. Double-clicking an object typically invokes its default operation.
Pressing	Positioning the pointer over an object and then holding down the mouse button. Pressing is often the beginning of a click or drag operation.
Dragging	Positioning the pointer over an object, pressing down the mouse button while holding the mouse button down, and moving the mouse. Use dragging for actions such as selection and direct manipulation of an object.

For most mouse interactions, pressing the mouse button only identifies an operation. User feedback is usually provided at this point. Releasing the mouse button activates (carries out) the operation. An auto-repeat function — for example, pressing a scroll arrow to continuously scroll — is an exception.

This guide does not cover other mouse behaviors such as *chording* (pressing multiple mouse buttons simultaneously) and multiple-clicking (triple- or quadruple-clicking). Because these behaviors require more user skill, they are not generally recommended for basic operations. However, you can consider them for special shortcut operations.

Because not every mouse has a third button, there is no basic action defined for a third (middle) mouse button. It is best to limit the assignment of operations to this button to those environments where the availability of a third mouse button can be assumed, and for providing redundant or shortcut access to operations supported elsewhere in the interface. When assigning actions to the button, you need to define the behaviors for the actions already described (pointing, clicking, dragging, and double-clicking) for this button.

## Keyboard Input



The keyboard is a primary means of entering or editing text information. However, the Windows interface also supports the use of keyboard input to navigate, toggle modes, modify input, and, as a shortcut, to invoke certain operations.



For more information about using the keyboard for navigation, selection, and editing, see Chapter 5, “General Interaction Techniques.”

Following are the common interactive behaviors performed with the keyboard.

Action	Description
Pressing	Pressing and releasing a key. Unlike mouse interaction, keyboard interaction occurs upon the down transition of the key. Pressing typically describes the keyboard interaction for invoking particular commands or for navigation.
Holding	Pressing and holding down a key. Holding typically describes interaction with keys such as ALT, SHIFT, and CTRL that modify the standard behavior of other input — for example, another key press or mouse action.
Typing	Typing input of text information from the keyboard.

## Text Keys

Text keys include the following:

- Alphanumeric keys (a–z, A–Z, 0–9)
- Punctuation and symbol keys
- TAB and ENTER keys
- The SPACEBAR

In text-entry contexts, pressing a text key enters the corresponding character and typically displays that character on the screen. Except in special views, the characters produced by the TAB and ENTER keys are not usually visible. In some contexts, text keys can also be used for navigation or for invoking specific operations.



Most keyboards include two keys labeled ENTER: one on the main keyboard and one on the numeric keypad. Because these keys have the same label (and on some keyboards the latter may not be available), assign both keys the same functionality.

## Access Keys

An access key is an alphanumeric key — sometimes referred to as a *mnemonic* — that when used in combination with the ALT key navigates to and activates a control. The access key matches one of the characters in the text label of the control. For example, pressing ALT+O activates a control whose label is “Open” and whose assigned access key is “O”. Typically, access keys are not case sensitive. The effect of activating a control depends on the type of control.



Assign access key characters to controls using the following guidelines (in order of choice):

1. The first letter of the label for the control, unless another letter provides a better mnemonic association.
2. A distinctive consonant in the label.
3. A vowel in the label.

Avoid assigning a character where the visual indication of the access key cannot be distinguished from the character. Also, avoid using a character normally assigned to a common function. For example, when you include an Apply button, reserve the “A” — or its localized equivalent — as the access key for that button. In addition, do not assign access keys to the OK and Cancel commands when they map to the ENTER and ESC keys, respectively.

Nonunique access key assignments within the same scope access the first control. Depending on the control, if the user presses the access key a second time, it may or may not access another control with the same assignment. Therefore, define an access key to be unique within the scope of its interaction — that is, the area in which the control exists and to which keyboard input is currently being directed.

Controls without explicit labels can use static text controls to create labels with assigned access keys. Software that supports a nonroman writing system (such as Kanji), but that runs on a standard keyboard, can prefix each control label with an alphabetic (roman) character as its access key.



For more information about static text controls, see Chapter 7, “Menus, Controls, and Toolbars.”

## Mode Keys

Mode keys change the actions of other keys (or other input devices). There are two kinds of mode keys: toggle keys and modifier keys.

A toggle key turns a particular mode on or off each time it is pressed. For example, pressing the CAPS LOCK key toggles uppercase alphabetic keys; pressing the NUM LOCK key toggles between numeric and directional input using the keypad keys.

Like toggle keys, modifier keys change the actions of normal input. Unlike toggle keys, however, modifier keys establish modes that remain in effect only while the modifier key is held down. Modifier keys include the SHIFT, CTRL, and ALT keys. Such a “spring-loaded” mode is often preferable to a “locked” mode because it requires the user to continuously activate it, making it a conscious choice and allowing the user to easily cancel the mode by releasing the key.

Because it can be difficult for a user to remember multiple modifier assignments, avoid using multiple modifier keys as the primary means of access to basic operations. In some contexts, such as environments that are specific to pen input, the keyboard may not be available. Therefore, use modifier-based actions only for quick access to operations that are supported adequately elsewhere in the interface.

## Shortcut Keys

Shortcut keys (also referred to as accelerator keys) are keys or key combinations that, when pressed, provide quick access to frequently performed operations. CTRL+*letter* combinations and function keys (F1 through F12) are usually the best choices for shortcut keys. By definition, a shortcut key is a keyboard equivalent of functionality that is supported adequately elsewhere in the interface. Therefore, avoid using a shortcut key as the only way to access a particular operation.

When defining shortcut keys, observe the following guidelines:

- Assign single keys where possible because these keys are the easiest for the user to perform.
- Make modified-letter key combinations case insensitive.
- Use SHIFT+*key* combinations for actions that extend or complement the actions of the key or key combination used without the SHIFT key. For example, ALT+TAB switches windows in a top-to-bottom order. SHIFT+ALT+TAB switches windows in reverse order. However, avoid SHIFT+*text* keys, because the effect of the SHIFT key may differ for some international keyboards.
- Use CTRL+*key* combinations for actions that represent a larger scale effect. For example, in text editing contexts, HOME moves to the beginning of a line, and CTRL+HOME moves to the beginning



Function key and modified function key combinations may be easier for international users because they have no mnemonic relationship. However, there is a tradeoff because function keys are often more difficult to remember and to reach. For a list of the most common shortcut key assignments, see Appendix B, “Keyboard Interface Summary.”



of the text. Use CTRL+*key* combinations for access to commands where a letter key is used — for example, CTRL+B for bold. Remember that such assignments may be meaningful only for English-speaking users.

- Avoid ALT+*key* combinations because they may conflict with the standard keyboard access for menus and controls. The ALT+*key* combinations — ALT+TAB, ALT+ESC, and ALT+SPACEBAR — are reserved for system use. ALT+*number* combinations enter special characters.
- Avoid assigning shortcut keys defined in this guide to other operations in your software. That is, if CTRL+C is the shortcut for the Copy command and your application supports the standard copy operation, don't assign CTRL+C to another operation.
- Provide support for allowing the user to change the shortcut key assignments in your software, when possible.
- Use the ESC key to stop a function in process or to cancel a direct manipulation operation. It is also usually interpreted as the shortcut key for a Cancel button.

Some keyboards also support three new keys, the Application key and the two Windows keys. The primary use for the Application key is to display the pop-up menu for current selection (same as SHIFT+F10). You may also use it with modifier keys for application-specific functions. Pressing either of the Windows keys — left or right — displays the Start menu. These keys are also used by the system as modifiers for system-specific functions. Do not use these keys as modifiers for nonsystem-level functions.



**Windows key and  
Application key**



# Pen Input



Systems with a Windows pen driver installed support user input using tapping or writing on the surface of the screen or a tablet with a pen, and in some cases with a finger.

Depending on the placement of the pen, you can use it for both pointing and writing. For example, if you move the pen over menus or most controls, it acts as a pointing device. Because of the pointing capabilities of the pen, the user can perform most mouse-based operations. When over a text entry or drawing area, the pen becomes a writing or drawing tool; the pointer changes to a pen shape to provide feedback to the user. When the tip of the pen touches the input surface, the pen starts *inking* — that is, tracing lines on the screen. The user can then draw shapes, characters, and other patterns; these patterns remain on the screen exactly as drawn or can be recognized, interpreted, and redisplayed.

The pen can retain the functionality of a pointing device (such as a mouse) even in contexts where it would normally function as a writing or drawing tool. For example, you can use timing to differentiate operations; that is, if the user holds the pen tip in the same location for a predetermined period of time, a different action may be inferred. However, this method is often unreliable or inefficient for many operations, so it may be better to use toolbar buttons to switch to different modes of operation. Choosing a particular button allows the user to define whether to use the pen for entering information (writing or drawing) or as a pointing device.

You can also provide the user with access to other operations using an action handle. An *action handle* is a special graphic displayed for a selection. An action handle can be used to support direct manipulation operations or to provide access to pop-up menus.




The **GetSystemMetrics** function provides access to the `SM_PENWINDOWS` constant that indicates when a pen is installed. For more information about this function, see the documentation included in the Microsoft Win32 Software Development Kit (SDK).



For more information about action handles, see Chapter 5, “General Interaction Techniques.”


Following are the fundamental behaviors defined for a pen.

Action	Description
Pressing	Positioning and pressing the tip to the input surface. A pen press is equivalent to a mouse press and typically identifies a particular pen action.
Tapping	Pressing the pen tip on the input surface and lifting it without moving the pen. In general, tapping is equivalent to clicking mouse button 1. Therefore, this action typically selects an object, setting a text insertion point or activating a button
Double-tapping	Pressing and lifting the pen tip twice in rapid succession. Double-tapping is usually interpreted as the equivalent to double-clicking mouse button 1.
Dragging	Pressing the pen tip on the input surface and keeping it pressed while moving the pen. In inking contexts, you can use dragging for the input of pen strokes for writing, drawing, gestures, or for direct manipulation, depending on which is most appropriate for the context. In noninking contexts, it is the equivalent of a mouse drag.

 A user may move the pen more between taps when double-tapping than a user double-clicking with a mouse. As a result, you may want to slightly increase your hot zones for detecting a double-tap when of a pen device has been installed.

Some pens include buttons on the pen barrel that can be pressed. For pens that support barrel buttons, the following behaviors may be supported.

Action	Description
Barrel-tapping	Holding down the barrel button of the pen while tapping. Barrel-tapping is equivalent to clicking with mouse button 2.
Barrel-dragging	Holding down the barrel button of the pen while dragging the pen. Barrel-dragging is equivalent to dragging with mouse button 2.

 Because not all pens support barrel buttons, any behaviors that you support using a barrel button should also be supported by other techniques in the interface.

Pen input is delimited, by the lifting of the pen tip, an explicit termination tap (such as tapping the pen on another window or as the completion of a gesture), or a time-out without further input. You can also explicitly define an application-specific recognition time-out.



*Proximity* is the ability to detect the position of the pen without it touching the input surface. While Windows provides support for pen proximity, avoid depending on proximity as the exclusive means of access to basic functions, because not all pen hardware supports this feature. Even pen hardware that does support proximity may allow other non-pen input, such as touch input, where proximity cannot be supported.

## Pen Pointers

For pen tablets, as with a mouse, pointers play an important part in visually indicating the user's location of interaction on the screen. When the input surface is actually a screen display, pointers may seem superfluous; however, they still have an important role to play. Pointers help the pen user select small targets faster. Moreover, changes from one pointer to another provide useful feedback about the actions supported by the object under the pen. For example, when the pen moves over a resizable border, the pointer can change from a pen (indicating that writing is possible) to a resizing pointer (indicating that the border can be dragged to resize the object). Whenever possible, include this type of feedback in pen-enabled applications to help users understand the kinds of supported actions.

Following are two common pointers used with the pen.

**Table 4.2 Pen Pointers**

Shape	Common usage
	Pointing, selecting, moving, and resizing
	Writing and drawing



When the screen is the input surface — because a pointer may be partially obscured by the pen or by the user’s hand — you may need to consider including additional forms of feedback, such as toolbar button states or status bar information, to indicate the pen’s input state.

## Pen Gestures

When using the pen for writing, certain ink patterns are interpreted as *gestures*. Using one of these specially drawn symbols invokes a particular operation, such as deleting text, or produces a nonprinting text character, such as a carriage return or a tab. For example, a circled X gesture is equivalent to the Cut command. After the system interprets a gesture, the gesture’s ink is removed from the display.

All gestures include a circular stroke to distinguish them from ordinary characters. Most gestures also operate positionally; in other words, they act upon the objects on which they are drawn. Determining the position of the specific gesture depends on either the area surrounded by the gesture or a single point — the hot spot of the gesture.

Pen gestures usually cannot be combined with ink (writing or drawing actions) within the same recognition sequence. For example, the user cannot draw a few characters, immediately followed by a gesture, followed by more characters.

The rapidity of gestural commands is one of the key advantages of the pen. Do not rely on gestures as the only or primary way to perform commands, however, because gestures require memorization by users. Regard gestures as a quick access, shortcut method for operations adequately supported elsewhere in the interface, such as in menus or buttons. If the pen extensions are installed, you can optionally place a bitmap of the gesture next to the corresponding command (in place of the keyboard shortcut text) to help the user learn particular gestures.

In addition, avoid using gestures when they interfere with common functionality or make operations with parallel input devices, such as the mouse or keyboard, more cumbersome. For example, although writing a character gesture in a list box could be used as a way to



For more information about common gestures and their interpretation, see Chapter 5, “General Interaction Techniques.”

scroll automatically within the list, it would interfere with the basic and more frequent user action of selecting an item in the list. A better technique is to provide a text input field where the user can write and, based on the letters entered, scroll the list.

## Pen Recognition

*Recognition* is the interpretation of pen strokes into some standardized meaning. Consider recognition as a means to an end, not an end in itself. Do not use recognition if it is unnecessary or if it is not the best interface. For example, it may be more effective to provide a control that allows a user to select a date, rather than requiring the user to write it in just so your software can recognize it.

Accurate recognition is difficult to achieve, but you can greatly improve your recognition interface by providing a fast, easy means to correct errors. For example, if you allow users to overwrite characters or choose alternatives, they will be less frustrated and find recognition more useful. You can also improve recognition by using context and constraints. For example, a checkbook application can constrain certain fields to contain only numbers.

## Ink Input

In some cases — for example, signatures — recognition of pen input may be unnecessary; the ink is a sufficient representation of information. Ink is a standard data type supported by the Clipboard. Consider supporting ink entries as input wherever your software accepts normal text input, unless the representation of that input needs to be interpreted for other operations, such as searching or sorting.

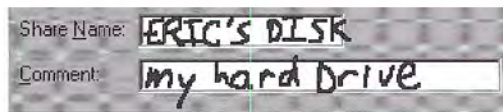
## Targeting

Targeting, or determining where to direct pen input, is an important design factor for pen-enabled software. For example, if the user gestures over a set of objects, which objects should be affected? If the user writes text that spans several writing areas, which text should be placed in which area? In general, you use the context of the input to determine where to apply pen input. More specifically, use the following guidelines for targeting gestures on objects:

- If the user draws the gesture on any part of a selection, apply the gesture to the selection.
- If the user draws the gesture on an object that is not selected, select that object, and apply the gesture to that object.
- If the user does not draw the gesture on any object or selection, but there is a selection, apply the gesture to that selection.

If none of these guidelines applies, ignore the gesture.

For handwriting, you can also use context to determine where to direct the input. Figure 4.1 demonstrates how the proximity of the text to the text boxes determines the destination of the written text.



**Figure 4.1 Targeting handwritten input**

The system's pen services provide basic support for targeting, but your application can also provide additional support. For example, your application can define a larger inking rectangle than the control usually provides. In addition, because your application often knows the type of input to expect, it can use this information to better interpret where to target the input.



# General Interaction Techniques



This chapter covers basic interaction techniques, such as navigation, selection, viewing, editing, and creation. Many of these techniques are based on an object-action paradigm in which a user identifies an object and an action to apply to that object. By maintaining these techniques consistently, you enable users to transfer their skills to new tasks.

Where applicable, support the basic interaction techniques for the mouse, keyboard, and pen. When adding or extending these basic techniques, consider how the feature or function can be supported across input devices. Techniques for a particular device need not be identical for all devices. Instead, tailor techniques to optimize the strengths of a particular device. In addition, make it easy for the user to switch between devices so that an interaction started with one device can be completed with another.

## Navigation

One of the most common ways of identifying or accessing an object is by navigating to it. The following sections include information about mouse, pen, and keyboard techniques.

## Mouse and Pen Navigation

Navigation with the mouse is simple; when a user moves the mouse left or right, the pointer moves in the corresponding direction on the screen. As the mouse moves away from or toward the user, the pointer moves up or down. By moving the mouse, the user can move the pointer to any location on the screen. Pen navigation is similar to mouse navigation, except that the user navigates by moving the pen without touching the input surface.

## Keyboard Navigation

Keyboard navigation requires a user to press specific keys and key combinations to move the *input focus* — the indication of where the input is being directed — to a particular location. The appearance of the input focus varies by context; in text, it appears as a text cursor or insertion point.



For more information about displaying the input focus, see Chapter 13, “Visual Design.”

## Basic Navigation Keys

The navigation keys are the four arrow keys and the HOME, END, PAGE UP, PAGE DOWN, and TAB keys. Pressed in combination with the CTRL key, a navigation key increases the movement increment. For example, where pressing RIGHT ARROW moves right one *character* in a text field, pressing CTRL+RIGHT ARROW moves right one *word* in the text field. Table 5.1 lists the common navigation keys and their functions. You can define additional keys for navigation.

Table 5.1 Basic Navigation Keys

Key	Moves cursor to	CTRL+key moves cursor to
LEFT ARROW	Left one unit.	Left one (larger) unit.
RIGHT ARROW	Right one unit.	Right one (larger) unit.
UP ARROW	Up one unit or line.	Up one (larger) unit.
DOWN ARROW	Down one unit or line.	Down one (larger) unit.
HOME	Beginning of line.	Beginning of data or file (topmost position).
END	End of line.	End of data or file (bottommost position).
PAGE UP	Up one screen (previous screen, same position).	Left one screen (or previous unit, if left is not meaningful).
PAGE DOWN	Down one screen (next screen, same position).	Right one screen (or next unit, if right is not meaningful).
TAB	Next field. (SHIFT+TAB moves in reverse order).	Next larger field.

Unlike mouse and pen navigation, keyboard navigation typically affects existing selections. Optionally, you can support the SCROLL LOCK key to enable scrolling navigation without affecting existing selections. If you do so, the keys scroll the appropriate increment.



For more information about keyboard navigation in secondary windows, such as dialog boxes, see Chapter 8, “Secondary Windows.”

## Selection

Selection is the primary means by which the user identifies objects in the interface. Consequently, the basic model for selection is one of the most important aspects of the interface.

Selection typically involves an overt action by the user to identify an object. This is known as an *explicit selection*. Once the object is selected, the user can specify an action for the object.

There are also situations where the identification of an object can be derived by inference or implied by context. An *implicit selection* works most effectively where the association of object and action is simple and visible. For example, when the user drags a scroll box, the user establishes selection of the scroll box and the action of



moving at the same time. Implicit selection may result from the relationships of a particular object. For example, selecting a character in a text document may implicitly select the paragraph of which the character is a part.

A selection can consist of a single object or multiple objects. Multiple selections can be *contiguous* — where the selection set is made up of objects that are logically adjacent to each other, also known as a *range selection*. A *disjoint selection* set is made up of objects that are spatially or logically separated.

Multiple selections may also be classified as *homogeneous* or *heterogeneous*, depending on the type or properties of the selected objects. Even a homogeneous selection might have certain aspects in which it is heterogeneous. For example, a text selection that includes bold and italic text can be considered homogeneous with respect to the basic object type (characters), but heterogeneous with respect to the values of its font properties. The homogeneity or heterogeneity of a selection affects the access of the operations or properties of the objects in the selection.

## Selection Feedback

Always provide visual feedback for explicit selections as the user makes the selection, so that the user can tell the effect of the selection operation. Display the appropriate selection appearance for each object included in the selection set. The form of selection appearance depends on the object and its context.

You may not need to provide immediate selection feedback for implicit selection; you can often indicate the effects of implicit selection in other ways. For example, when the user drags a scroll box, the scroll box moves with the pointer. Similarly, if the effect of selecting a word in a paragraph implicitly selects the paragraph, you would not use selection appearance on the entire paragraph, but rather reflect the implicit selection by including the paragraph's properties when the user chooses the Properties command.



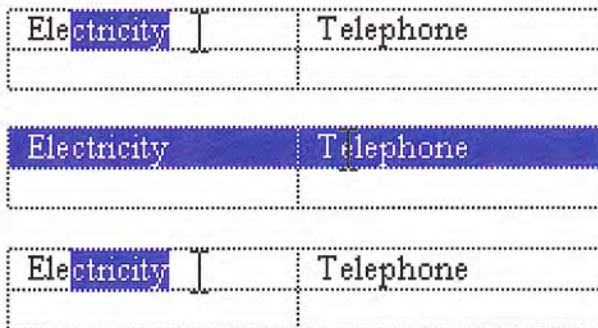
For more information about how to visually render the selection appearance of an object, see Chapter 13, “Visual Design.” For more information about how the context of an object can affect its selection appearance, see Chapter 11, “Working with OLE Embedded and OLE Linked Objects.”

## Scope of Selection

The *scope* of a selection is the area, extent, or region in which, if other selections are made, they will be considered part of the same selection set. For example, you can select two document icons in the same folder window. However, the selection of these icons is independent of the selection of the window's scroll bar, a menu, the window itself, or selections made in other windows. So, the selection scope of the icons is the area viewed through that window. Selections in different scopes are independent of each other. For example, selections in one window are typically independent of selections in other windows. Their windows define the scope of each selection independently. The scope of a selection is important because you use it to define the available operations for the selected items and how the operations are applied.

## Hierarchical Selection

Range selections typically include objects at the same level. However, you can also support a user's elevating a range selection to the next higher level if it extends beyond the immediate containment of the object (but within the same window). When the user adjusts the range back within the containment of the start of the range, return the selection to the original level. For example, extending a selection from within a cell in a table to the next cell, as shown in Figure 5.1, should elevate the selection from the character level to the cell level; adjusting the selection back within the cell should reset the selection to the character level.



**Figure 5.1** Hierarchical selection



## Mouse Selection

Selection with the mouse relies on the basic actions of clicking and dragging. In general, clicking selects a single item or location, and dragging selects a single range consisting of all objects logically included from the button-down to the button-up location. If you also support dragging for object movement, use keyboard-modified mouse selection or region selection to support multiple selection.

### Basic Selection

Support user selection using either mouse button. When the user presses the mouse button, establish the starting point, or *anchor point*, of a selection. If, while pressing the mouse button, the user drags the mouse, extend the selection to the object nearest the hot spot of the pointer. If, while continuing to hold the mouse button down, the user drags the mouse within the selection, reduce the selection to the object now nearest the pointer. Tracking the selection with the pointer while the mouse button continues to be held down allows the user to adjust a range selection dynamically. Use appropriate selection feedback to indicate the objects included in the selection.

The release of the mouse button ends the selection operation and establishes the *active end* of the selection. If the user presses mouse button 2 to make a selection, display the contextual pop-up menu for the selected objects when the user releases the mouse button.

The most common form of selection optimizes for the selection of a single object or a single range of objects. In such a case, creating a new selection within the scope of an existing selection (for example, within the same area of the window) cancels the selection of the previously selected objects. This allows simple selections to be created quickly and easily.

When using this technique, reset the selection when the user presses the mouse button and the pointer (hot spot) is outside (not on) any existing selection. If the pointer is over a selected item, however, don't cancel the former selection. Instead, determine the appropriate result according to whether the user pressed mouse button 1 or 2.



For more information about the appearance of selection feedback, see Chapter 13, “Visual Design.”



For more information about pop-up menus, see Chapter 7, “Menus, Controls, and Toolbars.”



If the user presses mouse button 1 and the pointer does not move from the button down point, the effect of the release of the mouse button is determined by the context of the selection. You can support whichever of the following best fits the nature of the user's task:

- The result may have no effect on the existing selection. This is the most common and safest effect.
- The object under the pointer may receive some special designation or distinction; for example, become the next anchor point or create a subselection.
- The selection can be reset to be only the object under the pointer.

If the user pressed mouse button 2, the selection is not affected, but you display a pop-up menu for selection.

Although selection is typically done by positioning the pointer over an object, it may be inferred based on the logical proximity of an object to a pointer. For example, when selecting text, the user can place the pointer on the blank area beyond the end of the line and the resulting selection is inferred as being the end of the line.

## Selection Adjustment

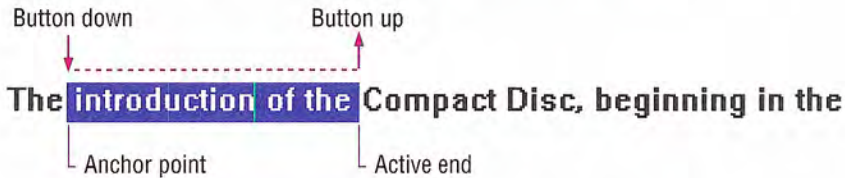
Selections are adjusted (elements added to or removed from the selection) using keyboard modifiers with the mouse. The CTRL key is the disjoint, or toggle, modifier. If the user presses the CTRL key while making a new selection, preserve any existing selection within that scope and reset the anchor point to the new mouse button-down location. Toggle the selection state of the object under the pointer — that is, if it is not selected, select it; if it is already selected, unselect it.

If a selection modified by the CTRL key is made by dragging, the selection state is applied for all objects included by the drag operation (from the anchor point to the current pointer location). This means if the first item included during the drag operation is not selected, select all objects included in the range. If the first item included was already selected, unselect it and all the objects included in the range regardless of their original state.

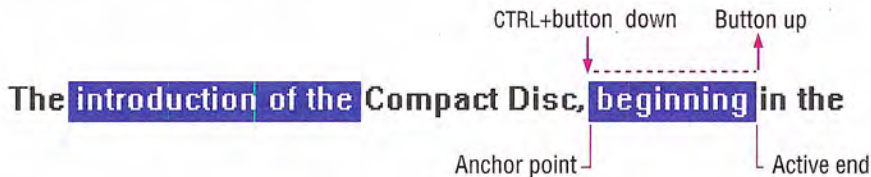


Disjoint selection techniques may not apply to all situations where you support selection.

For example, the user can make an initial selection by dragging.



The user can then press the CTRL key and drag to create a disjoint selection, resetting the anchor point.



The user must press the CTRL key before using the mouse button for a disjoint (toggle) selection. After a disjoint selection is initiated, it continues until the user releases the mouse button (even if the user releases the CTRL key before the mouse button).

The SHIFT key adjusts (or extends) a single selection or range selection. When the user presses the mouse button while holding down the SHIFT key, reset the active end of a selection from the anchor point to the location of the pointer. Continue tracking the pointer, resetting the active end as the user drags, similar to a simple range drag selection. When the user releases the mouse button, the selection operation ends. You should then set the active end to the object nearest to the mouse button release point. Do not reset the anchor point. It should remain at its current location.

Only the selection made from the current anchor point is adjusted. Any other disjoint selections are not affected unless the extent of the selection overlaps an existing disjoint selection.

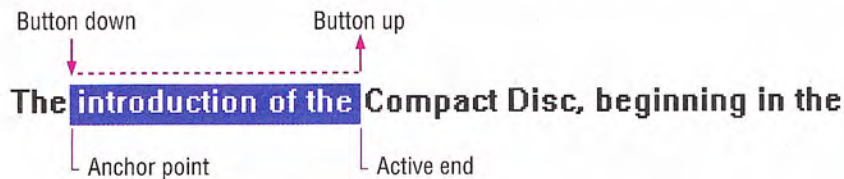
The effect on the selection state of a particular object is based on the first item included in the selection range. If the first item is already selected, select (not toggle the selection state of) all objects included in the range; otherwise, unselect (not toggle the selection state of) the objects included.

The user must press and hold down the SHIFT key before pressing the mouse button for the action to be interpreted as adjusting the selection. When the user begins adjusting a selection by pressing the SHIFT key, continue to track the pointer and adjust the selection (even if the user releases the modifier key) until the user releases the mouse button.

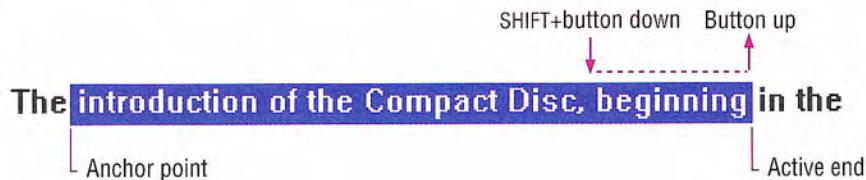
Pressing the SHIFT modifier key always adjusts the selection from the current anchor point. This means the user can always adjust the selection range of a single selection or CTRL key–modified disjoint selection. For example, the user can make a range selection by dragging.



The same result can be accomplished by making an initial selection.

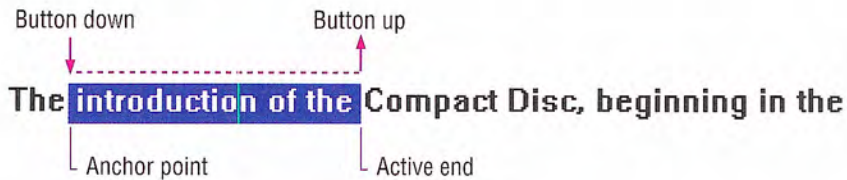


The user can adjust the selection with the SHIFT key and dragging.





The following sequence illustrates how the user can use the SHIFT key and dragging to adjust a disjoint selection. The user makes the initial selection by dragging.



The user presses the CTRL key and drags to create a disjoint selection.



The user can then extend the disjoint selection using the SHIFT key and dragging. This adjusts the selection from the anchor point to the button down point and tracks the pointer to the button up point.



Figure 5.2 shows how these same techniques can be applied within a spreadsheet.

1. The user selects four cells by dragging from A2 to B3.

	A	B	C
1	20	40	60
2	50	70	90
3	80	100	120
4	110	130	150
5	140	160	180
6	170	190	210

Anchor point (pointing to cell A2)

Active end (pointing to cell B3)

2. The user holds down the SHIFT key and clicks C4.

	A	B	C
1	20	40	60
2	50	70	90
3	80	100	120
4	110	130	150
5	140	160	180
6	170	190	210

Anchor point (pointing to cell A2)

Active end (pointing to cell C4)

3. The user holds down the CTRL key and clicks A6.

	A	B	C
1	20	40	60
2	50	70	90
3	80	100	120
4	110	130	150
5	140	160	180
6	170	190	210

Anchor point (pointing to cell A2)

4. The user holds down the SHIFT key and clicks C6.

	A	B	C
1	20	40	60
2	50	70	90
3	80	100	120
4	110	130	150
5	140	160	180
6	170	190	210

Anchor point (pointing to cell A2)

Active end (pointing to cell C6)

**Figure 5.2 Selection within a spreadsheet**

The following summarizes the mouse selection operations.

Operation	Mouse action
Select object (range of objects)	Click (drag)
Disjoint selection state of noncontiguous object (range of objects)	CTRL+click (drag)
Adjust current selection to object (or range of objects)	SHIFT+click (drag)

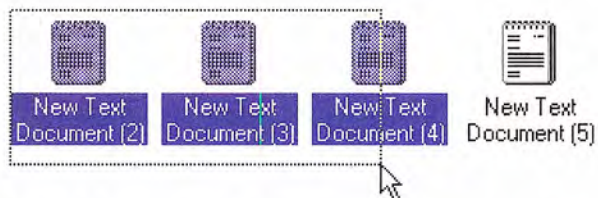


For more information about the mouse interface, including selection behavior, see Appendix A, "Mouse Interface Summary."

## Region Selection

In Z-ordered, or layered, contexts, in which objects may overlap, user selection can begin on the background (sometimes referred to as *white space*). To determine the range of the selection in such cases, a bounding outline (sometimes referred to as a marquee) is drawn. The outline is typically a rectangle, but other shapes (including freeform outline) are possible.

When the user presses the mouse button and moves the pointer (a form of selection by dragging), display the bounding outline, as shown in Figure 5.3. You set the selection state of objects included by the outline using the selection guidelines described in the previous sections, including operations that use the SHIFT and CTRL modifier keys.




**Figure 5.3** Region selection

You can use the context of your application and the user's task to determine whether an object must be totally enclosed or only intersected by the bounding region to be affected by the selection operation. Always provide good selection feedback during the operation to communicate to the user which method you support. When the user releases the mouse button, remove the bounding region, but retain the selection feedback.



## Pen Selection

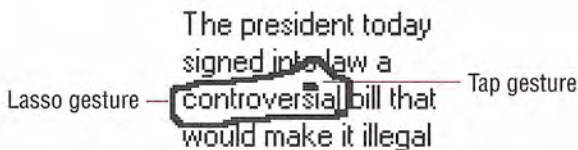
When the pen is being used as the pointing device, you can use the same selection techniques defined for the mouse. For example, in text input controls, you support user selection of text by dragging through it. Standard pen interfaces also support text selection using a special pen selection handle. In discrete object scenarios, like drawing programs, you support selection of individual objects by tapping or by performing region selection by dragging.

 For more information about supporting selection in pen-enabled controls, see the “Pen-Specific Editing Techniques” section later in this chapter.

In some specialized contexts, you can also use a press-hold-drag technique or the *lasso-tap* gesture to support selection of individual objects or ranges of objects. However, avoid implementing these techniques when it might interfere with primary operations such as direct manipulation. In general, consider using a pen selection handle or pen controls that include the selection handles before you consider these methods.

For the press-hold-drag technique, you switch to a selection mode when the user holds the pen tip at the same location for a predefined time-out. Then the user can drag to make a selection.

Lasso-tap involves making a circular gesture around the object, then tapping within the gesture. For example, in Figure 5.4, making the lasso-tap gesture selects the word “controversial.”



**Figure 5.4 A lasso-tap gesture**

In text contexts, base the selection on the extent of the lasso gesture and the character-word-paragraph granularity of the text elements covered. For example, if the user draws the lasso around a single character, select only that character. If the user draws the lasso around multiple characters within a word, select the entire word. If the gesture encompasses characters in multiple words, select the range of words logically included by the gesture. This reduces the need for the user to be precise.

## Keyboard Selection

Keyboard selection relies on the input focus to define selected objects. The input focus can be an insertion point, a dotted outline box, or some other cursor or visual indication of the location where the user is directing keyboard input.



For more information about input focus, see Chapter 13, “Visual Design.”

In some contexts, selection may be implicit with navigation. When the user presses a navigation key, you move the input focus to the location (as defined by the key) and automatically select the object at that location.

In other contexts, it may be more appropriate to move the input focus and require the user to make an explicit selection with the Select key. The recommended keyboard Select key is the SPACEBAR, unless this assignment directly conflicts with the specific context — in which case, you can use CTRL+SPACEBAR. (If this conflicts with your software, define another key that best fits the context.) In some contexts, pressing the Select key may also unselect objects; in other words, it will toggle the selection state of an object.

## Contiguous Selection

In text contexts, the user moves the insertion point to the desired location using the navigation keys. Set the anchor point at this location. When the user presses the SHIFT key with any navigation key (or navigation key combinations, such as CTRL+END), set that location as the active end of the selection and select all characters between the anchor point and the active end. (Do not move the anchor point.) If the user presses a subsequent navigation key, cancel the selection and move the insertion point to the appropriate location defined by the key. If the user presses LEFT ARROW or RIGHT ARROW keys, move the insertion point to the end of the former selection range. If UP ARROW or DOWN ARROW are used, move the insertion point to the previous or following line at the same relative location.

You can use this technique in other contexts, such as lists, where objects are logically contiguous. However, in such situations, the selection state of the objects logically included from the anchor point to the active end depend on the selection state of the object at, or



first traversed from, the anchor point. For example, if the object at the anchor point is selected, then select all the objects in the range regardless of their current state. If the object at the anchor point is not selected, unselect all the items in the range.

## Disjoint Selection

You use the Select key for supporting disjoint selections. The user uses navigation keys or navigation keys modified by the SHIFT key to establish the initial selection. The user can then use navigation keys to move to a new location and subsequently use the Select key to create an additional selection.

In some situations, you may prefer to optimize for selection of a single object or single range. In such cases, when the user presses a navigation key, reset the selection to the location defined by the navigation key. Creating a disjoint selection requires supporting the Add mode key (SHIFT+F8). In this mode, you move the insertion point when the user presses navigation keys without affecting the existing selections or the anchor point. When the user presses the Select key, toggle the selection state at the new location and reset the anchor point to that object. At any point, the user can use the SHIFT+navigation key combination to adjust the selection from the current anchor point.

When the user presses the Add mode key a second time, you toggle out of the mode, preserving the selections the user created in Add mode. But now, if the user makes any new selections within that selection scope, you return to the single selection optimization — canceling any existing selections — and reset the selection to be only the new selection.

## Selection Shortcuts

Double-clicking with mouse button 1 and double-tapping — its pen equivalent — is a shortcut for the default operation of an object. In text contexts, it is commonly assigned as a shortcut to select a word. When supporting this shortcut, select the word and the space following the word, but not the punctuation marks.



Double-clicking as a shortcut for selection only applies to text. In other contexts, it may perform other operations.



You can define additional selection shortcuts or techniques for specialized contexts. For example, selecting a column label may select the entire column. Because shortcuts cannot be generalized across the user interface, however, do not use them as the only way to perform a selection.

## Common Conventions for Supporting Operations

There are many ways to support operations for an object, including direct manipulation of the object or its control point (handle), menu commands, buttons, dialog boxes, tools, or programming. Support for a particular technique is not exclusive to other techniques. For example, the user can size a window by using the Size menu command and by dragging its border.

Design operations or commands to be *contextual*, or related to, the selected object to which they apply. That is, determine which commands or properties, or other aspects of an object, are made accessible by the characteristics of the object and its context (relationships). Often the context of an object may add to or suppress the traits of the object. For example, the menu for an object may include commands defined by the object's type and commands supplied by the object's current container.

### Operations for a Multiple Selection

When determining which operations to display for a multiple selection, use an intersection of the operations that apply to the members of that selection. The selection's context may add to or filter out the available operations or commands displayed to the user.

It is also possible to determine the effect of an operation for a multiple selection based upon a particular member of that selection. For example, when the user selects a set of graphic objects and chooses an alignment command, you can make the operation relative to a particular item identified in the selection.

Limit operations on a multiple selection to the scope of the selected objects. For example, deleting a selected word in one window should not delete selections in other windows (unless the windows are viewing the same selected objects).

## Default Operations and Shortcut Techniques

An object can have a default operation; a *default operation* is an operation that is assumed when the user employs a shortcut technique, such as double-clicking or drag and drop. For example, double-clicking a folder displays a window with the content of the folder. In text editing situations, double-clicking selects the word. The behavior differs because the default commands in each case differ: for a folder, the default command is Open; and for text, it is Select Word.

Similarly, when the user drags and drops an object at a new location with mouse button 1, there must be a default operation defined to determine the result of the operation. Dragging and dropping to some locations can be interpreted as a move, copy, link, or some other operation. In this case, the drop destination determines the default operation.

Shortcut techniques for default operations provide greater efficiency in the interface, an important factor for more experienced users. However, because they typically require more skill or experience and because not all objects may have a default operation defined, avoid shortcut techniques as the exclusive means of performing a basic operation. For example, even though double-clicking opens a folder icon, the Open command appears on its menu.



For more information about supporting default operations for drag and drop, see the “Transfer Operations” section later in this chapter; also see Chapter 11, “Working with OLE Embedded and OLE Linked Objects.”

## View Operations

Following are some of the common operations associated with viewing objects. Although these operations may not always be used with all objects, when supported, they should follow similar conventions.

Operation	Action
Open	Opens a primary window for an object. For container objects, such as folders and documents, this window displays the content of the object.
Close	Closes a window.
Properties	Displays the properties of an object in a window, typically in a property sheet window.
Help	Displays a window with the contextual Help information about an object.



When the user opens a new window, you should display it at the top of the Z order of its peer windows and activate it. Primary windows are typically peers with each other. Display supplemental or secondary windows belonging to a particular application at the top of their local Z order — that is, the Z order of the windows of that application, not the Z order of other primary windows.

If the user interacts with another window before the new window opens, the new window does not appear on top; instead, it appears where it would usually be displayed if the user activated another window. For example, if the user opens window A, then opens window B, window B appears on top of window A. If the user clicks back in window A before window B is displayed, however, window A remains active and at the top of the Z order; window B appears behind window A.

Whether opening a window allows the user to also edit the information in that window's view depends on a number of factors. These factors can include who the user is, the type of view being used, and the content being viewed.

After the user opens a window, re-executing the command that opened the window should activate the existing window instead of opening another instance of the window. For example, if the user chooses the Properties command for a selected object whose property sheet is already open, the existing property sheet is activated, rather than a second window opened.

Closing a window does not necessarily mean quitting the processes associated with the object being viewed. For example, closing a printer's window does not cancel the printing of documents in its queue. Quitting an application closes its windows, but closing a window does not necessarily quit an application. Similarly, you can use other commands in secondary windows which result in closing the window — for example, OK and Cancel. However, the effect of closing the window with a Close command depends on the context of the window. Avoid assuming that the Close command is the equivalent of the Cancel command.



For more information about opening windows, property sheets, and Help windows, see Chapter 6, "Windows," Chapter 8, "Secondary Windows," and Chapter 12, "User Assistance," respectively.



This guideline applies per user desktop. Two users opening a window for the same object on a network can each see separate windows for the object from their individual desktops.



If there are changes transacted in a window that have not yet been applied and the user chooses the Close command, and those changes will be lost if not applied, display a message asking whether the user wishes to apply or discard the changes or cancel the Close operation. If there are no outstanding changes or if pending changes are retained for the next time the window is opened, remove the window.

## View Shortcuts

Following are the recommended shortcut techniques for the common viewing commands.

Shortcut	Operation
CTRL+O	Opens a primary window for an object. For container objects, such as folders and documents, this window displays the content of the object.
ALT+F4	Closes a window.
F1	Displays a window with contextual Help information.
SHIFT+F1	Starts context-sensitive Help mode.
Double-click (button 1) or ENTER	Carries out the default command.
ALT+double-click or ALT+ENTER	Displays the properties of an object in a window, typically in a property sheet window.



For more information on reserved and recommended shortcut keys, see Appendix B, “Keyboard Interface Summary.”

Use double-clicking and the ENTER key to open a view of an object when that view command is the default command for the object. For example, double-clicking a folder opens the folder’s primary window. But double-clicking a sound object plays the sound; this is because the Open command is the default command for folders, and the Play command is the default command for sound objects.

## Editing Operations

Editing involves changing (adding, removing, replacing) some fundamental aspect about the composition of an object. Not all changes constitute editing of an object, though. For example, changing the view of a document to an outline or magnified view (which has no effect on the content of the document) is not editing. The following sections cover some of the common interface techniques for editing objects.

### Editing Text

Editing text requires that you target the input focus at the text to be edited. For mouse input, the input focus always coincides with the pointer (button down) location. For the pen, it is the location of the pointer when the pen touches the input surface. For the keyboard, the input focus is determined with the navigation keys. In all cases, the visual indication that a text field has the input focus is the presence of the text cursor, or insertion point.

### Inserting Text

Inserting text involves the user placing the insertion point at the appropriate location and then typing. For each character typed, your application should move the insertion point one character to the right (or left, depending on the language).

If the text field supports multiple lines, *wordwrap* the text; that is, automatically move text to the next line as the textual input exceeds the width of the text-entry area.

### Overtyping Mode

Overtyping is an optional text-entry behavior that operates similarly to the insertion style of text entry, except that you replace existing characters as new text is entered — with one character being replaced for each new character entered.

Use a block cursor that appears at the current character position to support overtyping mode, as shown in Figure 5.5. This looks the same as the selection of that character and provides the user with a visual cue about the difference between the text-entry modes.

The 1993 statistics are complete

**Figure 5.5 An overtyping cursor**

Use the INSERT key to toggle between the normal insert text-entry convention and overtyping mode.

## Deleting Text

The DELETE and BACKSPACE keys support deleting text. The DELETE key deletes the character to the right of the text insertion point. BACKSPACE removes the character to the left. In either case, move text in the direction of the deletion to fill the gap — this is sometimes referred to as *auto-joining*. Do not place deleted text on the Clipboard. For this reason, include at least a single-level undo operation in these contexts.

For a text selection, when the user presses DELETE or BACKSPACE, remove the entire block of selected text. Delete text selections when new text is entered directly or by a transfer command. In this case, replace the selected text by the incoming input.

## Handles

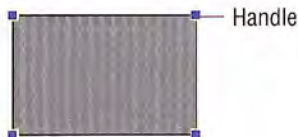
Objects may include special control points, called *handles*. You can use handles to facilitate certain types of operations, such as moving, sizing, scaling, cropping, shaping, or auto-filling. The type of handle you use depends on the type of object. For example, the title bar acts as a “move handle” for windows. The borders of the window act as “sizing handles.” For icons, the selected icon acts as its own “move handle.” In pen-enabled controls, special handles may appear for selection and access to the operations available for an object.

A common form of handle is a square box placed at the edge of an object, as shown in Figure 5.6.



For more information about pen handles, see the “Pen-Specific Editing Techniques” section later in this chapter.





**Figure 5.6** A graphic object with handles

When the handle’s interior is solid, the handle implies that it can perform a certain operation, such as sizing, cropping, or scaling. If the handle is “hollow,” the handle does not currently support an operation. You can use such an appearance to indicate selection even when an operation is not available.

 For more information about the design of handles, see Chapter 13, “Visual Design.”


## Transactions

A *transaction* is a unit of change to an object. The granularity of a transaction may span from the result of a single operation to that of a set of multiple operations. In an ideal model, transactions are applied immediately, and there is support for “rolling back,” or undoing, transactions. Because there are times when this is not practical, specific interface conventions have been established for committing transactions. If there are pending transactions in a window when it is closed, always prompt the user to ask whether to apply or discard the transactions.

Transactions can be committed at different levels, and a commitment made at one level may not imply a permanent change. For example, the user may change font properties of a selection of text, but these text changes may require saving the document file before the changes are permanent.

Use the following commands for committing transactions at the file level.

Command	Function
Save	Saves all interim edits, or checkpoints, to disk and begins a new editing session.
Save As	Saves the file (with all interim edits) to a new filename and begins a new editing session.
Close	Prompts the user to save any uncommitted edits. If confirmed, the interim edits are saved and the window is removed.


 Use the Save command in contexts where committing file transactions applies to transactions for an entire file, such as a document, and are committed at one time. It may not necessarily apply for transactions committed on an individual basis, such as record-oriented processing.

On a level with finer granularity, you can use the following commands for common handling transactions within a file.

<b>Command</b>	<b>Function</b>
Repeat	Duplicates the last/latest user transaction.
Undo	Reverses the last, or specified, transaction.
Redo	Restores the most recent, or specified, “undone” transaction.
OK	Commits any pending transactions and removes the window.
Apply	Commits any pending transactions, but does not remove the window.
Cancel	Discards any pending transactions and removes the window.


Following are the recommended commands for handling process transactions.

<b>Command</b>	<b>Function</b>
Pause	Suspends a process.
Resume	Resumes a suspended process.
Stop	Halts a process.

 Although you can use the Cancel command to halt a process, Cancel implies that the state will be restored to what it was before the process was initiated.

## Properties

Defining and organizing the properties of an application’s components are a key part of evolving toward a more data-centered design. Commands such as Options, Info, Summary Info, and Format often describe features that can be redefined as properties of a particular object (or objects). The Properties command is the common command for accessing the properties of an object; when the user chooses this command, display a secondary window with the properties of the object.

 For more information about property sheets, see Chapter 8, “Secondary Windows.”

Defining how to provide access to properties for visible or easily identifiable objects, such as a selection of text, cells, or drawing objects, is straightforward. It may be more difficult to define how to access properties of less tangible objects, such as paragraphs. In



some cases, you can include these properties by implication. For example, requesting the properties of a text selection can also provide access to the properties of the paragraph in which the text selection is included.

Another way to provide access to an object's properties is to create a representation of the object. For example, the properties of a page could be accessed through a graphic or other representation of the page in a special area (for example, the status bar) of the window.

Yet another technique to consider is to include specific property entries on the menu of a related object. For example, the pop-up menu of a text selection could include a menu entry for a paragraph. Or consider using the cascading submenu of the Properties command for individual menu entries, but only if the properties are not easily made accessible otherwise. Adding entries for individual properties can easily end up cluttering a menu.

The Properties command is not the exclusive means of providing access to the properties of an object. For example, folder views display certain properties of objects stored in the file system. In addition, you can use toolbar controls to display properties of selected objects.

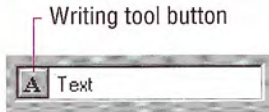
## Pen-Specific Editing Techniques

A pen is more than just a pointing device. When a standard pen device is installed, the system provides special interfaces and editing techniques.

### Editing in Pen-Enabled Controls

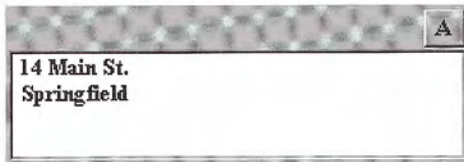
If a pen is installed, the system automatically provides a special interface, called the *writing tool*, to make text editing as easy as possible, enhance recognition accuracy, and streamline correction of errors. The writing tool interface, as shown in Figure 5.7, adds a button to your standard text controls. Because this effectively reduces the visible area of the text box, take this into consideration when designing their size.





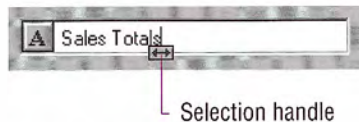
**Figure 5.7** A standard text box with writing tool button

Figure 5.8 shows how you can also add writing tool support for any special needs of your software, such as a multiline text box.



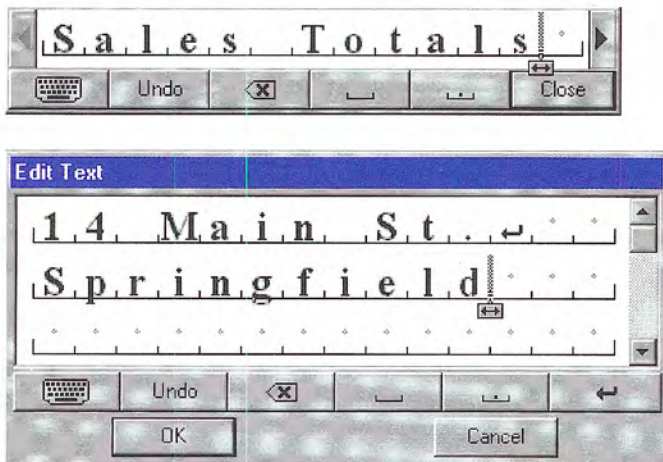
**Figure 5.8** Adding the writing tool button

When the text box control has the focus, a selection handle appears, as shown in Figure 5.9. The user can drag this handle to make a selection.



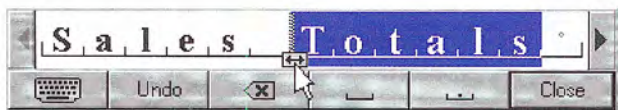
**Figure 5.9** Text box displaying a pen selection handle

Tapping the writing tool button with a pen (or clicking it with a mouse) presents a special text editing window, as shown in Figure 5.10. Within this window, the user can write text that is recognized automatically.



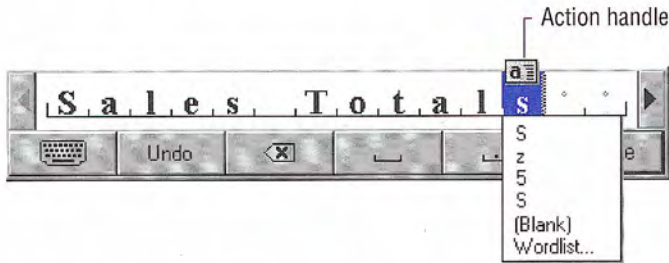
**Figure 5.10** Single and multiline writing tool windows

In the writing tool editing window, each character is displayed within a special cell. If the user selects text in the original text field, the writing tool window reflects that selection. The user can reset the selection to an insertion point by tapping between characters. This also displays a selection handle that can be dragged to select multiple characters, as shown in Figure 5.11.



**Figure 5.11** Selecting text with the selection handle

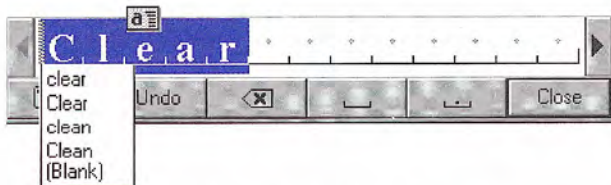
The user can select a single character in its cell by tapping, or double-tapping to select a word. When the user taps a single character, an action handle displays a list of alternative characters, as shown in Figure 5.12.



**Figure 5.12** An action handle with a list of alternatives


Choosing an alternative replaces the selected character and removes the list. Writing over a character or tapping elsewhere also removes the list. The new character replaces the existing one and resets the selection to an insertion point placed to the left of the new character.

The list also includes an item labeled Wordlist. When the user selects this choice, the word that contains the character becomes selected and a list of alternatives is displayed, as shown in Figure 5.13. This list also appears when the user selects a complete word by double-tapping. Choosing an alternative replaces the selected word.

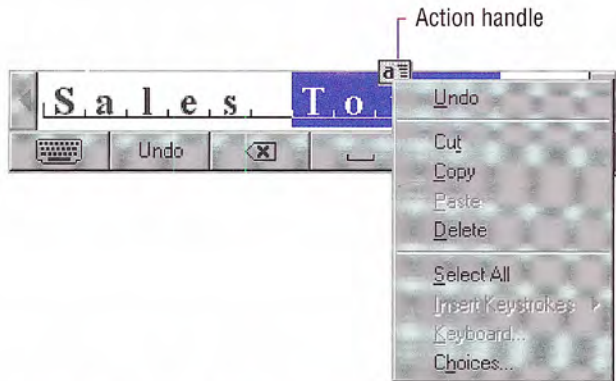


**Figure 5.13** Tapping displays a list of alternatives

Whenever a selection exists in the window, an action handle appears; the user can use it to perform other operations on the selected items. For example, using the action handle moves or copies the selection by dragging, or the pop-up menu for the selection can be accessed by tapping on the handle, as shown in Figure 5.14.

 For more information about pop-up menus, see Chapter 7, “Menus, Controls, and Toolbars.”





**Figure 5.14 Tapping on the handle displays a pop-up menu**

The buttons on the writing tool window provide for scrolling the text as well as common functions such as Undo, Backspace, Insert Space, Insert Period, and Close (for closing the text window). A multiline writing tool window includes Insert New Line.

The writing tool window also provides a button for access to an onscreen keyboard as an alternative to entering characters with the pen, as shown in Figure 5.15. The user taps the button with the corresponding keyboard glyph on it and the writing tool onscreen keyboard pop-up window replaces the normal writing tool window.



**Figure 5.15 The writing tool onscreen keyboard window**

The writing tool “remembers” its previous use — for text input or as an onscreen keyboard — and opens in the appropriate editing window when subsequently used. In addition, note that when the user displays a writing tool window, it gets the input focus, so avoid using the loss of input focus to a field as an indication that the user is finished with that field or that all text editing occurs directly within a text box.












## Pen Editing Gestures


The pen, when used as a pointing device, supports editing techniques defined for the mouse. When used as a writing device, the pen supports gestures for editing. Gestures (except for Undo) operate positionally, acting upon the objects on which they are drawn. If the user draws a gesture on an unselected object, it applies to that object, even if a selection exists elsewhere within the same selection area. Any pending selections become unselected. If a user draws a gesture over both selected and unselected objects, however, it applies only to the selected ones. If a gesture is drawn over only one element of the selection, it applies to the entire selection. If the gesture is drawn in empty space (on the background), it applies to any existing selection within that selection scope. If no selection exists, the gesture has no effect.

For most gestures, the hot spot of the gesture determines specifically which object the gesture applies to. If the hot spot occurs on any part of a selection, it applies to the whole selection.

Table 5.2 lists the common pen editing gestures. For these gestures, the hot spot of the gesture is the area inside the circle stroke of the gesture.

**Table 5.2 Pen Editing Gestures**

Gesture	Name	Operation
	circled-check	Edit (displays the writing tool editing window) for text; Properties for all other objects.
	circled-c	Copy
	circled-d	Delete (or Clear)
	circled-m	Menu
	circled-n	New line
	circled-p	Paste
	circled-s	Insert space
	circled-t	Insert tab
	circled-u	Undo
	circled-x	Cut
	circled-^	Insert text

 These gestures may be localized in certain international versions. In Japanese versions, the circled-k gesture is used to convert Kana to Kanji.

## Transfer Operations

Transfer operations are operations that involve (or can be derived from) moving, copying, and linking objects from one location to another. For example, printing an object is a form of a transfer operation because it can be defined as copying an object to a printer.

Three components make up a transfer operation: the object to be transferred, the destination of the transfer, and the operation to be performed. You can define these components either explicitly or implicitly, depending on which interaction technique you use.



The operation defined by a transfer is determined by the destination. Because a transfer may have several possible interpretations, you can define a default operation and other optimal operations, based on information provided by the source of the transfer and the compatibility and capabilities of the destination. For example, attempting to transfer an object to a container can result in one of the following alternatives:

- Rejecting the object.
- Accepting the object.
- Accepting a subset or transformed form of the object (for example, extract its content or properties but discard its present containment, or convert the object into a new type).

Most transfers are based on one of the following three fundamental operations.

Operation	Description
Move	Relocates or repositions the selected object. Because it does not change the basic identity of an object, a move operation is not the same as copying an object and deleting the original.
Copy	Makes a duplicate of an object. The resulting object is independent of its original. Duplication does not always produce an identical clone. Some of the properties of a duplicated object may be different from the original. For example, copying an object may result in a different name or creation date. Similarly, if some component of the object restricts copying, then only the unrestricted elements may be copied.
Link	Creates a connection between two objects. The result is usually an object in the destination that provides access to the original.

There are two different methods for supporting the basic transfer interface: the command method and the direct manipulation method.

## Command Method

The command method for transferring objects uses the Cut, Copy, and Paste commands. Place these commands on the Edit drop-down menu and on the pop-up menu for a selected object. You can also include toolbar buttons to support these commands.

To transfer an object, the user:

1. Makes a selection.
2. Chooses either Cut or Copy.
3. Navigates to the destination (and sets the insertion location, if appropriate).
4. Chooses a Paste operation.

Cut removes the selection and transfers it (or a reference to it) to the Clipboard. Copy duplicates the selection (or a reference to it) and transfers it to the Clipboard. Paste completes the transfer operation. For example, when the user chooses Cut and Paste, remove the selection from the source and relocate it to the destination. For Copy and Paste, insert an independent duplicate of the selection and leave the original unaffected. When the user chooses Copy and Paste Link or Paste Shortcut, insert an object at the destination that is linked to the source.

Choose a form of Paste command that indicates how the object will be transferred into the destination. Use the Paste command by itself to indicate that the object will be transferred as native content. You can also use alternative forms of the Paste command for other possible transfers, using the following general form.

**Paste** [*type name*] [**as** *type name* | **to** *object name*]

For example, Paste Cells as Word Table, where [*type name*] is Cells and Word Table is the converted type name.

The following summarizes common forms of the Paste command.

Command	Function
Paste	Inserts the object on the Clipboard as native content (data).
Paste [ <i>type name</i> ]	Inserts the object on the Clipboard as an OLE embedded object. The OLE embedded object can be activated directly within the destination.
Paste [ <i>type name</i> ] as Icon	Inserts the object on the Clipboard as an OLE embedded object. The OLE embedded object is displayed as an icon.
Paste Link	Inserts a data link to the object that was copied to the Clipboard. The object's value is integrated or transformed as native content within the destination, but remains linked to the original object so that changes to it are reflected in the destination.
Paste Link to [ <i>object name</i> ]	Inserts an OLE linked object, displayed as a picture of the object copied to the Clipboard. The representation is linked to the object copied to the Clipboard so that any changes to the original source object will be reflected in the destination.
Paste Shortcut	Inserts an OLE linked object, displayed as a shortcut icon, to the object that was copied to the Clipboard. The representation is linked to the object copied to the Clipboard so that any changes to the original source object will be reflected in the destination.
Paste Special	Displays a dialog box that gives the user explicit control over how to insert the object on the Clipboard.



For more information about object names, including their type name, see Chapter 10, "Integrating with the System."



For more information about these Paste command forms and the Paste Special dialog box, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."



Use the destination's context to determine what form(s) of the Paste operation to include based on what options it can offer to the user, which in turn may depend on the available forms of the object that its source location object provides. It can also be dependent on the nature or purpose of the destination. For example, a printer defines the context of transfers to it.

Typically, you will need only Paste and Paste Special commands. The Paste command can be dynamically modified to reflect the destination's default or preferred form by inserting the transferred object — for example, as native data or as an OLE embedded object. The Paste Special command can be used to handle any special forms of transfer. Although, if the destination's context makes it reasonable to provide fast access to another specialized form of transfer, such as Paste Link, you can also include that command.

Use the destination's context also to determine the appropriate side effects of the Paste operation. You may also need to consider the type of object being inserted by the Paste operation and the relationship of that type to the destination. The following are some common scenarios:

- When the user pastes into a destination that supports a specific insertion location, replace the selection in the destination with the transferred data. For example, in text or list contexts, where the selection represents a specific insertion location, replace the destination's active selection. In text contexts where there is an insertion location, but there is no existing selection, place the insertion point after the inserted object.
- For destinations with nonordered or Z-ordered contexts where there is no explicit insertion point, add the pasted object and make it the active selection. Also use the destination's context to determine where to place the pasted object. Consider any appropriate user contextual information. For example, if the user chooses the Paste command from a pop-up menu, you can use the pointer's location when the mouse button is clicked to place the incoming object. If the user supplies no contextual clues, place the object at the location that best fits the context of the destination — for example, at the next grid position.

- If the new object is automatically connected (linked) to the active selection (for example, table data and a graph), you may insert the new object in addition to the selection and make the inserted object the new selection.

You also use context to determine whether to display an OLE embedded or OLE linked object as content (view or picture of the object's internal data) or as an icon. For example, you can decide what presentation to display based on what Paste operation the user selects; Paste Shortcut implies pasting an OLE link as an icon. Similarly, the Paste Special command includes options that allow the user to specify how the transferred object should be displayed. If there is no user-supplied preference, the destination application defines the default. For documents, you typically display the inserted OLE object as in its content presentation. If icons better fit the context of your application, make the default Paste operation display the transferred OLE object as an icon.

The execution of a Paste command should not affect the content of the Clipboard. This allows data on the Clipboard to be pasted multiple times, although subsequent Paste operations should always result in copies of the original. However, a subsequent Cut or Copy command replaces the last entry on the Clipboard.

## Direct Manipulation Method

The command method is useful when a transfer operation requires the user to navigate between source and destination. However, for many transfers, direct manipulation is a natural and quick method. In a direct manipulation transfer, the user selects and drags an object to the desired location, but because this method requires motor skills that may be difficult for some users to master, avoid using it as the exclusive transfer method. The best interfaces support the transfer command method for basic operations and direct manipulation transfer as a shortcut technique.

When a pen is being used as a pointing device, or when it drags an action handle, it follows the same conventions as dragging with mouse button 1. For pens with barrel buttons, use the barrel+drag action as the equivalent of dragging with mouse button 2. There is no keyboard interface for direct manipulation transfers.



You can support direct manipulation transfers to any visible object. The object (for example, a window or icon) need not be currently active. For example, the user can drop an object in an inactive window. The drop action activates the window. If an inactive object cannot accept a direct manipulation transfer, it (or its container) should provide feedback to the user.

How the transferred object is integrated and displayed in the drop destination is determined by the destination's context. A dropped object can be incorporated either as native data, an OLE object, a partial form of the object — such as its properties — or a transformed object. You determine whether to add to or replace an existing selection based on the context of the operation, using such factors as the formats available for the object, the destination's purpose, and any user-supplied information such as the specific location that the user drops or commands (or modes) that the user has selected. For example, an application can supply a particular type of tool for copying the properties of objects.

## Default Drag and Drop

*Default drag and drop* transfers an object using mouse button 1. How the operation is interpreted is determined by what the destination defines as the appropriate default operation. As with the command method, the destination determines this based on information about the object (and the formats available for the object) and the context of the destination itself. Avoid defining a destructive operation as the default. When that is unavoidable, display a message box to confirm the intentions of the user.

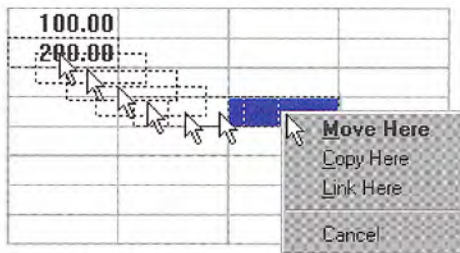
Using this transfer technique, the user can directly transfer objects between documents defined by your application as well as to system resources, such as folders and printers. Support drag and drop following the same conventions the system supports: the user presses button 1 down on an object, moves the mouse while holding the button down, and then releases the button at the destination. For the pen, the destination is determined by the location where the user lifts the pen tip from the input surface.



The most common default transfer operation is Move, but the destination (dropped on object) can reinterpret the operation to be whatever is most appropriate. Therefore, you can define a default drag and drop operation to be another general transfer operation such as Copy or Link, a destination specific command such as Print or Send To, or even a specialized form of transfer such as Copy Properties.

## Nondefault Drag and Drop

*Nondefault drag and drop* transfers an object using mouse button 2. In this case, rather than executing a default operation, the destination displays a pop-up menu when the user releases the mouse button, as shown in Figure 5.16. The pop-up menu contains the appropriate transfer completion commands.



**Figure 5.16** A nondefault drag and drop operation

The destination always determines which transfer completion commands to include on the resulting pop-up menu, usually factoring in information about the object supplied by the source location.

The form for nondefault drag and drop transfer completion verbs follows similar conventions as the Paste command. Use the common transfer completion verbs, Move Here, Copy Here, and Link Here, when the object being transferred is native data of the destination. When it is not, include the type name. You can also display alternative completion verbs that communicate the context of the destina-

tion; for example, a printer displays a Print Here command. For commands that support only a partial aspect or a transformation of an object, use more descriptive indicators — for example, Copy Properties Here, or Transpose Here.

Use the following general form for nondefault drag and drop transfer commands.

[*Command Name*] [*type name* | *object name*] **Here** [**as** *type name*]

The following summarizes common forms for nondefault transfer completion commands.

<b>Command</b>	<b>Function</b>
Move Here	Moves the selected object to the destination as native content (data).
Copy Here	Creates a copy of the selected object in the destination as native content.
Link Here	Creates a data link between the selected object and the destination. The original object's value is integrated or transformed as native data within the destination, but remains linked to the original object so that changes to it are reflected in the destination.
Move [ <i>type name</i> ] Here Copy [ <i>type name</i> ] Here	Moves or copies the selected object as an OLE embedded object. The OLE embedded object is displayed in its content presentation and can be activated directly within the destination.
Link [ <i>type name</i> ] Here	Creates an OLE linked object displayed as a picture of the selected object. The representation is linked to the selected object so that any changes to the original object will be reflected in the destination.

*(Continued)*

Command	Function
Move [ <i>type name</i> ] Here as Icon	Moves or copies the selected object as an OLE embedded object and displays it as an icon.
Copy [ <i>type name</i> ] Here as Icon	Moves or copies the selected object as an OLE embedded object and displays it as an icon.
Create Shortcut Here	Creates an OLE linked object to the selected object; displayed as a shortcut icon. The representation is linked to the selected object so that any changes to the original object will be reflected in the destination.

Define and appropriately display one of the commands in the pop-up menu to be the default drag and drop command. This is the command that corresponds to the effect of dragging and dropping with mouse button 1.



For more information about how to display default menu commands, see Chapter 13, "Visual Design."

## Canceling a Drag and Drop Transfer

When a user drags and drops an object back on itself, interpret the action as cancellation of a direct manipulation transfer. Similarly, cancel the transfer if the user presses the ESC key during a drag transfer. In addition, include a Cancel command in the pop-up menu of a nondefault drag and drop action. When the user chooses this command, cancel the operation.

## Differentiating Transfer and Selection When Dragging

Because dragging performs both selection and transfer operations, provide a convention that allows the user to differentiate between these operations. The convention you use depends on what is most appropriate in the current context of the object, or you can provide specialized handles for selection or transfer. The most common technique uses the location of the pointer at the beginning of the drag operation. If the pointer is within an existing selection, interpret the drag to be a transfer operation. If the drag begins outside of an existing selection, on the background's white space, interpret the drag as a selection operation.



## Scrolling When Transferring by Dragging

When the user drags and drops an object from one scrollable area (such as a window, pane, or list box) to another, some tasks may require transferring the object outside the boundary of the area. Other tasks may involve dragging the object to a location not currently in view. In this latter case, it is convenient to automatically scroll the area (also known as *automatic scrolling* or autoscroll) when the user drags the object to the edge of that scrollable area. You can accommodate both these behaviors by using the velocity of the dragging action. For example, if the user is dragging the object slowly at the edge of the scrollable area, you scroll the area; if the object is being dragged quickly, do not scroll.

To support this technique during a drag operation, you sample the pointer's position at the beginning of the drag each time the mouse moves, and on an application-set timer (every 100 milliseconds recommended). If you use OLE drag and drop support, you need not set a timer. Store each value in an array large enough to hold at least three samples, replacing existing samples with later ones. Then calculate the pointer's velocity based on at least the last two locations of the pointer.

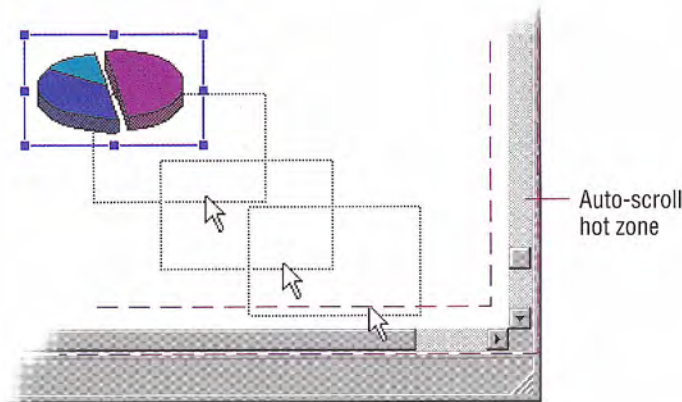
To calculate the velocity, sum the distance between the points in each adjacent sample and divide the total by the sum of the time elapsed between samples. Distance is the absolute value of the difference between the x and y locations, or  $(\text{abs}(x1 - x2) + \text{abs}(y1 - y2))$ . Multiply this by 1024 and divide it by the elapsed time to produce the velocity. The 1024 multiplier prevents the loss of accuracy caused by integer division.

You also predefine a hot zone along the edges of the scrollable area and a scroll time-out value. Use twice the width of a vertical scroll bar or height of a horizontal scroll bar to determine the width of the hot zone.

During the drag operation, scroll the area if the following conditions are met: the user moves the pointer within the hot zone, the current velocity is below a certain threshold velocity, and the scrollable area is able to scroll in the direction associated with the hot zone it is in. The recommended threshold velocity is 20 pixels per second. These conventions are illustrated in Figure 5.17.




Distance as implemented in this algorithm is not true Cartesian distance. This implementation uses an approximation for purposes of efficiency, rather than using the square root of the sum of the squares,  $(\text{sqrt}((x1 - x2)^2 + (y1 - y2)^2))$ , which is more computationally expensive.



**Figure 5.17 Automatic scrolling based on velocity of dragging**


The amount you scroll depends on the type of information and reasonable scrolling distance. For example, for text, you typically scroll vertically one line at a time. Consider using the same scrolling granularity that is provided for the scroll bar arrows.

To support continuous scrolling, determine what the scroll frequency you want to support — for example, four lines per second. After using a velocity check to initiate auto-scrolling, set a timer — for example, 100 milliseconds. When the timer expires, determine how long it has been since the last time you scrolled. If the elapsed time is greater than your scrolling frequency, scroll another unit. If not, reset your timer and check again when the timer completes.

 For more information about scrolling, see Chapter 6, “Windows.”

## Transfer Feedback

Because transferring objects is one of the most common user tasks, providing appropriate feedback is an important design factor. Inconsistent or insufficient feedback can result in user confusion.

 For more information about designing transfer feedback, see Chapter 13, “Visual Design.”

## Command Method Transfers

For a command method transfer, remove the selected object visually when the user chooses the Cut command. If there are special circumstances that make removing the object’s appearance impractical, you can instead display the selected object with a special appearance to inform the user that the Cut command was completed, but that the



object's transfer is pending. For example, the system displays icons in a checkerboard dither to indicate this state. You will also need to restore the visual state of the object if the user chooses Cut or Copy for another object before choosing a Paste command, effectively canceling the pending Cut command. The user will expect Cut to remove a selected object, so carefully consider the impact of inconsistency if you choose this alternate feedback.

The Copy command requires no special feedback. A Paste operation also requires no further feedback than that already provided by the insertion of the transferred object. However, if you did not remove the display of the object and used an alternate representation when the user chose the Cut command, you must remove it now.

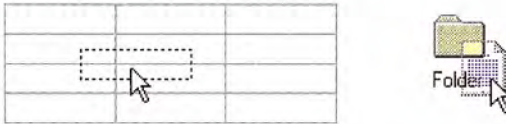
## Direct Manipulation Transfers

During a direct manipulation transfer operation, provide visual feedback for the object, the pointer, and the destination. Specifically:

- Display the object with selected appearance while the view it appears in has the focus. To indicate that the object is in a transfer state, you can optionally display the object with some additional appearance characteristics. For example, for a move operation, you can use the checkerboard dithered appearance used by the system to indicate when an icon is Cut. Change this visual state based on the default completion operation supported by the destination the pointer is currently over. Retain the representation of the object at the original location until the user completes the transfer operation. This not only provides a visual cue to the nature of the transfer operation, it provides a convenient visual reference point.
- Display a representation of the object that moves with the pointer. Use a presentation that provides the user with information about how the information will appear in the destination and that does not obscure the context of the insertion location. For example, when transferring an object into a text context, it is important that the insertion point not be obscured during the drag operation. A translucent or outline representation, as shown in Figure 5.18,



works well because it allows the underlying insertion location to be seen while also providing information about the size, position, and nature of the object being dragged.



**Figure 5.18** Outline and translucent representations for transfer operations

- The object's existing source location provides the transferred object's initial appearance, but any destination can change the appearance. Design the presentation of the object to provide feedback as to how the object will be integrated by that destination. For example, if an object will be embedded as an icon, display the object as an icon. If the object will be incorporated as part of the native content of the destination, then the presentation of the object that the destination displays should reflect that. For example, if a table being dragged into a document will be incorporated as a table, the representation could be an outline or translucent form of the table. On the other hand, if the table will be converted to text, display the table as a representation of text, such as a translucent presentation of the first few words in the table.
- Display the pointer appropriate to the context of the destination, usually used for inserting objects. For example, when dragging an object into a text editing context such that the object will be inserted between characters, display the usual text editing pointer (sometimes called the I-beam pointer).
- Display the interpretation of the transfer operation at the lower right corner of the pointer, as shown in Figure 5.19. No additional glyph is required for a move operation. Use a plus sign (+) when the transfer is a copy operation. Use the shortcut arrow graphic for linking.



**Figure 5.19** Pointers - move, copy, and link operations

- Use visual feedback to indicate the receptivity of potential destinations. You can use selection highlighting and optionally animate or display a representation of the transfer object in the destination. Optionally, you can also indicate when a destination cannot accept an object by using the “no drop” pointer when the pointer is over it, as shown in Figure 5.20.

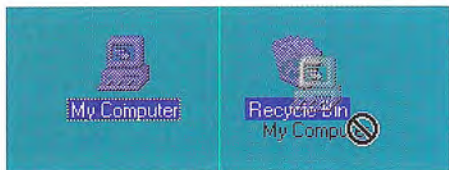



Figure 5.20 A “no drop “ pointer

## Specialized Transfer Commands

In some contexts, a particular form of a transfer operation may be so common, that introducing an additional specialized command is appropriate. For example, if copying existing objects is a frequent operation, you can include a Duplicate command. Following are some common specialized transfer commands.

Command	Function
Delete	Removes an object from its container. If the object is a file, the object is transferred to the Recycle Bin.
Clear	Removes the content of a container.
Duplicate	Copies the selected object.
Print	Prints the selected object on the default printer.
Send To	Displays a list of possible transfer destinations and transfers the selected object to the user selected destination.

 Delete and Clear are often used synonymously. However, they are best differentiated by applying Delete to an object and Clear to the container of an object.

## Shortcut Keys for Transfer Operations

Following are the defined shortcut techniques for transfer operations.

Shortcut	Operation
CTRL+X	Performs a Cut command.
CTRL+C	Performs a Copy command.
CTRL+V	Performs a Paste command.
CTRL+drag	Toggles the meaning of the default direct manipulation transfer operation to be a copy operation (provided the destination can support the copy operation). The modifier may be used with either mouse button.
ESC	Cancels a drag and drop transfer operation.



For more information about reserved and recommended shortcut key assignments, see Appendix B, “Keyboard Interface Summary.”

Because of the wide use of these command shortcut keys throughout the interface, do not reassign them to other commands.

## Creation Operations

Creating new objects is a common user action in the interface. Although applications can provide the context for object creation, avoid considering an application’s interface as the exclusive means of creating new objects. Creation is typically based on some predefined object or specification and can be supported in the interface in a number of ways.

### Copy Command

Making a copy of an existing object is the fundamental paradigm for creating new objects. Copied objects can be modified and serve as prototypes for the creation of other new objects. The transfer model conventions define the basic interaction techniques for copying objects. Copy and Paste commands and drag and drop manipulation provide this interface.




## New Command

The New command facilitates the creation of new objects. New is a command applied to a specific object, automatically creating a new instance of the object's type. The New command differs from the Copy and Paste commands in that it is a single command that generates a new object.


## Insert Command

The Insert command works similarly to the New command, except that it is applied to a container to create a new object, usually of a specified type, in that container. In addition to inserting native types of data, use the Insert command to insert objects of different types. By supporting OLE, you can support the creation of a wide range of objects. In addition, objects supported by your application can be inserted into data files created by other OLE applications.

 For more information about inserting objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."

## Using Controls

You can use controls to support the automatic creation of new objects. For example, in a drawing application, buttons are often used to specify tools or modes for the creation of new objects, such as drawing particular shapes or controls. Buttons can also be used to insert OLE objects.

 For more information about using buttons to create new objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."

## Using Templates

A *template* is an object that automates the creation of a new object. To distinguish its purpose, display a template icon as a pad with the small icon of the type of the object to be created, as shown in Figure 5.21.



**Figure 5.21** A template icon

Define the New command as the default operation for a template object; this starts the creation process, which may either be automatic or request specific input from the user. Place the newly created object in the same location as the container of the template. If circumstances make that impractical, place the object in a common location, such as the desktop, or, during the creation process, include a prompt that allows a user to specify some other destination. In the former situation, display a message box informing the user where the object will appear.

## Operations on Linked Objects

A *link* is a connection between two objects that represents or provides access to another object that is in another location in the same container or in a different, separate container. The components of this relationship include the link source (sometimes referred to as the referent) and the link or linked object (sometimes referred to as the reference). A linked object often has operations and properties independent of its source. For example, a linked object's properties can include attributes like update frequency, the path description of its link source, and the appearance of the linked object. The containers in which they reside provide access to and presentation of commands and properties of linked objects.

Links can be presented in various ways in the interface. For example, a *data link* propagates a value between two objects, such as between two cells in a worksheet or a series of data in a table and a chart. *Jumps* (also referred to as hyperlinks) provide navigational access to another object. An *OLE linked object* provides access to any operation available for its link source and also supplies a presentation of the link source. A shortcut icon is a link, displayed as an icon.



For more information about OLE linked objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects." For more information about jumps, see Chapter 12, "User Assistance."



When the user transfers a linked object, store both the absolute and relative path to its link source. The absolute path is the precise description of its location, beginning at the root of its hierarchy. The relative path is the description of its location relative to its current container.

The destination of a transfer determines whether to use the absolute or relative path when the user accesses the link source through the linked object. The relative path is the most common default path. However, regardless of which path you use, if it fails, use the alternative path. For example, if the user copies a linked object and its link source to another location, the result is a duplicate of the linked object and the link source. The relative path for the duplicate linked object is the location of the duplicate of the link source. The absolute path for the duplicate linked object is the description of the location of the initial link source. Therefore, when the user accesses the duplicate of the linked object, its inferred connection should be with the duplicate of the link source. If that connection fails — for example, because the user deletes the duplicate of the linked source — use the absolute path, the connection to the original link source.

Optionally, you can make the preferred path for a linked object a field in the property sheet for linked object. This allows the user to choose whether to have a linked object make use of the absolute or relative path to its link source.

When the user applies a link operation to a linked object, link to the linked object rather than its linked source. That is, linking a linked object results in a linked object linked to a linked object. If such an operation is not valid or appropriate - for example, because the linked object provides no meaningful context - then disable any link commands or options when the user selects a linked object.

Activation of a linked object depends on the kind of link. For example, a single click can activate a jump. However, a single click only results in selecting a data link or an OLE linked object. If you use a single click to do anything other than select the linked object,



distinguish the object by either presenting it as a button control, displaying the hand pointer (as shown in Figure 5.22) when the user moves the pointer over the linked object, or both. These techniques provide feedback to the user that the clicking involves more than selection.

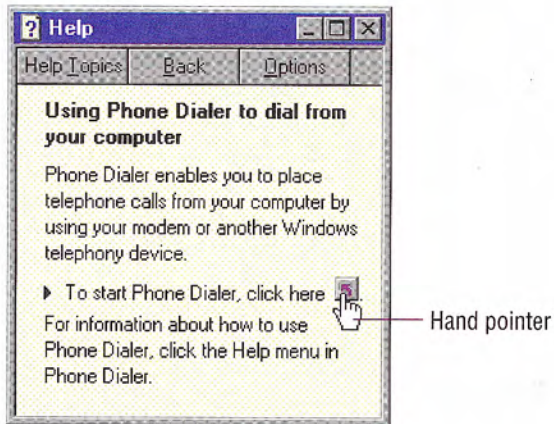


Figure 5.22 The hand pointer







## **Part II**

### Windows Interface Components






# Windows



Windows provide the fundamental way a user views and interacts with data. Consistency in window design is particularly important because it enables users to easily transfer their learning skills and focus on their tasks rather than learn new conventions. This chapter describes the common window types and presents guidelines for general appearance and operation.

## Common Types of Windows

Because windows provide access to different types of information, they are classified according to common usage. Interacting with objects typically involves a *primary window* in which most primary viewing and editing activity takes place. In addition, multiple supplemental *secondary windows* can be included to allow users to specify parameters or options, or to provide more specific details about the objects or actions included in the primary window.

 For more information about secondary windows, see Chapter 8, “Secondary Windows.”

## Primary Window Components

A typical primary window consists of a frame (or border) which defines its extent, and a title bar which identifies what is being viewed in the window. If the viewable content of the window exceeds the current size of the window, scroll bars are used. The window can also include other components like menu bars, toolbars, and status bars.

Figure 6.1 shows the common components of a primary window.

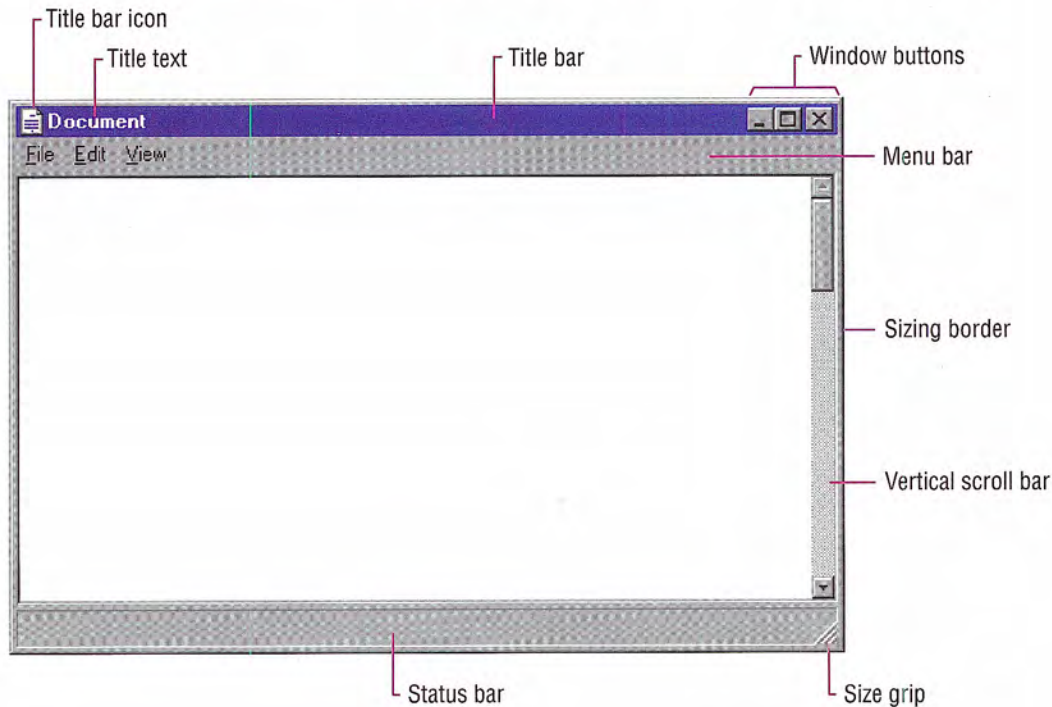


Figure 6.1 A primary window

## Window Frames

Every window has a boundary that defines its shape. A sizable window has a distinct border that provides control points (handles) for resizing the window using direct manipulation. If the window cannot be resized, the border coincides with the edge of the window.

## Title Bars

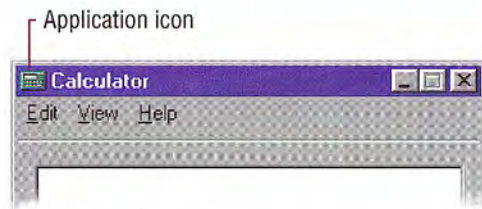
At the top edge of the window, inside its border, is the *title bar* (also referred to as the caption or caption bar), which extends across the width of the window. The title bar identifies what the window is viewing. It also serves as a control point for moving the window and an access point for commands that apply to the window and its



associated view. For example, clicking on the title bar with mouse button 2 displays the pop-up menu for the window. Pressing the ALT+SPACEBAR key combination also displays the pop-up menu for the window.

## Title Bar Icons

A primary window includes the small version of the object's icon. The small icon appears in the upper left corner of the title bar and represents the object being viewed in the window. If the window represents a "tool" or utility application (that is, an application that does not create, load, and save its own data files), use the small version of the application's icon in its title bar, as shown in Figure 6.2.




**Figure 6.2** "Tool" title bar


If the application creates, loads and saves documents or data files and the window represents the view of one of its files, use the icon that represents its document or data file type in the title bar, as shown in Figure 6.3. Display the data file icon even if the user has not saved the file yet, rather than displaying the application icon and then the data file icon once the user saves the file.

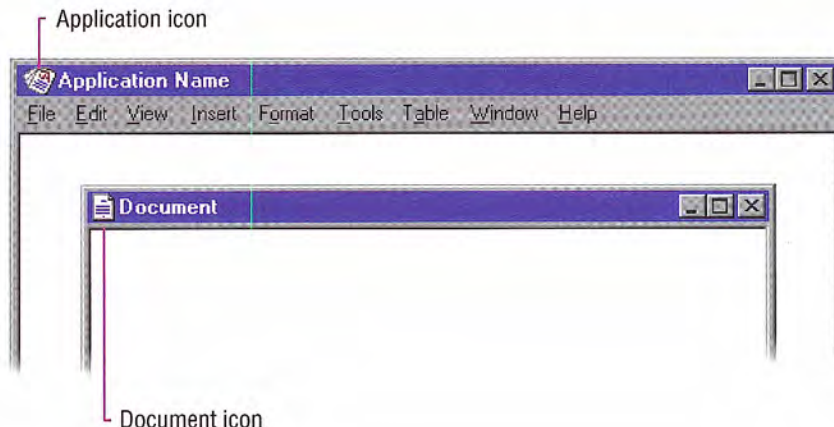


**Figure 6.3** Document title bar

If an application uses the multiple document interface (MDI) design, place the application's icon in the parent window's title bar, and place an icon that reflects the application's data file type in the child window's title bar, as shown in Figure 6.4.

 For information about how to register icons for your application and data file types, see Chapter 10, "Integrating with the System." For more information about designing icons, see Chapter 13, "Visual Design."


 For more information about MDI, see Chapter 9, "Window Management."



**Figure 6.4 MDI application and document title bars**

However, when a user maximizes the child window, and you hide its title bar and merge its title information with the parent, display the icon from the child window's title bar in the menu bar of the parent window. If multiple child windows are open within the MDI parent window, display only the icon from the active (topmost) child window.

When the user clicks the title bar icon with mouse button 2, display the pop-up menu for the object. Typically, the menu contains a similar set of commands that are available for the icon from which the window was opened, except that Close replaces Open. Also define Close as the default command, so when the user double-clicks the title bar icon, the window closes. Clicking elsewhere with button 2 on the title bar displays the pop-up menu for the window.

 When the user clicks the title bar icon with mouse button 1, the system also displays the pop-up menu for the window. However, this behavior is only supported for backward compatibility with Windows 3.1. Avoid documenting it as the primary way to access the pop-up menu for the window. Instead, document the use of button 2 as the correct way to display the pop-up menu for the window.

## Title Text

The window title text identifies the name of the object being viewed in the window. It should always correspond to the icon of the type you display in the title bar. It should also match the label of the icon in the file system that represents the object. For example, if the user opens a data file named "My Document" in the resulting window,


you display the icon for that document type followed by the name of the data file. You may also include the name of the application in use; however, if it is used, display the name of the data file first, followed by a dash and the application name, as shown in Figure 6.5.



**Figure 6.5 Title text order: document name — application name**

If the window represents a “tool” application that does not create or edit its own data files, such as the Windows Calculator, display the application’s name, as displayed for the application’s icon label, in the title bar. If the tool application operates as a utility for other files created by other applications — such as a special viewer or browser application — where the view displayed is not the primary open view of the file, or where the “tool” application requires an additional specification to indicate its context — such as the Windows Explorer — place the name of the application first, then include a dash and the specification text. For example, the title text of the Windows Explorer includes the name of the current container displayed in the browser.

For an MDI application, use the application’s name in the parent window and the data file’s name in the child windows. When the user maximizes the file’s child window, format the title text following the same convention as a tool application, with the application’s name first, followed by the data filename, as shown in Figure 6.6.

 The order of the document (or data) filename and application name differs from the Windows 3.1 guidelines. The new convention is better suited for the design of a data-centered interface.






**Figure 6.6 Document follows application name for maximized child window**

When the user directly opens an application that displays a new data file, supply a name for the file and place it in the title bar, even if the user has not saved the file yet. Use the type name — for example Document (*n*), Sheet (*n*), Chart (*n*), where *n* is a number, as in Document (1). Make certain that the proposed name does not conflict with an existing name in the current directory. Also use this name as the proposed default filename for the object in the Save As dialog box. If it is impractical or inappropriate to supply a default name, display a placeholder in the title, such as (Untitled).

Follow the same convention if your application includes a New command that creates new files. Avoid prompting the user for a name. Instead, you can supply a Save As dialog box that allows the user to confirm or change your proposed name when they save or close the file or attempt to create a new file.

Display a filename in the title bar exactly as it appears to the user in the file system, using both uppercase and lowercase letters. However, avoid displaying the file's extension or the path in the title bar. This information is not meaningful for most users and can make it more difficult for them to identify the file. However, because the system provides an option for users to display filename extensions, use the system-supplied functions to format a filename, which will display the filename appropriately based on the user's preference.

If your application supports multiple windows for viewing the same file, you may use the title text to distinguish between the views — but use a convention that will not be confused as part of the filename. For example, you may want to append *:n*, where *n* represents the instance of the window, as in Document:2. Make certain you do not include this view designation as part of the filename you supply in the Save As dialog box.

 For more information about type names, see Chapter 10, “Integrating with the System.” For more information about the Save As dialog box, see Chapter 8, “Secondary Windows.”

 The **GetFileTitle** and **SHGetFileInfo** functions automatically format names correctly. For more information about these functions, see the documentation included in the Microsoft Win32 Software Development Kit (SDK).

If the name of the displayed object in the window changes — for example, when the user edits the name in the object's property sheet — update the title text to reflect that change. Always try to maintain a clear association between an object and its open window.

The title text and title bar icon should always represent the outmost container — the object that was opened — even if the user selects an embedded object or navigates the internal hierarchy of the object being viewed in the window. If you need an additional specification to clarify what the user is viewing, place this specification after the filename and separate it clearly from the filename, such as enclosing it in parentheses — for example, My HardDisk (C:). Because the system now supports long filenames, avoid additional specification whenever possible. Complex or verbose additions to the title text also make it more difficult for the user to easily read and identify the window.

When the width of the window does not allow you to display the complete title text, you may abbreviate the title text, being careful to maintain the essential information that allows the user to quickly identify the window.



For more information about abbreviating names, see Chapter 10, "Integrating with the System."

Avoid drawing directly into the title bar or adding other controls. Such added items can make reading the name in the title difficult, particularly because the size of the title bar varies with the size of the window. In addition, the system uses this area for displaying special controls. For example, in some international versions of Windows, the title area provides information or controls associated with the input of certain languages.

## Title Bar Buttons





Include command buttons associated with the common commands of the primary window in the title bar. These act as shortcuts to specific window commands. Clicking a title bar button with mouse button 1 invokes the command associated with the command button. Optionally, you can also support clicking a title bar command button



with mouse button 2 to display the pop-up menu for the window. For the pen, tapping a window button invokes its associated command, and optionally you may support barrel-tapping it (or using the pen menu gesture) to display the pop-up menu for the window.

In a typical situation, one or more of the following buttons appear in a primary window (provided that the window supports the respective functions).

### 6.1 Title Bar Buttons

Button	Command	Operation
	Close	Closes the window.
	Minimize	Minimizes the window.
	Maximize	Maximizes the window.
	Restore	Restores the window.

When displaying these buttons, use the following guidelines:

- When a command is not supported by a window, do not display its corresponding button.
- The Close button always appears as the rightmost button. Leave a gap between it and any other buttons.
- The Minimize button always precedes the Maximize button.
- The Restore button always replaces the Maximize button or the Minimize button when that command is carried out.



The system does not support the inclusion of the context-sensitive Help button available for secondary windows. Applications wishing to provide this functionality can do so by including a Help toolbar button. Similarly, avoid including Maximize, Minimize, and Restore buttons in the title bars of secondary windows because those commands do not apply to those windows.



# Basic Window Operations

The basic operations for a window include: activation and deactivation, opening and closing, moving, sizing, scrolling, and splitting. The following sections describe these operations.

## Activating and Deactivating Windows

While the system supports the display of multiple windows, the user generally works within a single window at a time. This window is called the *active window*. The active window is typically at the top of the window Z order. It is also visually distinguished by its title bar that is displayed in the active window title color. All other windows are *inactive* with respect to the user's input; that is, while other windows can have ongoing processes, only the active window receives the user's input. The title bar of an inactive window displays the system inactive window color. Your application can query the system for the current values for the active title bar color and the inactive title bar color.

The user activates a primary window by switching to it; this inactivates any other primary windows. To activate a window with the mouse or pen, the user clicks or taps on any part of the window, including its interior. If the window is minimized, the user clicks (taps) the button representing the window in the taskbar. From the keyboard, the system provides the ALT+TAB key combination for switching between primary windows. The SHIFT+ALT+TAB key also switches between windows, but in reverse order. (The system also supports ALT+ESC for switching between windows.) The reactivation of a window should not affect any pre-existing selection within it; the selection and focus are restored to the previously active state.

When the user reactivates a primary window, the window and all its secondary windows come to the top of the window order and maintain their relative positions. If the user activates a secondary window, its primary window comes to the top of the window order along with the primary window's other secondary windows.



For more information about using the **GetSysColor** function to access the COLOR\_ACTIVECAPTION and COLOR\_INACTIVECAPTION constants, see the documentation included in the Win32 SDK.


When a window becomes inactive, hide the selection feedback (for example, display of highlighting or handles) of any selection within it to prevent confusion over which window is receiving keyboard input. A direct manipulation transfer (drag and drop) is an exception. Here, you can display transfer feedback if the pointer is over the window during the drag operation. Do not activate the window unless the user releases the mouse button (the pen tip is lifted) in that window.


## Opening and Closing Windows

When the user opens a primary window, include an entry for it on the taskbar. If the window has been opened previously, restore the window to its size and position when it was last closed. If possible and appropriate, reinstate the other related view information, such as selection state, scroll position, and type of view. When opening a primary window for the first time, open it to a reasonable default size and position as best defined by the object or application. For details about storing state information in the system registry, see Chapter 10, “Integrating with the System.”

Because display resolution and orientation varies, your software should not assume a fixed display size, but rather adapt to the shape and size defined by the system. If you use standard system interfaces the system automatically places your windows relative to the current display configuration.

Opening the primary window activates that window and places it at the top of the window order. If the user attempts to open a primary window that is already open within the same desktop, activate the existing window using the following recommendations. If the existing window is minimized, restore it when you activate it.

 Only primary windows, not secondary windows, should include an entry on the taskbar.

 The **SetWindowPlacement** function is an example of a system interface that will automatically place windows correctly relative to the current display. For more information about this function, see the documentation included in the Win32 SDK.



<b>File type</b>	<b>Action when repeating an Open operation</b>
Document or data file	Activates the existing window of the object and displays it at the top of the window Z order.
Application file	Displays a message box indicating that an open window of that application already exists and offers the user the option to switch to the open window or to open another window. Either choice activates the selected window and brings it to the top of the window Z order.
Document file that is already open in an MDI application window	Activates the existing window of the file. Its MDI parent window comes to the top of the window Z order, and the file appears at the top of the Z order within its MDI parent window.
Document file that is not already open, but its associated MDI application is already running (open)	Opens a new instance of the file's associated MDI application at the top of the window Z order and displays the child window for the file. Optionally, as an alternative, displays a message box indicating that an open window of that application already exists and offers the user the option to use the existing window or to open a new parent window.



For more information about MDI, see Chapter 9, "Window Management."

The user closes a primary window by clicking (for a pen, tapping the screen) the Close button in the title bar or choosing the Close command from the window's pop-up menu. Although the system supports double-clicking (with a pen, double-tapping the screen) on the title bar icon as a shortcut for closing the window for compatibility with previous versions of Windows, avoid documenting this as the primary way to close a primary window. Instead, document the Close button.

When the user chooses the Close command, if your application does not automatically save these changes and pending transactions or edits that have not yet been saved to file remain, display a message asking the user whether to save any changes, discard any changes, or cancel the Close operation before closing the window. If there are no pending transactions, just close the window. Follow this same convention for any other command that results in closing the primary window (for example, Exit or Shut Down).



For more information about supporting the Close command, see Chapter 5, "General Interaction Techniques."



When closing the primary window, close any of its dependent secondary windows as well. The design of your application determines whether closing the primary window also ends the application processes. For example, closing the window of a text document typically halts any application code or processes remaining for inputting or formatting text. However, closing the window of a printer has no effect on the jobs in the printer's queue. In both cases, closing the window removes its entry from the taskbar.

## Moving Windows

The user can move a window either by dragging its title bar using the mouse or pen or by using the Move command on the window's pop-up menu. On most configurations, an outline representation moves with the pointer during the operation, and the window is redisplayed in the new location after the completion of the move. (The system also provides a display property setting that redraws the window dynamically as it is moved.) After choosing the Move command, the user can move the window with the keyboard interface by using arrow keys and pressing the ENTER key to end the operation and establish the window's new location. Never allow the user to reposition a window so that it cannot be accessed.

A window need not be active before the user can move it. The action of moving the window implicitly activates it.

Moving a window may clip or reveal information shown in the window. In addition, activation can affect the view state of the window — for example, the current selection can be displayed. However, when the user moves a window, avoid making any changes to the content being viewed in that window.

## Resizing Windows

Make your primary windows resizable unless the information displayed in the window is fixed, such as in the Windows Calculator program. The system provides several conventions that support user resizing of a window.

## Sizing Borders

The user resizes a primary window by dragging the sizing border with the mouse or pen at the edge of a window or by using the Size command on the window's menu. An outline representation of the window moves with the pointer. (On some configurations, the system may include a display option to dynamically redraw the window as it is sized.) After completing the size operation, the window assumes its new size. Using the keyboard, the user can size the window by choosing the Size command, using the arrow keys, and pressing the ENTER key.

A window does not need to be active before the user can resize it. The action of sizing the window implicitly makes it active, and it remains active after the sizing operation.

When the user resizes a window to be smaller, you must determine how to display the information viewed in that window. Use the context and type of information to help you choose your approach. The most common approach is to clip the information. However, in other situations where you want the user to see as much information as possible, you may want to consider using different methods, such as rewrapping or scaling the information. Use these variations carefully because they may not be consistent with the resizing behavior of most windows. In addition, avoid these methods when readability or maintaining the structural relationship of the information is important.

Although the size of a primary window can vary, based on the user's preference, you can define a window's maximum size. When defining this size, consider the reasonable usage within the window, and the size and orientation of the screen.

## Maximizing Windows

Although the user may be able to directly resize a window to its maximum size, the Maximize command optimizes this operation. Include this command on a window's pop-up menu, and as the Maximize command button in the title bar of the window.

Maximizing a window increases the size of the window to its largest, optimum size. The system default setting for the maximum size is as large as the display, excluding the space used by the taskbar (or other application-defined desktop toolbars). For an MDI child window, the default maximize size fills the parent window. But, you can define the size to be less or, in some cases, more than the default dimensions.


When the user maximizes a window, replace the Maximize button with a Restore button. Then, disable the Maximize command and enable the Restore command on the pop-up menu for the window.

## Minimizing Windows

Minimizing a window reduces it to its smallest size. To support this command, include it on the pop-up menu for the window and as the Minimize command button in the title bar of the window.

For primary windows, minimizing removes the window from the screen, but leaves its entry in the taskbar. For MDI child windows, the window resizes to a minimum size within its parent window. To minimize a window, the user chooses the Minimize command from the window's pop-up menu or the Minimize command button on the title bar.

When the user minimizes a window, disable the Minimize command on the pop-up menu and enable the Restore command.

 Because the representation of minimized windows changed in Microsoft Windows 95, using an icon as the way to reflect state information may not be appropriate. As an alternative, you may want to consider creating a status indicator for the taskbar. For more information about status notification, see Chapter 10, "Integrating with the System."

## Restoring Windows

Support the Restore command to restore a window to its previous size and position after the user has maximized or minimized the window. For maximized windows, enable this command on the window's pop-up menu and replace the Maximize button with the Restore button in the title bar of the window.



For minimized windows, also enable the Restore command in the pop-up menu of the window. The user restores a minimized primary window to its former size and position by clicking (for pens, tapping the screen) on its button in the taskbar that represents the window, selecting the Restore command on the pop-up menu of the window's taskbar button, or using the ALT+TAB (or the SHIFT+ALT+TAB) key combination.

## Size Grip

When you define a sizable window, you may include a size grip. A *size grip* is a special handle for sizing a window. It is not exclusive to the sizing border. To size the window, the user drags the grip and the window resizes following the same conventions as the sizing border.

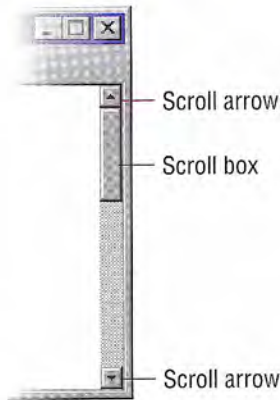
Always locate the size grip in the lower right corner of the window. Typically, this means you place the size grip at the right end of a horizontal scroll bar or the bottom of a vertical scroll bar. However, if you include a status bar in the window, display the size grip at the far corner of the status bar instead. Never display the size grip in both locations at the same time.



For more information about the use of the size grip in a status bar, see Chapter 7, "Menus, Controls, and Toolbars."

## Scrolling Windows


When the information viewed in a window exceeds the size of that window, the window should support scrolling. Scrolling enables the user to view portions of the object that are not currently visible in a window. Scrolling is commonly supported through the use of a scroll bar. A *scroll bar* is a rectangular control consisting of *scroll arrows*, a *scroll box*, and a *scroll bar shaft*, as shown in Figure 6.7.



**Figure 6.7 Scroll bar and its components**


You can include a vertical scroll bar, a horizontal scroll bar, or both. Align a scroll bar with the vertical or horizontal edge of the window orientation it supports. If the content is never scrollable in a particular direction, do not include a scroll bar for that direction.

The common practice is to display scroll bars if the view requires some scrolling under any circumstances. If the window becomes inactive or resized so that its content does not require scrolling, you should continue to display the scroll bars. While removing the scroll bars when the window is inactive potentially allows the display of more information and feedback about the state of the window, it also requires the user to explicitly activate the window to scroll. Consistently displaying scroll bars provides a more stable environment.

 Scroll bars are also available as separate window components. For more information about scroll bar controls, see Chapter 7, “Menus, Controls, and Toolbars.”

## Scroll Arrows

Scroll arrow buttons appear at each end of a scroll bar, pointing in opposite directions away from the center of the scroll bar. The scroll arrows point in the direction that the window “moves” over the data. When the user clicks (for pens, tapping the screen) a scroll arrow, the data in the window moves, revealing information in the direction of the arrow in appropriate increments. The granularity of the increment depends on the nature of the content and context, but it is typically based on the size of a standard element. For example, you can

 The default system support for scroll bars does not disable the scroll arrow buttons when the region or area is no longer scrollable in this direction. However, it does provide support for you to disable the scroll arrow button under the appropriate conditions.

use one line of text for vertical scrolling, one row for spreadsheets. You can also use an increment based a fixed unit of measure. Which-ever convention you choose, maintain the same scrolling increment throughout a window. The objective is to provide an increment that provides smooth but efficient scrolling. When a window cannot be scrolled any further in a particular direction, disable the scroll arrow corresponding to that direction.

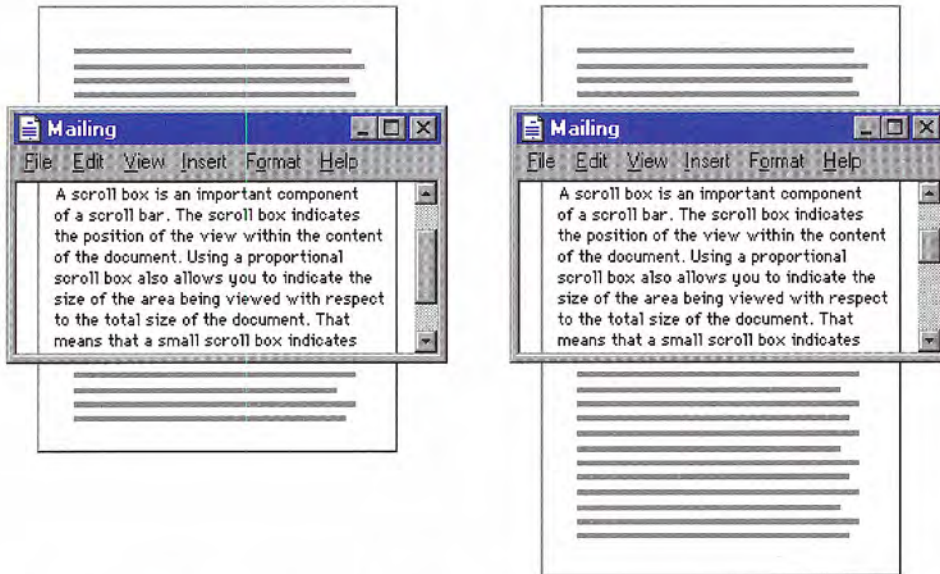
When scroll arrow buttons are pressed and held, they exhibit a special auto-repeat behavior. This action causes the window to continue scrolling in the associated direction as long as the pointer remains over the arrow button. If the pointer is moved off the arrow button while the user presses the mouse button, the auto-repeat behavior stops and does not continue unless the pointer is moved back over the arrow button (also when the pen tip is moved off the control).

## Scroll Box

The scroll box, sometimes referred to as the elevator, thumb, or slider, moves along the scroll bar to indicate how far the visible portion is from the top (for vertical scroll bars) or from the left edge (for horizontal scroll bars). For example, if the current view is in the middle of a document, the scroll box in the vertical scroll bar is displayed in the middle of the scroll bar.

The size of the scroll box can vary to reflect the difference between what is visible in the window and the entire content of the file, as shown in Figure 6.8.





**Figure 6.8 Proportional relationship between scroll box and content**

For example, if the content of the entire document is visible in a window, the scroll box extends the entire length of the scroll bar, and the scroll arrows are disabled. Make the minimum size of the scroll box no smaller than the width of a window's sizing border.

The user can also scroll a window by dragging the scroll box. Update the view continuously as the user moves the scroll box. If you cannot support scrolling at a reasonable speed, you can scroll the information at the end of the drag operation as an alternative.

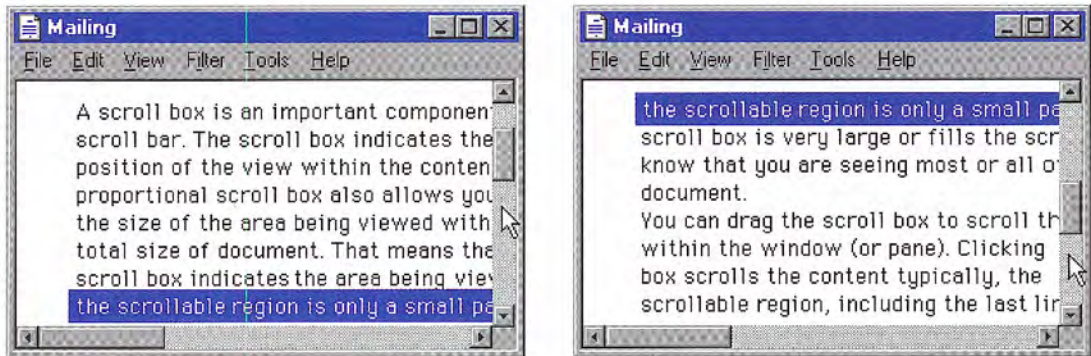
If the user starts dragging the scroll box and then moves the pointer outside of the scroll bar, the scroll box returns to its original position. The distance the user can move the pointer off the scroll bar before the scroll box snaps back to its original position is proportional to the width of the scroll bar. If dragging ends at this point, the scroll action is canceled — that is, no scrolling occurs. However, if the user moves the pointer back within the scroll-sensitive area, the scroll box returns to tracking the pointer movement. This behavior allows the user to scroll without having to remain within the scroll bar and to selectively cancel the initiation of a drag-scroll operation.

Dragging the scroll box to the end of the scroll bar implies scrolling to the end of that dimension; this does not always mean that the area cannot be scrolled further. If your application's document structure extends beyond the data itself, you can interpret dragging the scroll box to the end of its scroll bar as moving to the end of the data rather than the end of the structure. For example, a typical spreadsheet exceeds the data in it — that is, the spreadsheet may have 65,000 rows, with data only in the first 50 rows. This means you can implement the scroll bar so that dragging the scroll box to the bottom of the vertical scroll bar scrolls to the last row containing data rather than the last row of the spreadsheet. The user can use the scroll arrow buttons to scroll further to the end of the structure. This situation also illustrates why disabling the scroll arrow buttons can provide important feedback so that the user can distinguish between scrolling to the end of data from scrolling to the end of the extent or structure. In the example of the spreadsheet, when the user drags the scroll box to the end of the scroll bar, the arrow would still be shown as enabled because the user can still scroll further, but it would be disabled when the user scrolls to the end of the spreadsheet.

## Scroll Bar Shaft

The scroll bar shaft not only provides a visual context for the scroll box, it also serves as part of the scrolling interface. Clicking in the scroll bar shaft should scroll the view an equivalent size of the visible area in the direction of the click. For example, if the user clicks in the shaft below the scroll box in a vertical scroll bar, scroll the view a distance equivalent to the height of the view. Where possible, allow overlap from the previous view, as shown in Figure 6.9. For example, if the user clicks below the scroll box, the bottom line becomes the top line of scrolled view. The same thing applies for clicking above the scroll box and horizontal scrolling. These conventions provide the user with a common reference point.





**Figure 6.9** Scrolling with the scroll bar shaft by a screenful

Pressing and holding mouse button 1 with the pointer in the shaft auto-repeats the scrolling action. If the user moves the pointer outside the scroll-sensitive area while pressing the button, the scrolling action stops. The user can resume scrolling by moving the pointer back into the scroll bar area. (This behavior is similar to the effect of dragging the scroll box.)

## Automatic Scrolling

The techniques previously summarized describe the explicit ways for scrolling. However, the user can also scroll as a secondary result of another user action. This type of scrolling is called *automatic scrolling*. The situations in which to support automatic scrolling are as follows:

- When the user begins or adjusts a selection and drags it past the edge of the scroll bar or window, scroll the area in the direction of the drag.
- When the user drags an object and approaches the edge of a scrollable area, scroll the area following the recommended auto-scroll conventions covered in Chapter 5, “General Interaction Techniques.” Base the scrolling increment on the context of the destination and, if appropriate, on the size of the object being dragged.



- When the user enters text from the keyboard at the edge of a window or moves or copies an object into a location at the edge of a window, the view should scroll to allow the user to focus on the inserted information. The amount to scroll depends on context. For example, for text, vertically scroll a single line at a time. When scrolling horizontally, scroll in units greater than a single character to prevent continuous or uneven scrolling. Similarly, when the user transfers a graphic object near the edge of the view, base scrolling on the size of the object.
- If an operation results in a selection or moves the cursor, scroll the view to display the new selection. For example, for a Find command that selects a matching object, scroll the object into view because usually the user wants to focus on that location. In addition, other forms of navigation may cause scrolling. For example, completing an entry field in a form may result in navigating to the next field. In this case, if the field is not visible, the form can scroll to display it.


## Keyboard Scrolling

Use navigation keys to support scrolling with the keyboard. When the user presses a navigation key, the cursor moves to the appropriate location. For example, in addition to moving the cursor, pressing arrow keys at the edge of a scrollable area scrolls in the corresponding direction. Similarly, the PAGE UP and PAGE DOWN keys are comparable to clicking in the scroll bar shaft, but they also move the cursor.

Optionally, you can use the SCROLL LOCK key to facilitate keyboard scrolling. In this case, when the SCROLL LOCK key is toggled on and the user presses a navigation key, scroll the view without affecting the cursor or selection.

## Placing Adjacent Controls

It is sometimes convenient to locate controls or status bars adjacent to a scroll bar and position the end of the scroll bar to accommodate them. Take care when placing adjacent elements; too many can make it difficult for users to scroll, particularly if you reduce the scroll bar too much. If you need a large number of controls, consider using a conventional toolbar instead.

 For more information about toolbars, see Chapter 7, “Menus, Controls, and Toolbars.”

## Splitting Windows

A window can be split into two or more separate viewing areas, which are called *panes*. For example, a split window allows the user to examine two parts of a document at the same time. You can also use a split window to display different, yet simultaneous, views of the same information, as shown in Figure 6.10.

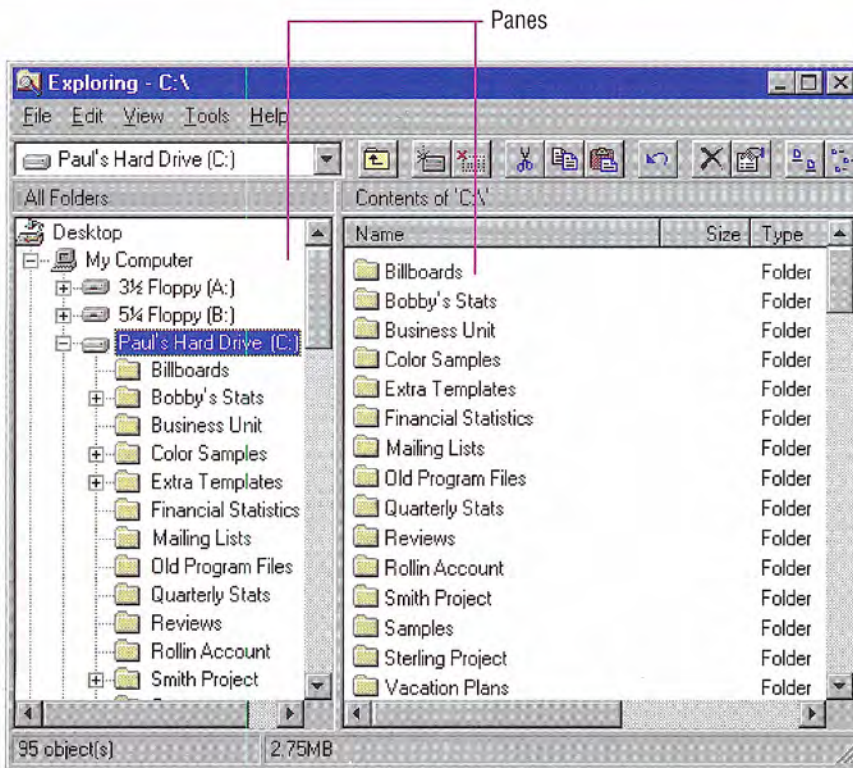
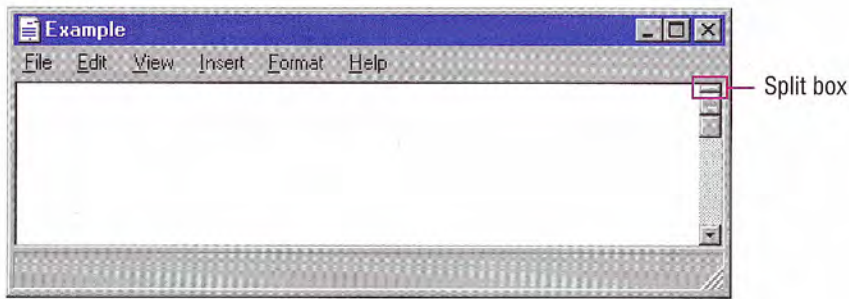


Figure 6.10 A split window

While you can use a split window pane to view the contents of multiple files or containers at the same time, displaying these in separate windows typically allows the user to better identify the files as individual elements. When you need to present views of multiple files as a single task, consider window management techniques, such as MDI.

The panes that appear in a split window can be implemented either as part of a window's basic design or as a user-configurable option. To support splitting a window that is not presplit by design, include a split box. A *split box* is a special control placed adjacent to the end of a scroll bar that splits or adjusts the split of a window. The size of the split box should be just large enough for the user to successfully target it with the pointer; the default size of a size handle, such as the window's sizing border, is a good guideline. Locate the split box at the top of the up arrow button of the vertical scroll bar, as shown in Figure 6.11, or to the left of the left arrow button of a horizontal scroll bar.

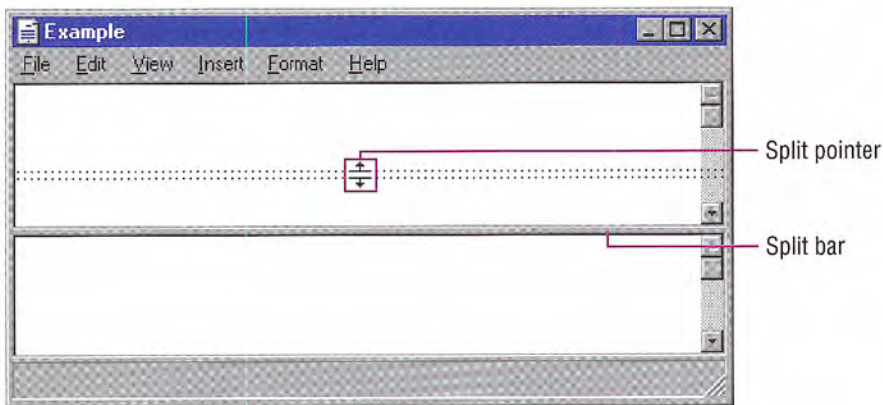


**Figure 6.11** Split box location

The user splits a window by dragging the split box to the desired position. When the user positions the hot spot of the pointer over a split box, change the pointer's image to provide feedback and help the user target the split box. While the user drags the split box, move a representation of the split box and split bar with the pointer, as shown in Figure 6.12.



At the end of the drag, display a visual separator, called the *split bar*, that extends from one side of the window to the other, defining the edge between the resulting panes, as shown in Figure 6.12. Base the size for the split bar to be, at a minimum, the current setting for the size of window sizing borders. This allows you to appropriately adjust when a user adjusts size borders. If you display the split box after the split operation, place it adjacent to the split bar.



**Figure 6.12** Moving the split bar

You can support dragging the split bar (or split box) to the end of the scroll bar to close the split. Optionally, you can also support double-clicking (or, for pens, double-tapping) as a shortcut technique for splitting the window at some default location (for example, in the middle of the window or at the last split location) or for removing the split. This technique works best when the resulting window panes display peer views. It may not be appropriate when the design of the window requires that it always be displayed as split or for some types of specialized views.

To provide a keyboard interface for splitting the window, include a Split command on the window or view's menu. When the user chooses the Split command, split the window in the middle or in a context-defined location. Support arrow keys for moving the split box up or down; pressing the ENTER key sets the split at the current location. Pressing the ESC key cancels the split mode.

You can also use other commands to create a split window. For example, you can define specialized views that, when selected by the user, split a window to a fixed or variable set of panes. Similarly, you can enable the user to remove the split of a window by closing a view pane or by selecting another view command.

When the user splits a window, add scroll bars if the resulting panes require scrolling. In addition, you may need to scroll the information in panes so that the split bar does not obscure the content over which it appears. Use a single scroll bar, at the appropriate side of the window, for a set of panes that scroll together. However, if the panes each require independent scrolling, a scroll bar should appear in each pane for that purpose. For example, the vertical scroll bars of a set of panes in a horizontally split window are typically controlled separately.

When you use split window panes to provide separate views, independently maintain each pane's view properties, such as view type and selection state. Display only the selection in the active pane. However, if the selection state is shared across the panes, display a selection in all panes and support selection adjustment across panes.

When a window is closed, save the window's split state (that is, the number of splits, the place where they appear, the scrolled position in each split, and its selection state) as part of the view state information for that window so that it can be restored the next time the window is opened.





# Menus, Controls, and Toolbars



Microsoft Windows provides a number of interactive components that make it easy to provide interfaces to carry out commands and specify values. These components also provide a consistent structure and set of interface conventions. This chapter describes the interactive elements of menus, controls, and toolbars, and how to use them.

## Menus

*Menus* list commands available to the user. By making commands visible, menus leverage user recognition rather than depending on user recollection of command names and syntax.

There are several types of menus, including drop-down menus, pop-up menus, and cascading menus. The following sections cover these menus in more detail.

### The Menu Bar and Drop-down Menus

A *menu bar*, one of the most common forms of a menu interface, is a special area displayed across the top of a window directly below the title bar (as shown in Figure 7.1). A menu bar includes a set of entries called *menu titles*. Each menu title provides access to a *drop-down menu* composed of a collection of *menu items*, or choices.



**Figure 7.1 A menu bar**

The content of the menu bar and its drop-down menus are determined by the functionality of your application and the context of a user's interaction. You can also optionally provide user configuration of the menu structure, including hiding the menu bar. If you provide this kind of option, supplement the interface with other components such as pop-up menus, handles, and toolbars, so that a user can access the functionality typically provided by the menu bar.

When displayed, a drop-down menu appears as a panel with its menu items arranged in a column. While the system supports multiple columns for a drop-down menu, avoid this form of presentation because it adds complexity to browsing and interaction of the menu.

## Drop-down Menu Interaction

When the user chooses a menu title, it displays its associated drop-down menu. To display a drop-down menu with the mouse, the user points to the menu title and presses or clicks mouse button 1. This action highlights the menu title and opens the menu. Tapping the menu title with a pen has the same effect as clicking the mouse.

If the user opens a menu by pressing the mouse button while the pointer is over the menu title, the user can drag the pointer over menu items in the drop-down menu. As the user drags, each menu item is highlighted, tracking the pointer as it moves through the menu. Releasing the mouse button with the pointer over a menu item chooses the command associated with that menu item and the system removes the drop-down menu. If the user moves the pointer off the menu and then releases the mouse button, the menu is "canceled" and the drop-down menu is removed. However, if the user moves the pointer back onto the menu before releasing the mouse button, the tracking resumes and the user can still select a menu item.

If the user opens a menu by clicking on the menu title, the menu title is highlighted and the drop-down menu remains displayed until the user clicks the mouse again. Clicking a menu item in the drop-down menu or dragging over and releasing the mouse button on a menu item chooses the command associated with the menu item and re



moves the drop-down menu. When the system displays a drop-down menu, clicking its associated menu title again cancels the menu and removes the drop-down. Clicking another menu title also results in canceling any displayed drop-down menu, and displays the menu associated with that menu title.

The keyboard interface for drop-down menus uses the ALT key to activate the menu bar. When the user presses an alphanumeric key while holding the ALT key, or after the ALT key is released, the system displays the drop-down menu whose access key for the menu title matches the alphanumeric key (matching is not case sensitive). Pressing a subsequent alphanumeric key chooses the menu item in the drop-down menu with the matching access character.

The user can also use arrow keys to access drop-down menus from the keyboard. When the user presses the ALT key, but has not yet selected a drop-down menu, LEFT ARROW and RIGHT ARROW keys highlight the previous or next menu title, respectively. At the end of the menu bar, pressing another arrow key in the corresponding direction wraps the highlight around to the other end of the menu bar. Pressing the ENTER key displays the drop-down menu associated with the selected menu title. If a drop-down menu is already displayed on that menu bar, then pressing LEFT ARROW or RIGHT ARROW navigates the highlight to the next drop-down menu in that direction, unless the drop-down menu displays its content in multiple columns, in which case the arrow keys move the highlight to the next column in that direction, and then to the next drop-down menu.

Pressing UP ARROW or DOWN ARROW in the menu bar also displays a drop-down menu if none is currently open. In an open drop-down menu, pressing these keys moves to the next menu item in that direction, wrapping the highlight around at the top or bottom. If the drop-down menu has multiple columns, then pressing the arrow keys first wraps the highlight around to the next column.

The user can cancel a drop-down menu by pressing the ALT key whenever the menu bar is active. This not only closes the drop-down menu, it also deactivates the menu bar. Pressing the ESC key also cancels a drop-down menu. However, the ESC key cancels only the current menu level. For example, if a drop-down menu is open, pressing ESC closes the drop-down menu, but leaves its menu title highlighted. Pressing ESC a second time unhighlights the menu title and deactivates the menu bar, returning input focus to the content information in the window.



You can assign shortcut keys to commands in drop-down menus. When the user presses a shortcut key associated with a command in the menu, the command is carried out immediately. Optionally, you can also highlight its menu title, but do not display the drop-down.

## Common Drop-down Menu

This section describes the conventions for drop-down menus commonly used in applications. While these menus are not required for all applications, apply these guidelines when including these menus in your software's interface.

### The File Menu

The File menu provides an interface for the primary operations that apply to a file. Your application should include commands such as Open, Save, Send To, and Print. These commands are often also included on the pop-up menu of the icon displayed in the title bar of the window.

If your application supports an Exit command, place this command at the bottom of the File menu preceded by a menu separator. When the user chooses the Exit command, close any open windows and files, and stop any further processing. If the object remains active even when its window is closed — for example, like a folder or printer — then include the Close command instead of Exit.



For more information about the commands in the pop-up menu for a title bar icon, see the section, "Icon Pop-up Menus," later in this chapter.

### The Edit Menu

Include general purpose editing commands on the Edit menu. These commands include the Cut, Copy, and Paste transfer commands, OLE object commands, and the following commands (if they are supported).



For more information about transfer commands, see Chapter 5, "General Interaction Techniques."

<b>Command</b>	<b>Function</b>
Undo	Reverses last action.
Repeat	Repeats last action.
Find and Replace	Searches for and substitutes text.
Delete	Removes the current selection.
Duplicate	Creates a copy of the current selection.


Include these commands on this menu and on the pop-up menu of the selected object.

## The View Menu

Put commands on the View menu that change the user's view of data in the window. Include commands on this menu that affect the view and not the data itself — for example, Zoom or Outline. Also include commands for controlling the display of particular interface elements in the view — for example, Show Ruler. Also place these commands on the pop-up menu of the window or pane.


## The Window Menu

Use the Window menu in multiple document interface-style (MDI) applications for commands associated with managing the windows within an MDI workspace. Also include these commands on the pop-up menu of the parent MDI window.

 For more information about the design of MDI software, see Chapter 9, "Window Management."

## The Help Menu

Use the Help menu for commands that provide access to Help information. Include a Help Topics command; this command provides access to the Help Topics browser, which displays topics included in your application's Help file. Alternatively, you can provide individual commands that access specific pages of the Help Topics browser, such as Contents, Index, and Find Topic. You can also include other user assistance commands on this drop-down menu.

 For more information about the Help Topics browser and support for user assistance, see Chapter 12, "User Assistance."

If you provide access to copyright and version information for your application, include an *About application name* command on this menu. When the user chooses this command, display a window containing the application's name, version number, copyright information, and any other informational properties related to the application. Display this information in a dialog box or alternatively as a copyright page of the property sheet of the application's main executable file. Do not use an ellipsis at the end of this command, because the resulting window does not require the user to provide any further parameters.

## Pop-up Menus

Even if you include a menu bar in your software's interface, you should also support pop-up menus. Pop-up menus provide an efficient way for the user to access the operations of objects, as shown in Figure 7.2. Because pop-up menus are displayed at the pointer's current location, they eliminate the need for the user to move the pointer to the menu bar or a toolbar. In addition, because you populate pop-up menus with commands specific to the object or its immediate context, they reduce the number of commands the user must browse through. Pop-up menus also minimize screen clutter because they are displayed only upon demand and do not require dedicated screen space.

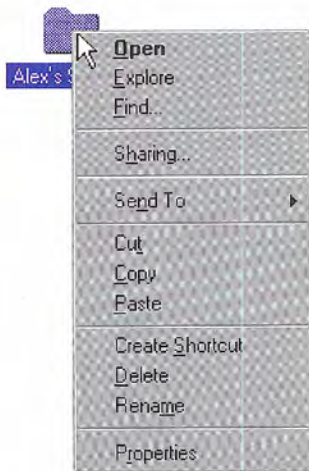


Figure 7.2 A pop-up menu



While a pop-up menu looks similar to a drop-down menu, a pop-up menu should only contain commands that apply to the selected object or objects and its context, rather than commands grouped by function. For example, a pop-up menu for a text selection can include commands for moving and copying the text and access to the font properties of the text and the paragraph properties of which the selection is a part. However, keep the size of the pop-up menu as small as possible by limiting the items on the menu to common, frequent actions. It is better to include a single Properties command and allow the user to navigate among properties in the resulting property sheet than to list individual properties in the pop-up menu.

The container or the composition of which a selection is a part typically supplies the pop-up menu for the selection. Similarly, the commands included on a pop-up menu may not always be supplied by the object itself, but rather be a combination of those commands provided by the object and by its current container. For example, the pop-up menu for a file in a folder includes transfer commands. In this case, the folder (container) supplies the commands, not the files. Pop-up menus for OLE objects follow these same conventions.

Avoid using a pop-up menu as the exclusive means to a particular operation. At the same time, the items in a pop-up menu need not be limited only to commands that are provided in drop-down menus.

When ordering the commands in a pop-up menu, use the following guidelines:

- Place the object's primary commands first (for example, commands such as Open, Play, and Print), transfer commands, other commands supported by the object (whether provided by the object or by its context), and the What's This? command (when supported).
- Order the transfer commands as Cut, Copy, Paste, and other specialized Paste commands.
- Place the Properties command, when present, as the last command on the menu.



For more information about transfer commands and the Properties command, see Chapter 5, "General Interaction Techniques." For more information about the What's This? command, see Chapter 12, "User Assistance."

## Pop-up Menu Interaction

With a mouse, the user displays a pop-up menu by clicking an object with button 2. The down transition of the mouse button selects the object. Upon the up transition, display the menu to the right and below the hot spot of the pointer adjusted to avoid the menu being clipped by the edge of the screen.

If the pointer is over an existing selection when the user invokes a pop-up menu, display the menu that applies to that selection. If the menu is outside a selection but within the same selection scope, then establish a new selection (usually resetting the current selection in that scope) at the button down point and display the menu for the new selection. Dismiss the pop-up menu when the user clicks outside the menu with button 1 or if the user presses the ESC key.

You can support pop-up menus for objects that are implicitly selected or cannot be directly selected, such as scroll bars or items in a status bar. When providing pop-up menus for objects such as controls, include commands for the object that the control represents, rather than for the control itself. For example, a scroll bar represents a navigational view of a document, so commands might include Beginning of Document, End of Document, Next Page, and Previous Page. But when a control represents itself as an object, as in a forms layout or window design environment, you can include commands that apply to the control—for example, commands to move or copy the control.

The pen interface uses an action handle in pen-enabled controls to access the pop-up menu for the selection. Tapping the action handle displays the pop-up menu, as shown in Figure 7.3.

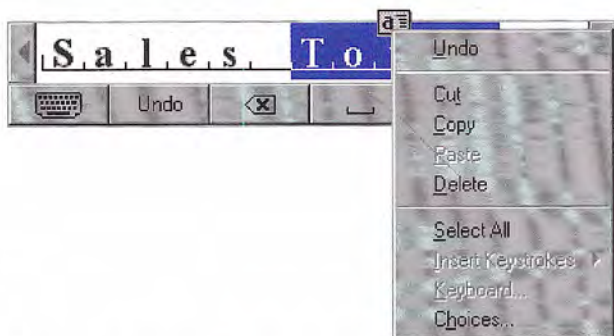


Figure 7.3 Using an action handle to provide pen access to pop-up menus