LIANT

Retooling Enterprise Systems

Cobol-WOW ™
Windows Object Workshop

User's Guide

Version 3.1 for Windows®

This manual is a user's guide for Cobol-WOW, Liant Software Corporation's graphical user interface development tool for RM/COBOL. It is assumed that the reader is familiar with programming concepts and with the COBOL language in general.

The information contained herein applies to systems running under Microsoft Windows 95, Microsoft Windows 98, Microsoft Windows NT, and Microsoft Windows 2000 operating systems.

The information in this document is subject to change without prior notice. Liant Software Corporation assumes no responsibility for any errors that may appear in this document. Liant reserves the right to make improvements and/or changes in the products and programs described in this manual at any time without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted.

Table of Contents

Preface

Cobol-WOW is Liant Software Corporation's powerful, yet easy-to-use, graphical user interface development tool for 32-bit Windows (9*x*/Me/NT/TS/2000). Cobol-WOW enables the COBOL developer to create true Windows applications with Windows event-handling and COBOL business logic, while leveraging the wealth of existing Windows and user-interface component technologies.

Cobol-WOW also enables COBOL developers of RM/Panels-based applications to create sophisticated Windows graphical user interfaces featuring many Windows controls. See Appendix D, *Using Cobol-WOW with RM/Panels*, for more information on using Cobol-WOW with RM/Panels.

What's New in Version 3.10

Version 3.10 of Cobol-WOW contains both enhancements and problem corrections to the previous release. Enhancements to Cobol-WOW v3.10 include the following:

- A new component, Cobol-WOW Thin Client, has been added that allows Cobol-WOW programs to be executed in a client/server architecture over a LAN or the Internet.

- Print support has been added to the Cobol-WOW Designer. It is now possible to print a form or print a packaging list of the application components, such as COBOL program, bitmaps, icons, animation files, ActiveX controls, and so on) that are required for distribution.

- Form and control properties can now be organized either alphabetically or by functional category.

- The static text box control can now have a transparent background.

- Version 3.10 allows you to enter a text string that you want to locate in your application code. Results are displayed in an area below the project tree and the desktop area of the Designer window.

Cobol-WOW v3.10 requires RM/COBOL 7.00.03 or higher.

Note For information on the significant enhancements in previous releases of RM/COBOL, see page **xxvii**.

Cobol-WOW Documentation

Liant now distributes the documentation for this product on the software distribution media (CD-ROM). This electronic documentation is formatted in Adobe Portable Document Format (PDF). There is one PDF file per manual, each with the extension .pdf. To view and print the PDF documentation requires using Adobe Acrobat Reader (version 4.0 or later). If needed, this software is available on the product CD and is also freely available for most operating systems at www.adobe.com.

The PDF file for this product is the *Cobol-WOW User's Guide*. On a Windows system, this PDF file is located in the directory $x:\docs$, where $x:$ is your CD-ROM drive. (Access to this documentation will also be provided by a shortcut icon entry to the Programs folder during installation of the Cobol-WOW application.) In addition, Cobol-WOW also comes with extensive online Help files, which are designed to help you learn and use the product. You can access Help through the Help menu, or by pressing F1 or clicking the What's This? toolbar button to get context-sensitive help for particular parts of the Cobol-WOW Designer programming interface. Tooltips also are available on controls, toolbar buttons, menu commands, and other screen elements during design time. The Help files include the following:

- *Designer*, a fundamental guide to the elements of the Cobol-WOW Designer interface.

- *Functions and Messages*, a comprehensive reference documenting the ActiveX, Cobol-WOW, and Windows API functions and messages used in Cobol-WOW.

The *Cobol-WOW User's Guide* and online Help files are designed to address the majority of users' questions,. If, however, these sources do not answer your question or problem, please check the following:

- README files included with the Cobol-WOW media

- Liant web site at http://www.liant.com/

Note The Cobol-WOW documentation set assumes you know how to use a mouse, open a menu, and choose menu and dialog box options. To review these techniques, consult the documentation for Windows.

How This Manual is Organized

This manual, the *Cobol-WOW User's Guide*, gives detailed information about all aspects of Cobol-WOW and is arranged as follows:

Chapter 1—Installing Cobol-WOW**.** This chapter provides the system requirements and installation instructions for Cobol-WOW.

Chapter 2—Tutorial**.** This chapter guides you through the building of a sample program that represents a fundamental building block typical of commercial applications.

Chapter 3—Introducing Cobol-WOW**.** This chapter describes the Cobol-WOW components, provides an overview of the development process, and discusses the Windows graphical operating environment as it relates to Cobol-WOW.

Chapter 4—Developing with Cobol-WOW**.** This chapter is designed to provide essential background information to help you understand what you are doing and why. Projects, event-driven applications, issues in data entry programs, and working with menus are all discussed.

Chapter 5—Debugging**.** This chapter discusses three different approaches to debugging a Windows-based application created with Cobol-WOW: using COBOL DISPLAY statements, using the RM/COBOL Interactive Debugger, and using CodeWatch, Liant's standalone source-level debugger.

Appendix A—Setting Properties and Events for Intrinsic Controls and Forms**.** This appendix describes the properties and events of each of the intrinsic controls (or default controls) used in the Cobol-WOW programming system as well as the properties and events for forms.

Appendix B—Working with ActiveX Controls**.** This appendix describes special considerations for using ActiveX controls with Cobol-WOW.

Appendix C—Understanding the Application Architecture**.** This appendix covers the overall design and structure of the Cobol-WOW programming system.

Appendix D—Using Cobol-WOW with RM/Panels**.** This appendix describes how to use Cobol-WOW with RM/Panels to enhance existing panel libraries and also discusses how to migrate panel libraries to Cobol-WOW forms.

Appendix E—Using Cobol-WOW Thin Client**.** This appendix describes how to install and use Cobol-WOW Thin Client, which allows the user interface to exist on the Windows client machine and the COBOL program (data processing) to occur on the server.

Symbols and Conventions

The following typographic conventions are used throughout this manual to help you understand the text material and to define syntax:

1. Words in all capital letters indicate COBOL reserved words, such as statements, phrases, and clauses; acronyms; configuration keywords; and environment variables.

2. Names of properties, events, and special objects appear with initial letter capitalized. Key names, such as Enter, also have the initial letter capitalized.

3. A plus sign (+) between key names indicates a combination of keys. For example, Ctrl+X means to press and hold down the Ctrl key while pressing the X key. Then release both keys.

4. Text displayed in a monospace font indicates user input or system output (according to context). This type style sets off sample command lines, program code and file listing examples, and sample sessions.

5. Bold, lowercase letters represent filenames, directory names, and programs. Note that Cobol-WOW accepts uppercase and lowercase filenames. Words you are instructed to type appear in bold. Bold type style is also used for emphasis, generally in some types of lists.

6. Italic text identifies the titles of other books, and it is also used occasionally for emphasis. In syntax, italic text denotes a placeholder or variable for information you supply, as described below.

7. The symbols found in the syntax charts are used as follows:

    italicized words indicate items for which you substitute a specific value.

    UPPERCASE WORDS indicate items that you enter exactly as shown (although not necessarily in uppercase).

    ... indicates indefinite repetition of the last item.

    | separates alternatives (an either/or choice).

    [ ] enclose optional items or parameters.

    { } enclose a set of alternatives, one of which is required.

    {| |} surround a set of unique alternatives, one or more of which is required, but each alternative may be specified only once; when multiple alternatives are specified, they may be specified in any order.

8. All punctuation must appear exactly as shown.

9. The term "window" refers to a delineated area of the screen, normally smaller than the full screen. The term "Windows" refers to the Microsoft Windows operating system.

Registration

Please take a moment to fill out and mail (or fax) the registration card you received with Cobol-WOW. You can also complete this process by registering your Liant product online at: **http://www.liant.com/**.

Registering your product entitles you to the following benefits:

- **Customer support.** Free 30-day telephone support, including direct access to support personnel and 24-hour message service.

- **Special upgrades.** Free media updates and upgrades within 60 days of purchase.

- **Product information.** Notification of upgrades or revisions to Cobol-WOW as soon as they are released.

You can also receive up-to-date information about Liant and all its products via our web site. Check back often for updated content.

Technical Support

Liant Software Corporation is dedicated to helping you achieve the highest possible performance from the Liant family of products. The technical support staff is committed to providing you prompt and professional service when you have problems or questions about your Liant products.

Technical support services are subject to Liant's prices, terms, and conditions in place at the time the service is requested.

While it is not possible to maintain and support specific releases of all software indefinitely, we offer priority support for the most current release of each product. For customers who elect not to upgrade to the most current release of the products, support is provided on a limited basis, as time and resources allow.

Support Guidelines

When you need assistance, you can expedite your call by having the following information available for the technical support representative:

1. Company name, support contract, partner, ADR, or distributor number.

2. Liant product serial number (found on the media label, registration card, or product banner message).

3. Liant product version number.

4. Operating system and version number.

5. Hardware, related equipment, and terminal type.

6. Exact message appearing on screen.

7. Concise explanation of the problem and process involved when the problem occurred.

Test Cases

You may be asked for an example (test case) that demonstrates the problem. Please remember the following guidelines when submitting a test case:

- The smaller the test case is, the faster we will be able to isolate the cause of the problem.

- Do not send full applications.

- Reduce the test case to the smallest possible combination of components required to reproduce the problem.

- If you have very large data files, write a small program to read in your current data files and to create new data files with as few records as necessary to reproduce the problem.

- Test the test case before sending it to us to ensure that you have included all the necessary components to recompile and run the test case. You may need to include an RM/COBOL configuration file.

When submitting your test case, please include the following items:

1. **README text file that explains the problems.** This file must include information regarding the hardware, operating system, versions of all relevant software (including the operating system and all Liant products). It must also include step-by-step instructions to reproduce the behavior.

2. **Program source files.** We require source for any program that is called during the course of the test case. Be sure to include any copy files necessary for recompilation.

3. **Data files required by the programs.** These files should be as small as possible to reproduce the problem described in the test case.

Enhancements

The following section summarizes the major enhancements available in earlier versions of Cobol-WOW.

Version 3.0

Version 3.0 of Cobol-WOW has been significantly enhanced to provide new functionality and improved reliability. With this version, Liant Software Corporation assumes all responsibilities for the product, including future enhancement and support. Prior versions were developed and maintained by England Technical Services, Inc.

Cobol-WOW 3.0, which requires RM/COBOL 7.00.03 or higher, includes improved ActiveX support, a new "look-and-feel" for the Cobol-WOW Designer, up-to-date documentation, and many defect corrections.

Note Cobol-WOW 3.0 is project-based. If you have a form-based application created with an earlier version of Cobol-WOW, you must create a project and add the form files in the existing application to it. For more information, see "Cobol-WOW Projects" on page **57**.

Chapter 1: Installing Cobol-WOW

This chapter provides the system requirements and installation instructions for Cobol-WOW.

System Requirements

Your computer configuration is the assembled set of hardware and software that makes up your system. The minimum hardware and software requirements your computer system needs to run Cobol-WOW successfully are shown in the following sections.

Required Hardware

Cobol-WOW requires the following minimum configuration:

- A PC with an Intel 80486 or higher processor; Pentium-class or higher recommended.

- Eight megabytes of available memory (RAM); 16 or more megabytes recommended.

- Ten megabytes of disk space.

- A standard VGA monitor (640 pixels by 480 lines); an Enhanced Super VGA monitor (800 pixels by 600 lines) recommended.

- A hard disk drive and a CD-ROM drive.

- A mouse.

Required Software

- Microsoft Windows 95, 98, Me, NT 4.0, NT 4.0 Terminal Server (TS), or 2000.

- RM/COBOL version 7.00.03 or higher for 32-bit Windows.

System Installation

This section describes the basic installation of the Cobol-WOW development system. The Setup program provided by Cobol-WOW performs all tasks for installing the Cobol-WOW components.

To install Cobol-WOW:

1. Start Windows.

    Note It is recommended that you close all other applications before proceeding with the installation.

2. Insert the Cobol-WOW CD in the CD-ROM drive.

    If the installation program does not start automatically, click **Start**, then click **Run**. In the Run dialog box, in **Open**, type **d: autorun**, where d is the drive letter of the CD-ROM drive.

3. Follow the instructions presented by the installation program.

Locating Required Tools

The options on the Tools page of the Preferences dialog box determine where Cobol-WOW locates the RM/COBOL compiler and runtime systems, the CodeWatch debugger, and the Cobol-WOW panel runtime (**wowpanrt.dll**).

To establish the location of the RM/COBOL compiler and runtime system, the CodeWatch debugger, and the Cobol-WOW panel runtime:

1. On the **Options** menu in the Cobol-WOW Designer window, click **Edit Preferences**.

2. In the Preferences dialog box, click the **Tools** tab to open the corresponding page of preferences.

3. Set the preferences to your specifications.

Customizing the Initialization File (cblwow.ini)

An initialization file with the name **cblwow.ini** is a text file that is used to contain configuration information for the Cobol-WOW Designer and runtime. The cblwow.ini file must be located in the Windows directory. It is processed whenever the Cobol-WOW Designer or runtime are executed.

If you add ActiveX controls to your Toolbox (see Appendix B, *Working with ActiveX Controls*), the controls will be recorded in the cblwow.ini file.

While it is possible to customize the cblwow.ini file to suit your needs, editing should be restricted to those features described specifically in the Cobol-WOW *Designer* online Help file as changes to the cblwow.ini file. For example, you can define default settings for various runtime activities by manually adding a [WOWRT] section (described below) to the initialization file. Furthermore, you can also add an [RMPanelsFunctionKeys] section to the .ini file so that the function key information (see page **267**) can be loaded by the Cobol-WOW runtime to run a Cobol-WOW-enhanced RM/Panels application.

Other entries in the cblwow.ini file are reserved for use by Cobol-WOW.

[WOWRT] Section

It is possible to customize the initialization file (**cblwow.ini**) in order to define default settings for various runtime activities. This can be done by using either of the following methods:

- Changing various options on the Runtime page of the Preferences dialog box in the Cobol-WOW Designer (on the Options menu, click Edit Preferences, and then click the Runtime tab).

- Or, manually adding a [WOWRT] section to the cblwow.ini file:

```
[WOWRT]
DevelopmentMode=True
RightJustifyMenus=True
FilterEvents=False
UseOEMConversion=True
ValidNumericChars=0123456789$,.+-
EditChar=x
DecChar=y
```

Setting DevelopmentMode to True enables messages that aid in debugging a Cobol-WOW application at runtime.

Setting RightJustifyMenus to True causes the MFT_RIGHTJUSTIFY style to be added at runtime to a menu when it is created.

Setting FilterEvents to False causes event filtering not to be performed for a form and its controls at runtime, overriding the default AllowEventFilter property setting (True).

Setting UseOEMConversion to True causes COBOL data to be converted from OEM to ANSI (and vice versa) when the WOWGETPROP and WOWSETPROP functions are used at runtime. This option is useful if you need to support extended characters in your Cobol-WOW application. Extended characters are those having an ASCII value greater than 128, such as 'Ü', and '¢'.

ValidNumericChars causes the WOWGETNUM function to return a value of 1 at runtime if an invalid numeric character is contained in the field. By default, valid characters are considered to be the digits 0 through 9, the dollar sign ($), the plus (+) and minus (-) characters, and the edit (comma) and decimal (period) characters. You can change this default to a character set of your choosing. All characters listed will be considered valid numeric characters. A limit of 80 characters is supported.

EditChar and DecChar allow you to override at runtime the default decimal (.) and edit (,) characters for currency editing.

Chapter 2: Tutorial

In this chapter, you will build a sample program that represents a fundamental building block typical of commercial applications. The first exercises demonstrate how you use the Cobol-WOW Designer to begin a project and create two forms that work together to build a full-fledged file maintenance application program. The last section of this chapter presents techniques on how to attach code to events associated with these forms and controls. These instructions contain most of the tasks you will need to perform when writing your own applications.

Using the File Maintenance Program

If you do not want to perform the exercises in this file maintenance program, but would like to run and examine the program in the Cobol-WOW Designer, a completed application exists.

To open and run this sample program:

1. In the Cobol-WOW Designer window, do one of the following:

    - Click the **Open Project** toolbar button.

    - On the **Project** menu, click **Open**.

2. In the Open Project dialog box, select the name of the project you want from the **File Name** list (in this case, select **project.wpj** in the Samples folder). Click **Open**. The forms for the selected project will appear in the Cobol-WOW Designer window.

3. Click the **Execute Project** toolbar button or click **Run** on the **Project** menu to run the program. Shortcut key: F7

To examine specific code that demonstrates ways to manipulate the controls in this sample program:

1. With the form open in the Cobol-WOW Designer window, select a control.

2. Open the Event-Handling Code dialog box by double-clicking the left mouse button. Alternatively, click **View Code** on the **Control** menu or click the **View Control Code** toolbar button.

    Any code associated with the control will be displayed in the Event-Handling Code dialog box.

Using Projects

By using a project, the Cobol-WOW Designer allows you to do your complete development in an integrated, visual framework. The default extension for Cobol-WOW project filenames is **.wpj**. The **.wpj** file is a text file that contains project configuration information and a list of the forms included in the project. The form files that are contained in a project are also known as members. The default extension for a form file is **.wow**. (For more information on forms and projects, see pages **46** and **57**, respectively.)

Create a New Project

To begin designing the customer file maintenance application, create a new project:

1. In the Cobol-WOW Designer window, click the **New Project** toolbar button or click **New** on the **Project** menu. The Save Project dialog box opens.

   Note Only one project can be opened at one time. If a project is open when you choose this command, you are prompted to close the active project and save any changes before a new project is created.
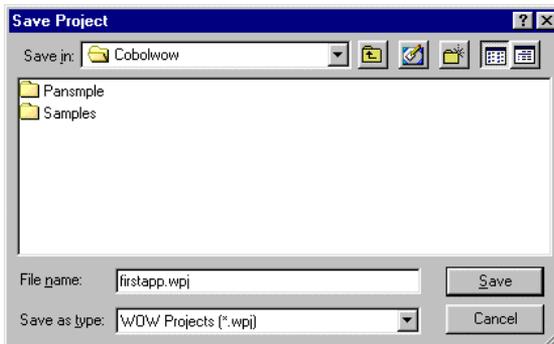
Save Project Dialog Box

2. In the **Save in** box, select the directory (folder) into which you wish to save the project.

3. In the **File Name** box, enter **firstapp.wpj** as the project name.

4. After entering the project name, click **Save** to create the project.

   The Edit Project dialog opens. The Edit Project dialog is used to add or remove forms from the project. Since you have not created any forms for your project yet, there are no files (or members) to add.
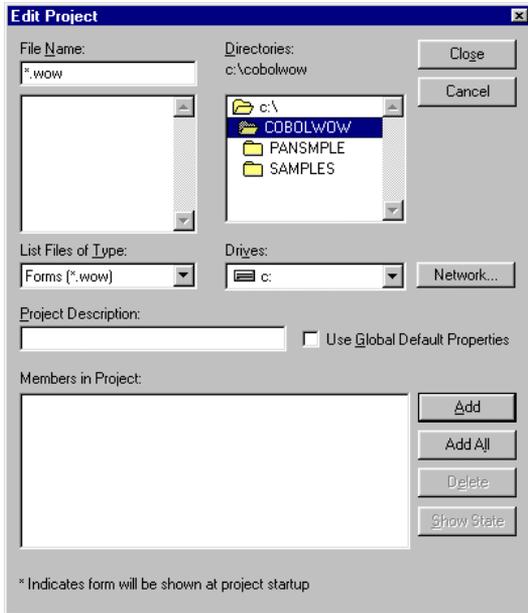
Edit Project Dialog Box

5. Click **Close** to close the dialog box. Now you are ready to design your forms for the project.

Designing Forms

Designing forms is as simple as arranging objects in a window. In Cobol-WOW, the objects you manipulate are called controls, and the window is the form. Forms are the foundation for all your Cobol-WOW applications.

An application program usually contains multiple forms. You develop an application by customizing a form and then adding and customizing additional forms for other parts of the interface. To customize a form, you add controls, set their properties, and create menus to provide user control over the application at runtime.

In this project, you will design the first form, FIRSTAPP, as the application's main window. It will contain a menu, a list box displaying customer names, and Add, Change, and Delete command buttons. You will design the second form, CUSTINFO, to pop up when the user chooses the Add or Change command buttons on the FIRSTAPP form to allow editing of the customer information.

The FIRSTAPP form is illustrated in the following figure.
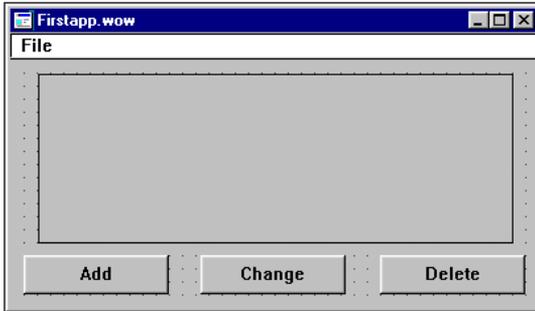
FIRSTAPP Form

Create the FIRSTAPP Form

To create a new, blank form in the Cobol-WOW Designer window:

- Click the **New Form** toolbar button, or click **New** on either the **Form** menu or the **File** menu.

Each new form is created with the same default, initial properties. Before you add any controls to this form, you need to change some of these initial properties.

Setting Form Properties

Cobol-WOW makes it easy for you to set properties, the attributes that define how forms and controls are displayed and how they function in the running application. To change some of the form's initial properties during design time, open the Properties dialog box for the associated form by doing one of the following:

- On the **View** menu, click **Properties**.
- Click the **View Properties** toolbar button.
- Right-click anywhere within the form.
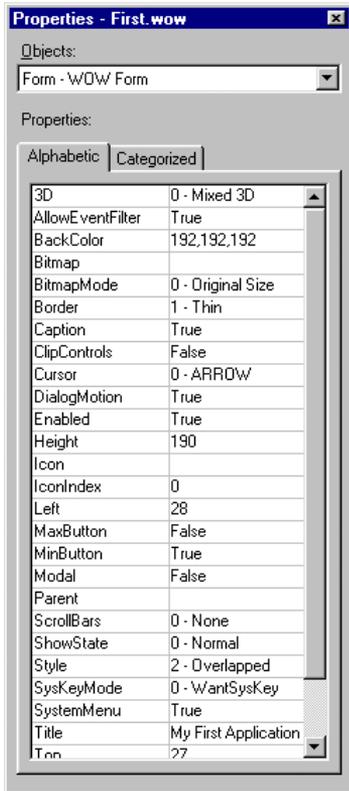
Properties Dialog Box

Although most of the default settings will be appropriate for the FIRSTAPP form, you do
need to modify the Border, Caption, MinButton, Style, SystemMenu, and Title properties.
Let's look at each of these properties, beginning with Style.

Note The Properties area of this dialog box is divided into two columns. The left column
displays all of the properties associated with the selected form. The right column displays
the current value (or setting) for each property. You can view the properties either
alphabetically or by functional category. Simply click the appropriate tab.

Style Property

Cobol-WOW provides three different types of forms from which you can choose:
Overlapped, Child, and Popup. The differences among these types are not dramatic. In
Windows application design, an overlapped window is a top-level window with a border,
a client area, and a title bar. Top-level windows are windows that are not children of
other windows and are generally appropriate for the main window of an application.

A pop-up window does not have a parent by default (although a parent can be set for it); a pop-up window can be drawn anywhere on the screen. The main differences between a pop-up window and an overlapped window is that a pop-up window can be displayed outside the border of its parent window (if it has one). A child window means that the form has a parent. The parent-child relationship determines where a window can be drawn on the screen. A child window can be drawn only within its parent's client area, and is destroyed along with its parent.

The Overlapped option is a convenient combination of a number of other property settings. Choosing Overlapped as the Style is equivalent to setting the Caption, MaxButton, MinButton, and SystemMenu properties all to True, and setting Border to Thick. These settings would create a form as an overlapped window with a title, System-menu box, Minimize button, and Maximize button in the title bar. The form would also have a sizable frame.

While you want an overlapped window style for your application's main window, you do not necessarily want exactly the type configured with the Overlapped option of the Style property. Since your window will contain a fixed layout of controls, sizing should not be allowed. (Sizing is determined by the MaxButton and Border properties.)

To enable the specific properties that you require for the FIRSTAPP form:

1. In the **Properties** list on the Properties dialog box, click **Style**.

2. Select the **Overlapped** option.

    Setting Style to Overlapped removes the thick frame and Maximize button settings. By removing the thick frame and Maximize button, you ensure that the window is always the size you specify at design time.

Note Because the Cobol-WOW Designer is a standard Windows multiple document interface (MDI) application, your form does not change in the Cobol-WOW Designer window to reflect these property changes. Windows is very particular about the nature of MDI windows. The multiple document interface feature is a means for applications to simultaneously open and display two or more files in the same application. MDI window styles defined in the Cobol-WOW Designer cannot change to reflect the current property settings at design time. For example, if a form did not have a sizable border, there would be no way of sizing it in the Cobol-WOW Designer. These settings will be reflected at runtime when the window is created.

Title Property

The Title property is a descriptive label and can contain any alphanumeric character of your keyboard, including spaces. This property setting can be used only for top-level windows.

To change the title of your form, type **My First Application** in the value column of the Title property. Remember, until you compile and run the project, the Title bar will continue to display the default form filename in the Cobol-WOW Designer, not the text you entered for the Title property.

Border, Caption, MinButton, and SystemMenu Properties

Set the remaining properties, as follows:

Property	Setting
Border	1 - Thin
Caption	True
MinButton	True
SystemMenu	True

Moving and Sizing a Form

The property settings for your FIRSTAPP form are now complete. Close the Properties dialog box.

Before you start adding controls to your form, try sizing and moving it.

To move the form, place the pointer on the form's title and press the left mouse button. Then drag the mouse to reposition the form.

To size the form, place the pointer on the form's border. The pointer changes shape to indicate the directions in which the form can be resized. Press the left mouse button and drag the border to resize the form. As you add controls to the form, you may need to resize it again.

Note The size and position the form has in the Cobol-WOW Designer window are the default size and position for the form at runtime.

Add Controls to the FIRSTAPP Form

This exercise explains how to add a pulldown (also known as drop-down) menu control with the Menu Editor dialog box, a list box control displaying customer names, and Add, Change, and Delete command button controls to the FIRSTAPP form.

Creating a Menu

This section discusses the basics of creating a menu at design time in the FIRSTAPP form, your application's main window. The Cobol-WOW Designer provides an easy method to add a menu control to a form at design time by using the Menu Editor dialog box. First, however, it is important to understand the implementation of menus in Windows programming and the relationship between a menu and a form.

The menu object is an independent object from the form. When a pulldown menu is added to a form, the menu maintains its own distinct identity. Because the menu is created as a unique object (behind the scenes) and then attached to the form, it can also be detached from the form, and then a different menu can be attached. Windows provides a wealth of API functions to create, modify, and destroy menus. These functions give you complete control and flexibility with menus at runtime. (For more information, see the *Functions and Messages* online Help file.)

It may also be helpful to learn some terminology associated with menus. A pulldown menu is represented by a menu title (for example, File, Edit, Help) that appears in the menu bar of an application window (form). The menu bar is the horizontal list of titles immediately below the title bar on the form. When you choose a pulldown menu title, a menu containing a list of menu items drops down. Menu items can include commands, separator bars, and submenu titles. Each menu title and menu item the user sees corresponds to a menu control you define in the Menu Editor dialog box.

At menu design time, this terminology is not pertinent. When you start to modify menus at runtime, however, you need to understand the distinctions. Each pulldown menu and menu item has a distinct handle. In order to change the menus at runtime, you must use the proper handle.

Main applications windows are the main windows that remain displayed throughout an application. Most application windows have a menu bar and pulldown menus, which provide a set of logically grouped commands to users.

The menu for the FIRSTAPP form will contain only one top-level option displayed in the menu bar: a File menu. The File menu will have one menu item, Quit.

To create a menu control:

1. From the **View** menu, click **Menu Editor** or click the **View Menu Editor** toolbar button.
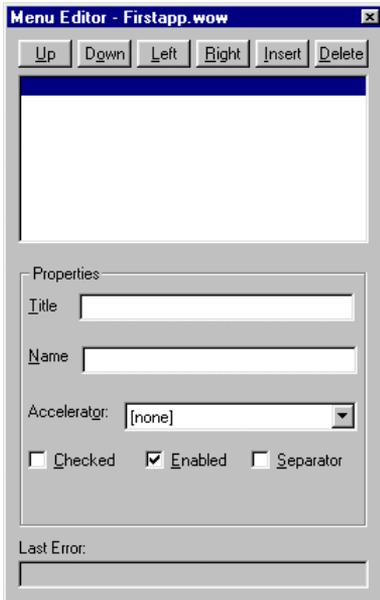
2. In the Menu Editor dialog box, the first position in the **Menu Control** list box is blank and highlighted. Select the position where you want to create the top-level menu. (For this exercise, if the first line in this list is not highlighted, click the first line.)

3. In the **Properties** area, click the **Title** box.

4. In the **Title** box, type **File**.

   This value is the text for the first menu title that you want to appear on the form's menu bar. The menu title, File, is displayed in the top line of the Menu Control list box. It is also immediately visible on the form as changes made in this dialog box are automatically applied to a form.

   As you give the menu item a Title property value of File, notice that Cobol-WOW automatically assigns the Name property of the menu control a default value. Under Cobol-WOW's default configuration, the naming convention used for menu controls includes the prefix M- followed by the text entered in the Title box. In this case, the value M-FILE is displayed in the Name box. Cobol-WOW adds this menu name to the form's declaration, and the menu name appears in the Events/Code Sections list

of the menu Event-Handling Code dialog box. It is the name Cobol-WOW uses to reference the menu control in the generated code.

Note You can configure the way in which the menu names are generated by changing the options in the Code page of the Preferences dialog box. To open the Preferences dialog box, click **Edit Preferences** on the **Options** menu, and then click the **Code** tab.

5. Click the second line in the **Menu Control** list box to add a menu item (option) to the menu created in the previous step. (You can also press Enter from the first line in the Menu Control area to advance to the second line.)

6. Click the **Title** box.

7. In the **Title** box, type **Quit**.

   This value is the text for the first menu item that you want to appear in the File menu. The menu item, Quit, is displayed in the second line of the Menu Control list box. The name value M-QUIT is automatically displayed in the Name box.

8. To indent the menu item, click the **Right** command button at the top of the dialog box.

   Indenting Quit to the right under File makes it a menu item in a menu list of the File menu. The name value M-QUIT is automatically changed in the Name box to display M-F-QUIT. If you did not indent this control, the menu bar would have two top-level options, File and Quit.

9. In the upper-right corner of the title bar, click ⊠ to close the Menu Editor dialog box.

The Menu Editor dialog box also allows you to apply several formats to the different menu items. For instance, you can add an accelerator (or shortcut key) to the menu item (for example, Ctrl+N) by clicking on the **Accelerator** list box to display a list of key combinations that may be used to assign accelerator keys to access menu commands. In order for the accelerator to appear next to the menu item, you must type the accelerator key sequence selected from the list box following the text you entered in the Title box. It is also possible to place a tab character in the menu item to align the text that describes the accelerator key sequence. A tab character is added by placing the sequence \t in the menu item. For example, if you select Ctrl+N in the Accelerator list box, you will type New\tCtrl+N in the Title box. Note that this should be done only for pulldown menu items, not for top-level titles. Only one tab character should be added to a menu item.

Additionally, you can display a check mark on the menu item at design time by clicking on the **Checked** check box. Check marks are commonly used to indicate an on/off condition. Choosing the menu command alternately adds and removes the check mark. If

a menu command is active, a check mark will appear next to the menu item. For more information about displaying a check mark on a menu item at runtime, see page **77**.

You may also enable the menu item at design time by clicking on the **Enabled** check box. When the Enabled property is checked (the default), the menu responds to user actions. If a menu control is disabled, it appears grayed or dimmed. For information about enabling a menu control at runtime, see page **78**.

Whenever a menu contains a set of related menu items, you can insert a horizontal line, known as a separator bar, between the menu items by clicking on the **Separator** check box. This provides a visual break in the list of items.

Creating a List Box

The primary control of your FIRSTAPP form is a list box that will be used to display customers. Within a form, a list box presents a list of available choices for the user. It is a good design choice whenever you have a large number of fixed choices; for example, a list of all the files in a directory or a list of customer accounts.

To create a list box:

1. On the **View** menu, click **Toolbox** or click the **View Toolbox** toolbar button.

2. In the **Toolbox**, click the **List Box** control.

3. Move the cross-hair pointer to the upper-left corner of the form and click the left mouse button.

4. Drag the pointer down towards the lower-right corner of the form to draw the box.

5. Release the mouse button. An empty list box is displayed.

A list box has its own set of unique properties and events. With two exceptions, the default set of properties will work satisfactorily for this application. Open the Properties dialog box for the list box and set the following properties:

Property	Setting
Name	CUST-LB
UseTabStops	True

The Name property setting is referenced by your application code and must conform to COBOL data name restrictions. Cobol-WOW automatically forces the entry to uppercase.

The UseTabStops setting will help you align the information in the list box. (For information on how to add items to a list box at a specific position, see page .)

By default, the list box, like all controls, is created with the same background color as the form. You may wish to set a different background color for the list box control. If so, modify the BackColor property in the Properties dialog box or click Background Color on the Control menu to select a different background color for the list box.

Creating the Command Buttons

The easiest way to allow the user to interact with an application is to provide a command button to click. Like menus, command buttons issue commands. You generally design pulldown menus to contain commands that fall into logical groups. If you have only a few commands and enough space on the form, you can create command buttons instead of menus.

In the following exercises, you will add three command buttons to the FIRSTAPP form The first command button, Add, will initiate and carry out the add operation in order to add customers to the list box you just created.

To create the Add button:

1. On the **View** menu, click **Toolbox** or click the **View Toolbox** toolbar button.

2. In the **Toolbox**, click the **Command Button** control.

3. Move the cross-hair pointer to the lower-left corner of the form and click the left mouse button.

4. Drag the pointer down to the right to outline an area for the button.

5. Release the mouse button. A command button appears with default text.

The command button, like the list box, is created with a default set of properties. For this application, you need to modify only a few of these properties. Open the Properties dialog box for the Add command button control and set the following properties:

Property	Setting
Caption	Add
Name	ADD-CMD

The Add value is the caption that identifies the command button on the form. The text Add replaces the default text of "Command Button."

ADD-CMD is the name you will use to refer to the command button control in code.

Note The TabIndex property is used to specify the order in which the controls are sequenced. This sequencing is used when a user presses Tab (to move forward) or Shift+Tab (to move backward). A TabIndex value of 2 causes the Add command button to be second in the entry order of the controls on this form. (The list box, because it was the first control created, has a TabIndex value of 1.)

You also need to create Change and Delete command buttons similar to the Add button you have already created. Create these two buttons on the same line and to the right of the Add button on the form.

Set the following properties for the Change button:

Property	Setting
Caption	Change
Name	CHANGE-CMD

Set the following properties for the Delete button:

Property	Setting
Caption	Delete
Name	DELETE-CMD

Arrange Controls on the FIRSTAPP Form

Once you have added all the controls, you can refine the appearance of your form by resizing the form, if necessary. You can also arrange or align the controls for a more balanced layout, and define the tab order and/or z-order for the controls.

Note You must select a control before you can manipulate it on a form.

Selecting

To select a single control, single-click the control on the form with the left mouse button.

You can also select more than one (or multiple) controls, which provides a convenient method for moving or aligning a group of controls at the same time. To select multiple controls, first hold down the **Shift** key, and then click the controls, one at a time. You can also select more than one control by positioning the pointer beside (not on) one of the controls you want to select. Then, drag diagonally through all the controls you want to select. While you drag, Cobol-WOW draws a rectangle around the controls. When you release the mouse button, all the controls in the rectangle are selected.

To select all the controls in a form, you can choose **Select All** from the **Edit** menu.

Note When more than one control is selected in the form, a single Properties dialog box displays all the properties that are shared among the selected controls. (The Objects list area in the Properties dialog will indicate "Form – Multiple Objects Selected".) This is true even when the value for the shared property differs among the selected controls. In this case, the property value column is empty (or blank). However, when you click on the value area, the value of the first control selected is displayed. When you change any of the shared properties in the Properties dialog, the property value changes to the new value in all the selected controls. There is one notable exception to this: when you select multiple controls in a form, their Name property no longer appears in the Properties list area even though they all have a Name property. This is because you cannot assign the same value for the Name property to more than one control in a form.

Resizing

When a control is selected on a form, small squares called sizing handles appear on the perimeter of the control. To resize a control, select the control on the form, then drag one of the sizing handles to the desired size:

- Drag the handles on the top and bottom to size the control vertically.

- Drag the handles on the left and right sides to size the control horizontally.

- Drag the handles in the corners to size the control both vertically and horizontally.

When you release the mouse button, the control is redrawn in the new size.

You can also size the controls on a form by using the **Size** command from the **Control** menu or by using toolbar buttons on the Sizing toolbar.

Moving

To move a control, select it, then click the body of the control (being careful not to select the sizing handles), and then drag the control to the desired location. If you wish to move the controls in a position not allowed by the grid, turn off the **Show Grid** and **Snap to Grid** commands in the **Form** menu.

Aligning and Spacing

You can align, center, and distribute controls by using the **Align**, **Center**, and **Space** commands from the **Control** menu or by using toolbar buttons on the Aligning, Centering, and Spacing toolbars. If you wish to place your controls in a position not allowed by the grid, turn off the **Show Grid** and **Snap to Grid** commands in the **Form** menu. (You can also change the position of controls with certain properties in the Properties dialog box.)

When aligning a group of controls, the <u>first</u> control you select is used as a guide to which the other controls are aligned. To align the three command buttons along the bottom of the form:

1. Click the **Add** button and move it to the position you want in the lower-left corner of the form.

2. Select the other two command buttons by pressing the **Shift** key as you click each control.

3. On the **Control** menu, click **Align** and point to the submenu.

4. On the submenu, click **Bottoms**.

    The three command buttons align horizontally relative to the bottom edge of the first control selected, in this case the Add command button.

At this point, you probably need to modify the spacing among the three command buttons in even intervals along the bottom of the form.

To space the controls evenly across the form:

1. On the **Options** menu, click **Edit Preferences**. The Preferences dialog box opens.

2. Click the **Alignment** tab.

3. In the **Spacing** area, click **Space Controls** and then click **OK**. The dialog box closes.

    With this option, selected controls are distributed equally between the left and right edges of the controls and the leftmost and rightmost boundaries of the form. (The Space Centers options distribute the spacing between the center points of each control and the leftmost and rightmost boundaries of the form.)

4. Select the three command buttons.

5. On the **Control** menu, click **Space**, and then point to **Horizontal** on the submenu.

    The spacing among the three command buttons is evenly distributed between the edges of the controls and the boundaries of the form.

You can continue to choose or modify alignment options as long as the controls remain selected.

Specifying Tab Order

Tab order is the order in which the Tab key moves the input focus from one control to the next. (Pressing Shift+Tab moves the focus in the reverse order.) When a control has focus, it can receive input from the user through the mouse or keyboard. The tab order is

initially set by Cobol-WOW and corresponds to the order in which controls are added to the form.

Note To enable the Tab key to shift focus to a control on a form in a running application, the TabStop property (see page **174**) must be set to True.

You can determine the tab order for your program by choosing the **Tab Order** command on the Control menu in the Designer window, or by using the shortcut key, Ctrl+T. A number in blue in the upper-left corner of each control shows its place in the current tab order. (Note that the Toolbox closes temporarily when the Designer is in tab order mode.) To change the tab order, double-click the control you want to be first in the tab order, and then single-click on the rest of the controls in the order in which you want them to be selected in the form when a user presses the Tab key. To exit tab order mode, click the mouse anywhere in the form or click the Tab Order command again. (See page **69** for more information handling tab order at runtime.)

Alternatively, you can change the tab order for selected controls by changing the TabIndex property (see page **173**) for the control in the Properties dialog box. Note, however, that there is a limitation when using this method since you can only change the value of the TabIndex property to a value that has not already been set. Although some controls (animation, bitmap, progress bar, static text box, status bar, tab, timer, toolbar, and all shapes) cannot accept mouse or keyboard focus, they still will have a valid tab order. When the user presses Tab, the focus skips over such a control and goes to the next control in the tab order.

By default, the first control added to the form in this exercise has a TabIndex property of 1, the second has a TabIndex of 2, and so on. In this example, the list box, which has a TabIndex value of 1, would have the focus at runtime. Pressing Tab would move the focus to the Add command button, then to the Change button, and finally to the Delete button. For this FIRSTAPP form, you do not need to change the tab order of the controls as long as you added them in the order specified. Examine the Properties dialog box for the list box and each of the command buttons to verify that they are in the correct tab order sequence.

Specifying Z-order

The z-order indicates the control stacking order, that is, the order in which controls are created. The controls with the smaller numbers are stacked "behind" the controls with the larger numbers. The controls with the larger numbers are "on top" of all the other controls. Cobol-WOW initially sets the z-order for each control to correspond to the order in which they are added to the form.

You can determine the z-order for your program by choosing the **Z-Order** command on the Control menu in the Designer window, or by using the shortcut key, Ctrl+R. A

number in red in the upper-left corner of each control shows its place in the current z-order. To change the z-order, select the Z-Order command (a check will appear to the left of the command), then click the control you want to be first in the stacking order. Its number will change. Then, continue to click controls until they are in the desired order. To exit z-order mode, click the mouse anywhere in the form or click the Z-Order command again. (Note that the Toolbox temporarily closes when the Designer is in z-order mode.)

You also can change the z-order for selected controls by changing the value of the ZOrder property in the Properties dialog box. (The first control in the z-order should have the ZOrder value of 1.)

For this FIRSTAPP form, you do not need to change the z-order of the controls as long as you added them in the order specified. Examine the Properties dialog box for the list box and each of the command buttons to verify that they are in the correct z-order sequence.

Save the FIRSTAPP Form

Once you have completed your form design layout and property settings, you are ready to save your work.

To save the FIRSTAPP form:

1. On either the **File** or **Form** menu, click **Save** or click the **Save Form and Generate Code** toolbar button.
2. In File Save As dialog box, type a filename in the **File Name** box. In this case, type **firstapp.wow**. Be sure to save the form in the appropriate working directory.

    It is generally a good idea to use the form title for the filename (or a shortened version of it), although you can use any name you want.

    Note All filenames must conform to MS-DOS naming conventions.

3. Click **Save**. A message box asks, "Do you want to add Firstapp to the project?"

4. Click **Yes**. Your form is now saved in the current project file, **firstapp.wpj**.

Note If you were designing several new forms at once in a project, you can choose the **Save All** command from either the **File** or the **Form** menu to save all the open forms. The save process varies depending upon whether the form has previously been saved. If you have not previously saved the form(s), Cobol-WOW displays the File Save As dialog box. This dialog box prompts you to supply a name for each open form that has been created. If you have previously saved the form, all open forms that reside in the directory created to store them, are saved to disk if they have been modified.

Name Property

The name you entered to save the form is very significant. First, this name is the name shown in your code — it is used to identify the form to the underlying program. Any time you need to reference the form from application code you will use this name. Form files are, by default, given an extension of **.wow** when they are saved. This filename extension represents a Cobol-WOW resource file. You may change the extension to whatever you desire, but it will be easier to locate your forms if you use this extension.

At the same time, the name you entered when you save a form is used to identify and generate two copy files. The first copy file, with the filename extension **.wpr**, contains the COBOL logic necessary for the form. The name of your file also identifies and generates the COBOL Working Storage copy file needed for the form. This copy file has a filename extension of **.wws**.

Create the CUSTINFO Form

You are ready to build the second form of your file maintenance application, the CUSTINFO form. By following the previous techniques, you now have enough experience with the Cobol-WOW Designer to design a form without step-by-step instructions. This exercise presents only the specifications for the second form of your application. Be sure to adhere strictly to these specifications, however, or you will have difficulty writing and attaching code to the events associated with these forms and controls.

The form you will create is a small form named CUSTINFO, which will pop up on top of the FIRSTAPP form (although not hiding it completely) when a user chooses the Add or Change command button. If you leave the FIRSTAPP form open in the Cobol-WOW Designer while you design the CUSTINFO form, you can see how they will be displayed together.

When completed, the form will appear as illustrated in the following figure.
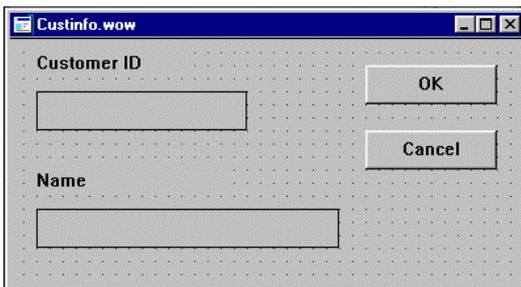
CUSTINFO Form

The CUSTINFO form will have two data entry controls, one for the customer identification number and one for customer name. These controls will both have captions (or labels) to identify them. The form will also contain OK and Cancel command buttons to confirm or cancel the action being executed.

Setting Form Properties

To create the CUSTOINFO form:

1. On either the **File** or **Form** menu, click **New** or click the **New Form** toolbar button.

2. Set the following properties for the form (use the default settings for all other properties):

Property	Setting
Border	1 - Thin
Caption	True
Style	1 - Popup
Modal	True
Title	Customer Information

Note Changing the Modal value to True causes the form to disable all other forms belonging to that application. When a form runs modally, the user must explicitly close it before working in another running form. (In contrast, when a running form is modeless, it remains onscreen while the user works in another form, for example, the application main form.) When a user needs to enter information into a form or otherwise complete its use prior to accessing other forms, create a modal form. This change becomes visible at runtime, not design time.

Add Controls to the CUSTINFO Form

Next, add the following controls to the form by using the following tools in the Toolbox:

Tool	Control
ab	Use the Edit Box tool to create two edit box controls: customer identification and customer name.
Aα	Use the Static Text tool to create label controls for the two edit box controls: customer identification and customer name.
▢	Use the Command Button tool to create two command button controls: OK and Cancel.

After you add the controls to the form, set the following properties for these controls (use the default settings for the remaining properties):

Control	Property	Setting
Edit Box	Border	True
	MaxChars	6
	Name	CUST-ID
	TabIndex	1
	Text	Delete default text and leave the text box blank.
Edit Box	Border	True
(Customer Name)	MaxChars	20
	Name	CUST-NAME
	TabIndex	2
	Text	Delete default text and leave the text box blank.
Static Text	Caption	Customer ID
Static Text	Caption	Name
Command Button	Caption	OK
	Name	OK-CMD
	TabIndex	3
Command Button	Caption	Cancel
	Name	CANCEL-CMD
	TabIndex	4

Save the CUSTINFO Form

Save the form with the filename, **custinfo.wow**, and add it to the project when prompted to do so.. (Use the method described on page **21** for saving the FIRSTAPP form.) You also can further edit this member of the project.

1. From the **Project** menu, click **Edit**. The Edit Project dialog box opens.

2. In the Members in Project area, select **custinfo.wow**.

    Notice that an asterisk is appended to the filename. The asterisk indicates that the form will be shown at project startup.

3. Click the **Show State** button to remove the asterisk and ensure that the CUSTINFO form does not appear at project startup.

Note The default behavior of Cobol-WOW is to display all forms in a project when the project is opened. If, after clicking the Show State button to remove the asterisk from the CUSTINFO form, you close and reopen firstapp.wpj, Cobol-WOW will revert to its default behavior. To change the default, click **Options** from the **Project** menu. In the At Project Open area of the Project Options dialog box, click **Open showstate forms at project open** to ensure that the CUSTINFO form does not appear at project startup.

4. Click **Close** to exit the Edit Project dialog box.

Writing Code

So far, this chapter has discussed design guidelines: how to create forms within a project, add controls to a form, set form and control properties, and save forms. This section takes you through six steps to build on this design of a customer file maintenance program. Before you can use the forms that you have designed, you must attach procedure code (logic) and functions to the events associated with these forms and controls (the window objects).

You must also provide access to the COBOL data files required by the application. Copy files provide this file access.

1. In the Cobol-WOW Designer, open the **FIRSTAPP** form.

2. From the **Project** menu, click **View Code**. Alternatively, click the **View Project Code** toolbar button. The project Event-Handling Code dialog box opens.

3. In the **Events/Code Sections** list, click **Declaratives**.

4. Press Tab once in order to start typing in column 8.

5. Add the following COPY statement: COPY "firstapp.dcl".

6. Continue adding COPY statements to code sections as follows:

Code Section	COPY Statement
File Section	COPY "firstapp.fd".
File-Control	COPY "firstapp.sl".
Procedure Division	COPY "firstapp.prc".
Working-Storage Section	COPY "firstapp.ws".

Each of the following exercises provides you with a working program. These techniques will provide an excellent foundation on how to build Windows-based applications with

Cobol-WOW. The methods learned here will transfer easily to the other types of programs you develop.

In each of these exercises, you will gain additional familiarity with the Windows API functions, their use, and the characteristics of window objects and different types of controls. Windows presents a number of controls you can use to develop Windows-based applications, including buttons, text boxes, list boxes, and many others. Each of these objects is created with a specific "personality" and capabilities. For example, buttons and text boxes provide the most fundamental methods of receiving input and displaying output in your programs. To effectively develop Windows-based software, you must learn the nature of these objects and how to work with them.

Step 1 — Exiting Methods

In the following exercise, you will provide the user with ways to exit the application.

Writing Code for Menu Controls

The customer file maintenance program's main form is the **firstapp.wow** file. It contains a System menu with a Close command, and a File menu that includes a Quit command. Whenever the user chooses either of these menu items in a running application, either by clicking the menu command or by using its accelerator or shortcut keys, the form is destroyed and the program is exited.

By default, Cobol-WOW automatically attaches code to destroy the window (form) and exit the program when a user chooses the Close command on the System menu. In this exercise, you will attach this same logic to the Quit menu item on the FIRSTAPP form, and then compile and run the program.

To attach this code to the Quit event on the File menu and save the form:

1. On the **Form** menu, click **View Code** and then click the submenu option, **Menus**. Alternatively, click the **View Form Menu Code** toolbar button.

2. In the Event-Handling Code dialog box, click **M-F-QUIT** in the Events/Code Sections list.

   There is only one event that can occur when a user chooses a menu item: the Click event. Therefore, when you select the M-F-QUIT object, the Click event is selected automatically in the Events/Code Sections list.

3. Move the cursor to the **Code Entry** area and press **Tab** twice in order to start typing in column 12, since this is the body of a procedure. The Line, Col identifier in the lower-right corner of the Event-Handling Code dialog box should read 1,12.

4. Type the following line of code:

```
SET WOW-QUIT TO TRUE.
```

5. Click **Close** to close the Event-Handling Code dialog box.

6. Click the **Save Form and Generate Code** toolbar button to save the changes made to the form.

Compiling and Running Program

To compile the COBOL source code for the project's main program file (**firstapp.cbl**, which was created when the project was first saved):

1. On the **Project** menu, click **Build** or click the **Build Project** toolbar button. Shortcut key: F7.

2. When the compiler has finished, close the COBOL compiler window. You are now ready to run the project.

To run the COBOL object code for the project's main program file (**firstapp.cob**, which was created when the project was first compiled):

1. On the **Project** menu, click **Run** or click the **Execute Project** toolbar button. Shortcut key: F5.

2. While the program is running, you can test the event-handling code that you added to the Close and Quit menu controls. Click **Close** from the System menu or double-click the System menu. The program exits to the Cobol-WOW Designer.

3. Run the program again, and this time click **Quit** from the **File** menu. Once again, the program returns to the Cobol-WOW Designer.

Controlling the COBOL Main Window

When you run the program, you may or may not see the standard COBOL main window displayed in addition to your Cobol-WOW form. By setting the Main Window Type property in the Windows Registry to a value of SHOW or HIDDEN, you can specify whether or not the COBOL main window is displayed. For more information, refer to the "Setting Properties" section of the chapter entitled "Installation and System Considerations for Microsoft Windows" in the *RM/COBOL User's Guide*.

The RM/COBOL Configuration utility (**rmconfig.exe**) also may be used to specify property values for the Main Window Type property to determine whether the COBOL main window is shown or hidden. You can also call a COBOL subprogram, C$SHOW, to dynamically hide and show the standard COBOL main window at runtime.

Step 2 — Loading the List Box

List boxes present a list of choices to the user. By default, the choices are displayed vertically in a single column. In this exercise, you will load your customer list into the list box control created previously on the FIRSTAPP form.

There are a number of places where you could load this information. At first, it may appear most likely to add the data to the skeleton program after the statement that creates the form. However, a better place to initialize the list box control is in response to the Windows message when the list box control is created. When a window object is created, Windows sends a message to the window object saying, "You are being created." Responding to this message is the appropriate place to initialize any and all controls on a form, including the list box. It is important that Windows-based applications be as event-driven as possible in order to make the program more maintainable. Internal program architecture is more likely to change than the Windows messaging system.

This tutorial so far has discussed how to set properties for forms and controls. Setting properties, however, is only one component of code development in Cobol-WOW. For some events or activities (loading a list box, for example), special functions and messages are used.

Using the WOWADDITEM Function

Loading the list box involves reading the customer file from start to finish and individually adding each customer to the list box. The easiest way to add an item to a list box is with the WOWADDITEM function.

The syntax of the function is as follows:

```
CALL WOWADDITEM USING WIN-RETURN WND-H NEWITEM
```

The WIN-RETURN parameter specifies the index of where the entry is added to the list box. This index is not used in this example. WNDORACTIVEX-H specifies the handle of the list box to which the entry should be added. NewItem specifies an alphanumeric field containing the text to add to the list box.

Although at first glance this function appears straightforward, it deserves closer examination. By default, a list box redisplays its contents every time an entry is added, which would, in this case, cause a distracting flicker on the screen. You can, however, tell the list box not to redisplay its contents during the loading operation by sending the message WM-SETREDRAW. WM-SETREDRAW works with all window objects (forms and controls), not just list boxes.

The syntax of the message appears as follows:

```
CALL SENDMESSAGE USING WIN-RETURN CUST-LB-H WM-SETREDRAW
    WIN-FALSE.
```

The WIN-RETURN parameter is not relevant in this context. CUST-LB-H specifies the handle of the list box for which to suppress redrawing. WM-SETREDRAW specifies the message identifier (ID). WIN-FALSE specifies that redraw should be turned off.

This same message can then be used with WIN-TRUE as the last parameter in order to turn redrawing back on after you have finished loading the list box.

Creating Logic to Load the List Box

Now you know where to write your code and what messages you will be using. To add the logic to load the list box to the FIRSTAPP form:

1. In the Cobol-WOW Designer, open the **FIRSTAPP** form.

2. Do one of the following to open the Event-Handling Code dialog box:

   • Select the form and double-click the left mouse button.

   • Click the **View Form Code** toolbar button.

3. In the **Events/Code Sections** list, click the **Create** event.

4. Move the cursor to the **Code Entry** area and press **Tab** twice in order to start typing in column 12, since this is the body of a procedure.

5. Type the following code:

```
PERFORM OPEN-CUST.
CALL SENDMESSAGE USING WIN-RETURN CUST-LB-H WM-SETREDRAW
    WIN-FALSE.
PERFORM READ-NEXT-CUST.
PERFORM UNTIL NOT VALID-CUST-IO
    PERFORM ADD-ENTRY-TO-LISTBOX
    PERFORM READ-NEXT-CUST
END-PERFORM.
CALL SENDMESSAGE USING WIN-RETURN CUST-LB-H WM-SETREDRAW
    WIN-TRUE.
```

This code uses two procedures, READ-NEXT-CUST and ADD-ENTRY-TO-LISTBOX. The READ-NEXT-CUST procedure, like all your file I/O logic, is supplied in the **firstapp.cbl** program. The ADD-ENTRY-TO-LISTBOX procedure, however, is not supplied in this manner.

The ADD-ENTRY-TO-LISTBOX procedure is not only used by the Create event logic, but also by other event-handling routines in the project. Since it is used by other routines,

you should create it in the Procedure Division of the PROJECT CODE SECTIONS object, rather than within this event procedure. While you could create it here and still perform it from other event procedures, it would be difficult to remember where it was defined. Placing shared procedures in the Procedure Division of the project Event-Handling Code dialog box eases maintenance. The next section describes how to do this.

Project Code Sections

When you create a project, Cobol-WOW allows you to specify the forms that are used in the project. Not only will Cobol-WOW keep track of the forms that are part of the project, it will create a skeleton COBOL program that creates, operates, and removes all of the forms. Better yet, you can edit any part of this COBOL program from inside the Cobol-WOW Designer. Because you are working with a project, you will select **View Code** from the **Project** menu. (Alternatively, you can click the **View Project Code** toolbar button.) Every code section of the COBOL program is listed in the Events/Code Sections list. You can copy in your file descriptions, declaratives, create additional Working Storage data items — in short, everything — from within the Cobol-WOW Designer.

Procedure Division Logic

The ADD-ENTRY-TO-LISTBOX procedure, which will add your customers to the list box, involves formatting the entry and sending the message to add it. You have already analyzed the message used to add the entry. There is, however, an interesting aspect to formatting the entry that should be discussed.

When you set the list box properties, you set UseTabStops to True because Windows supports fonts that are both fixed width and variable width. Variable-width fonts are more common under Windows, but they present some challenges to developers, especially when trying to align information.

With fixed-width fonts, the following entry would align properly by placing space characters between the number and the name:

```
0013422     John Smith
0015311     Harry Jones
```

With variable-width fonts, some characters are wider than others. Having the same number of characters in two lines does not necessarily cause the two entries to line up. In order to align the entries shown in this example, you must place a Tab character between the number and the name. In the case of a list box control, you must also tell the list box that you are using Tab characters by setting the UseTabStops property to True. Using this setting, the list box will interpret the Tab character as a positioning character and not as part of the text.

Note Be careful not to confuse this task with creating a multi-column list box control. In this case, you are separating two parts of a single entry with a Tab character so that it appears to be in two columns; it is still one entry. A multi-column list box would display as follows:

```
0013422 John Smith          014322 Frank Jones
0043255 Peter Parker        015322 Herb Black
```

To add the ADD-ENTRY-TO-LISTBOX procedure to the Procedure Division area:

1. From the **Project** menu, click **View Code** or click the **View Project Code** toolbar button.

2. The project Event-Handling Code dialog box is displayed.

3. In the **Events/Code Sections** list, click **Procedure Division**.

4. Move the cursor to the **Code Entry** area and press **Tab** once in order to start typing in column 8, since this is a complete procedure. The Line, Col identifier in the lower-right corner of the window should read 1,8.

5. Type the following code:

```
ADD-ENTRY-TO-LISTBOX.
    MOVE CUST-ID TO NEW-ENTRY (1:6).
    MOVE X"09" TO NEW-ENTRY (7:1).
    MOVE CUST-NAME TO NEW-ENTRY (8:40).
    CALL WOWADDITEM USING WIN-RETURN CUST-LB-H NEW-ENTRY.
```

The list box entry is formatted by moving the desired fields to an alphanumeric data item called NEW-ENTRY. The declaration variable NEW-ENTRY is described in the next section.

Working-Storage Section Logic

Because you are working with a project, you should declare variables in the Working-Storage Section of the project Event-Handling Code dialog box. This is the area where the variable NEW-ENTRY should be declared.

To declare NEW-ENTRY in the Working-Storage Section area:

1. In the **Events/Code Sections** list, click **Working-Storage Section**.

2. Move the cursor to the **Code Entry** area and press **Tab** once in order to start typing in column 8, since this is a variable declaration. The Line, Col identifier in the lower-right corner of the window should read 1,8.

3. Type the following code:

```
01 NEW-ENTRY        PIC X(50).
```

4. Click **Close** to close the project Event-Handling Code dialog box.

Saving, Generating, Compiling, and Running

To save the changes made to the list box, click the **Save Form and Generate Code** toolbar button.

The FIRSTAPP program can now be compiled. Then run the program to see the customers displayed in the list box.

Step 3 — Adding the Second Window

Right now your application displays only the FIRSTAPP form. When a user chooses the Add or Change options, you want the CUSTINFO form to appear for editing. To accomplish this, you will set an internal flag that indicates Add or Change mode, allowing the logic that pops up for the CUSTINFO form between the Add and Change operations to be shared. (The instructions for adding the logic to the Change command button are discussed on page **37**.)

You will pop up the CUSTINFO form by creating it and then remove it by destroying it, although showing and hiding the form would work equally well. When you pop up the CUSTINFO form, it will disable the FIRSTAPP form because the CUSTINFO form is modal. When a form runs as a modal window, the user must explicitly close it before accessing and working in another running form.

In this step, you will allow the user to remove the form only with the Cancel command button. To remove the CUSTINFO form, you will destroy it.

Adding Logic to the Add Command Button

To add the required logic to the Add command button:

1. In the Cobol-WOW Designer, open the **FIRSTAPP** form.

2. Do one of the following to open the Event-Handling Code dialog box for the control:

   • Select the Add command button control and double-click the left mouse button.

   • On the **Control** menu, click **View Code**.

   • Click the **View Control Code** toolbar button.

3. In the **Events/Code Sections** list, click the **Click** event.

4. Move the cursor to the **Code Entry** area and press **Tab** twice in order to start typing in column 12, since this is the body of a procedure.

5. Type the following code:

```
SET ADD-MODE TO TRUE.
PERFORM POPUP-RTN.
```

6. Click **Close** to close the control Event-Handling Code dialog box.

Declaring ADD-MODE

To declare the variable ADD-MODE in the Working-Storage Section:

1. From the **Project** menu, click **View Code** or click the **View Project Code** toolbar button. The project Event-Handling Code dialog box is displayed.

2. In the **Events/Code Sections** list, click **Working-Storage Section**. Existing code appears in the Code Entry area.

3. Move the cursor to the **Code Entry** area below the existing code. Press **Tab** once in order to start typing in column 8, since this is a variable declaration.

4. Type the following code:

```
01  PROGRAM-MODE         PIC X.
     88  ADD-MODE          VALUE "A".
     88  CHANGE-MODE       VALUE "C".
```


Declaring POPUP-RTN

Since the POPUP-RTN procedure will be used within both the Add and Change operations, create it in the Procedure Division of the PROJECT CODE SECTIONS object.

To add the POPUP-RTN procedure to the PROJECT CODE SECTIONS object:

1. In the **Events/Code Sections** list, click **Procedure Division**. Existing code appears in the Code Entry area.

2. Move the cursor to the **Code Entry** area below the existing code. Press **Tab** once in order to start typing in column 8, since this is a complete procedure.

3. Type the following code:

```
POPUP-RTN.
    PERFORM WP-CREATE-CUSTINFO.
```

4. Click **Close** to close the project Event-Handling Code dialog box.

Removing the CUSTINFO Window

To remove the CUSTINFO form and re-enable the FIRSTAPP form, you need to add logic to the Cancel command button on the CUSTINFO form:

1. In the Cobol-WOW Designer, open the **CUSTINFO** form.

2. From the **Form** menu, click **View Code** and then click **Form**, or click the **View Form Code** toolbar button. The form Event-Handling Code dialog box opens.

3. In the **Events/Code Sections** list, click the **Click** event.

4. Move the cursor to the **Code Entry** area and press **Tab** twice in order to start typing in column 12, since this is the body of a procedure.

5. Type the following code:

```
PERFORM WP-DESTROY-CUSTINFO.
```

The sequence of these two procedures is significant. Since FIRSTAPP is enabled before CUSTINFO is destroyed, FIRSTAPP becomes the active window when CUSTINFO is removed. If CUSTINFO were removed while FIRSTAPP was still disabled, some other enabled form would become the active window. Then, when FIRSTAPP was enabled, it would not automatically become active, and it would require an additional function call to make it the active window.

Saving, Compiling, and Running

Save, build, and run the project.

Step 4 — Adding Customers

When you pop up a CUSTINFO form, you need to be able to add customers. To do this, you add logic to the OK command button to save what you created in the CUSTINFO form.

When the OK command button is pressed, you want the user to retrieve the contents of the CUSTINFO edit fields, load the data record with them, and then write the new record. You also want to use this data to add a new entry to the list box. Then, you want to remove the pop-up window, just as you did with the Cancel command button. Notice that even though the FIRSTAPP form is disabled for user input, you can modify it (for example, add an entry to the list box).

Using the WOWGETPROP Function

Retrieving the contents of the CUSTINFO edit controls involves something new: retrieving the value of a property with the WOWGETPROP function. This function is very similar to the function used to set properties, WOWSETPROP.

The syntax of the WOWGETPROP function is as follows:

```
CALL WOWGETPROP USING WIN-RETURN CUST-ID-H "TEXT" CUST-ID.
```

The WIN-RETURN parameter is a status value for the function. CUST-ID-H specifies the handle of the form or control from which you want to retrieve a property value. "TEXT" is the name of the property to be retrieved. CUST-ID is the COBOL data item in which the property value should be stored.

This function can be used to retrieve any property for a form or control.

Adding Logic to the OK Command Button

To add the required logic to the OK command button:

1. In the Cobol-WOW Designer, open the **CUSTINFO** form.

2. Do one of the following to open the Event-Handling Code dialog box for the control:

    • Select the OK command button control and double-click the left mouse button.

    • On the **Control** menu, click **View Code**.

    • Click the **View Control Code** toolbar button.

3. In the **Events/Code Sections** list, click the **Click** event.

4. Move the cursor to the **Code Entry** area and press **Tab** twice to start typing in column 12, since this is the body of a procedure.

5. Type the following code:

```
PERFORM MOVE-DATA-TO-RECORD.
PERFORM WRITE-CUST.
PERFORM ADD-ENTRY-TO-LISTBOX.
PERFORM WP-DESTROY-CUSTINFO.
```

MOVE-DATA-TO-RECORD is a new procedure that you will create in a moment. WRITE-CUST is a file I/O procedure in the **firstapp.cbl** program. ADD-ENTRY-TO-LISTBOX, which formats an entry and adds it to the list box, is the procedure you created in Step 2 (see page **28**). WP-DESTROY-CUSTINFO is the same procedure you used with the Cancel command button to remove the CUSTINFO form and enable the FIRSTAPP form.

MOVE-DATA-TO-RECORD is used only by the OK command button event procedure. It is, however, such a discreet piece of functionality that good COBOL programming practice requires that you create it as a separate procedure. A procedure that is used by only one event-handling procedure should be created alongside that procedure. Create the MOVE-DATA-TO-RECORD procedure in the same Event-Handling Code dialog box (Click event for the OK-CMD object), but place it after the main body of the event-handling procedure. Since you are creating a procedure name, press **Tab** once to start typing in column 8. Type the following code:

```
MOVE-DATA-TO-RECORD.
    CALL WOWGETPROP USING WIN-RETURN CUST-ID-H "TEXT" CUST-ID.
    CALL WOWGETPROP USING WIN-RETURN CUST-NAME-H "TEXT" CUST-NAME.
```

Saving, Building, and Running

Save, build, and run the project.

When you run the project, press the Add command button to display the CUSTINFO form, enter the data, and press the OK command button. Your new entry should be displayed in the list box.

Step 5 — Changing Customers

Next, you need the ability to change customers, which requires adding logic to the Change command button, and modifying the POPUP-RTN and OK command button procedures.

When the Change button is pressed you want to make sure that a customer has been selected. If not, you do not want the CUSTINFO form to pop up. After determining that a customer is selected, you will read the customer, set CHANGE-MODE to TRUE, and then perform the POPUP-RTN. The POPUP-RTN procedure must be changed to load the current customer information into the CUSTINFO form after it is created. The OK command button logic must be changed to delete the customer from both the list box and the file before the new values are saved.

Working with List Box Selections

The presence or absence of a selection in a list box is determined with the CurSel property. This property is the 0 relative index of the currently selected item. If no item is selected, the property value is LB-ERR. The value of the selected list box item can be determined with the SelText property. If no item is selected the value is space.

An item can be deleted by using the WOWREMOVEITEM function as follows:

```
CALL WOWREMOVEITEM USING WIN-RETURN CUST-LB-H CUST-SEL-NUM.
```

The WIN-RETURN parameter is not relevant in this context. CUST-LB-H specifies the handle of the list box to be modified. CUST-SEL-NUM specifies the 0 relative index of the entry to be removed.

Adding Logic to the Change Command Button

To add the required logic to the Change command button:

1. In the Cobol-WOW Designer, open the **FIRSTAPP** form.

2. Do one of the following to open the Event-Handling Code dialog box for the control:

   - Select the **Change** command button control and double-click the left mouse button.

   - On the **Control** menu, click **View Code**.

   - Click the **View Control Code** toolbar button.

3. In the **Events/Code Sections** list, click the **Click** event.

4. Move the cursor to the **Code Entry** area and press **Tab** twice in order to start typing in column 12, since this is the body of a procedure.

5. Type the following code:

```
PERFORM CHECK-FOR-CUST-SELECTION.
IF NOT NO-CUST-SELECTED
    PERFORM READ-THIS-CUST
    SET CHANGE-MODE TO TRUE
    PERFORM POPUP-RTN
END-IF.
```

6. Click **Close** to close the control Event-Handling Code dialog box.

Adding Code to the Procedure Division

Because CHECK-FOR-CUST-SELECTION and READ-THIS-CUST are procedures that will also be used by the Delete operation, create these in the Procedure Division of the PROJECT CODE SECTIONS object. CHECK-FOR-CUST-SELECTION indicates whether or not a customer has been selected with the condition, NO-CUST-SELECTED.

To add the CHECK-FOR-CUST-SELECTION and READ-THIS-CUST procedures to the PROJECT CODE SECTIONS object:

1. From the **Project** menu, click **View Code** or click the **View Project Code** toolbar button. The project Event-Handling Code dialog box is displayed.

2. In the **Events/Code Sections** list, click **Procedure Division**. Existing code appears in the Code Entry area.

3. Move the cursor to the **Code Entry** area on the next line following the existing code. Press **Tab** once in order to start typing in column 8, since these are complete procedures.

4. Type the following code:

```
CHECK-FOR-CUST-SELECTION.
   CALL WOWGETPROP USING WIN-RETURN CUST-LB-H "CURSEL"
      CUST-SEL-NUM.

READ-THIS-CUST.
   CALL WOWGETPROP USING WIN-RETURN CUST-LB-H "SELTEXT"
      CUST-ID.
   PERFORM READ-CUST.
```

Both of these procedures use the CUST-SEL-NUM field. To declare this field in the Working-Storage Section:

1. In the **Events/Code Sections** list, click **Working-Storage Section**. Existing code appears in the Code Entry area.

2. Move the cursor to the **Code Entry** area on the next line following the existing code. Press **Tab** once in order to start typing in column 8, since this is a variable declaration.

3. Type the following code:

```
01 CUST-SEL-NUM          PIC S9(4).
     88 NO-CUST-SELECTED  VALUE -1.
```

Modifying the POPUP-RTN Procedure

To modify the POPUP-RTN procedure:

1. In the **Events/Code Sections** list, click **Procedure Division**.

2. Add three lines to the end of the POPUP-RTN procedure so that it appears as follows (the new lines of code appear as bold text):

```
POPUP-RTN.
    PERFORM WP-CREATE-CUSTINFO.
    IF CHANGE-MODE
       PERFORM MOVE-DATA-TO-WINDOW
    END-IF.
```

3. While still in the Procedure Division code section, create the MOVE-DATA-TO-WINDOW procedure following the existing code.

```
MOVE-DATA-TO-WINDOW.
    CALL WOWSETPROP USING WIN-RETURN CUST-ID-H "TEXT"
      CUST-ID.
    CALL WOWSETPROP USING WIN-RETURN CUST-NAME-H "TEXT"
      CUST-NAME.
```

Modifying the OK Command Button Procedure

To modify the OK command button procedure:

1. In the Cobol-WOW Designer, open the **CUSTINFO** form.

2. Do one of the following to open the Event-Handling Code dialog box for the control:

   - Select the OK command button control and double-click the left mouse button.

   - On the **Control** menu, click **View Code**.

   - Click the **View Control Code** toolbar button.

3. In the **Events/Code Sections** list, click the **Click** event.

4. Add four lines to the beginning of the procedure so that it appears as follows (the new lines of code appear as bold text):

```
IF CHANGE-MODE
    PERFORM DELETE-LISTBOX-ENTRY
    PERFORM DELETE-CUST
END-IF.
PERFORM MOVE-DATA-TO-RECORD.
PERFORM WRITE-CUST.
PERFORM ADD-ENTRY-TO-LISTBOX.
PERFORM WP-DESTROY-CUSTINFO.
```

Adding the Delete List Box Entry Procedure

The DELETE-LISTBOX-ENTRY procedure, used by both the Delete and Change operations, should be created in the Procedure Division of the PROJECT CODE SECTIONS object.

To create the DELETE-LISTBOX-ENTRY procedure:

1. From the **Project** menu, click **View Code** or click the **View Project Code** toolbar button. The project Event-Handling Code dialog box is displayed.

2. In the **Events/Code Sections** list, click **Procedure Division**. Existing code appears in the Code Entry area.

3. Move the cursor to the **Code Entry** area and press **Tab** once in order to start typing in column 8, since these are complete procedures.

4. Type the following code after the existing code:

```
DELETE-LISTBOX-ENTRY.
    CALL WOWREMOVEITEM USING WIN-RETURN CUST-LB-H
        CUST-SEL-NUM.
```

Saving, Building, and Running

Save, build, and run the project.

When you run the project, select a customer in the list box of the FIRSTAPP form and press the Change command button to display the CUSTINFO form. Then modify the data, and press the OK command button. The previous entry is deleted and the new one is displayed.

Step 6 — Deleting Customers

Finally, you need to add the ability to delete customers by creating logic to the Delete command button.

Like the Change command button, when the Delete command button is pressed, you want to be sure a customer is selected. When the customer is selected, a message box displays, asking the user to respond to the inquiry. When the user confirms the action, the customer is removed from the list box and the file.

Fortunately, all the required list box manipulation has already been created for the Change function. There is, however, one new technique that can be performed using the WOWMESSAGEBOX function.

WOWMESSAGEBOX Function

The WOWMESSAGEBOX function displays the confirmation message. The following syntax shows the logic required to use this function:

```
INITIALIZE MESSAGE-BOX-FLAGS.
SET MB-OKCANCEL MB-ICONQUESTION MB-TASKMODAL TO TRUE.
CALL WOWMESSAGEBOX USING WIN-RETURN 0
    "Are you sure you want to delete this customer"
    "Confirm deletion"
    MESSAGE-BOX-FLAGS.
```

The WOWMESSAGEBOX function has a large number of conditions associated with it. These conditions specify what buttons and icons should be placed in the message box and how the message box is displayed. These conditions are declared in MESSAGE-BOX-FLAGS.

MESSAGE-BOX-FLAGS must be initialized to clear all default conditions. Then the desired conditions are established again by setting their values to TRUE. In this example, the OK and Cancel command buttons are placed in the message box by setting MB-OKCANCEL to TRUE, a question mark icon is placed in the message box by setting MB-ICONQUESTION to TRUE, and the message box is displayed in task modal form by setting MB-TASKMODAL to TRUE. (Task modal means that the only item the user can access in this task is the WOWMESSAGEBOX. They could, however, switch to other tasks.)

The parameters for the WOWMESSAGEBOX function are described as follows:

- The WIN-RETURN parameter indicates what button was pressed to remove the dialog box.

- The 0 parameter is a parent for the message box; in this case, none.

- "Are you sure ..." is the text of the message to display.

- "Confirm deletion" is the title of the message box window.

- MESSAGE-BOX-FLAGS includes the conditions affecting the message box.

Adding Logic to the Delete Command Button

To add the required logic to the Delete command button:

1. In the Cobol-WOW Designer, open the **FIRSTAPP** form.

2. Do one of the following to open the Event-Handling Code dialog box for the control:

   - Select the Delete command button control and double-click the left mouse button.

   - On the **Control** menu, click **View Code**.

   - Click the **View Control Code** toolbar button.

3. In the **Events/Code Sections** list, click the **Click** event.

4. Move the cursor to the **Code Entry** area and press **Tab** twice in order to start typing in column 12, since this is the body of a procedure.

5. Type the following code:

```
PERFORM CHECK-FOR-CUST-SELECTION.
    IF NOT NO-CUST-SELECTED
       PERFORM CONFIRM-DELETE
    END-IF.
       IF WIN-RETURN = IDOK
           PERFORM READ-THIS-CUST
           PERFORM DELETE-LISTBOX-ENTRY
           PERFORM DELETE-CUST
       END-IF.
```

6. Type the following code beginning at column 8, since this is a complete procedure:

```
CONFIRM-DELETE.
    INITIALIZE MESSAGE-BOX-FLAGS.
    SET MB-OKCANCEL MB-ICONQUESTION MB-TASKMODAL TO TRUE.
    CALL WOWMESSAGEBOX USING WIN-RETURN 0
        "Are you sure you want to delete this customer?"
        "Confirm Deletion"
        MESSAGE-BOX-FLAGS.
```

Saving, Building, and Running

Save, build, and run the project.

When you run the project, press the Delete command button to display the message box, and then select OK to delete the currently selected customer.

Chapter 3: Introducing Cobol-WOW

Cobol-WOW (Windows Object Workshop) is a programming tool that allows you to design and to develop full-featured, event-driven Windows applications completely in COBOL.

This chapter includes the following topics:

- Cobol-WOW Components
- Cobol-WOW Development Process Overview
- Windows Graphical Operating Environment

Cobol-WOW Components

The Cobol-WOW development environment consists of three major components: a design facility, a runtime system, and the Cobol-WOW Thin Client program.

Cobol-WOW Designer

The Cobol-WOW Designer, **cblwow.exe,** is a standard Windows, multiple document interface (MDI) application that provides COBOL developers with the capability to define the visual interface elements of the application. The multiple document interface feature allows an application to manage multiple files within the single, parent (or application) window. In Cobol-WOW, this means you can open and edit multiple forms at one time in the Cobol-WOW Designer window. You can also copy information back and forth between forms, move and resize the forms, and so forth.

You first design the forms, populate those forms with controls, and adjust the properties of those forms and controls. Cobol-WOW collectively refers to these forms and controls as objects.

Then you use Cobol-WOW to write and manage the source code to support these objects. Every object has certain events to which it can respond. In the Designer, you write and attach COBOL event-handling logic to the specific Windows events and the code necessary to respond to user input events. Because Windows programming is event-driven, you write code to respond to user events rather than control the sequence of events.

Cobol-WOW Runtime System

The Cobol-WOW DLL, **wowrt.dll**, is a Windows dynamic link library (DLL) that manages Windows messages, provides runtime support for the forms and controls, and provides a COBOL interface to the Windows Application Programming Interface (API). When the Cobol-WOW runtime system is invoked by the Cobol-WOW Thin Client program (**tclient.exe**), it causes all Windows-based Cobol-WOW functions to be executed on the client workstation.

Note The Cobol-WOW DLL must be distributed with your Cobol-WOW applications.

Windows provides hundreds of functions for application programming, collectively referred to as the Windows API. The interface to these functions is a C-language interface that does not accept COBOL data types. Sometimes the architecture of these functions prevents direct access from COBOL. The Cobol-WOW DLL provides a COBOL interface to these Windows functions, providing direct access to the power and flexibility of Windows. For more information, see the *Functions and Messages* online Help file.

Execution of a Windows program also generates a number of messages. Again, the generation and dispatching of these messages are designed for a C-language interface. The Cobol-WOW DLL conveniently captures, organizes, stores, and reports these messages to the COBOL application. For more information, see the *Functions and Messages* online Help file.

It is possible to customize the initialization file (**cblwow.ini**) in order to define default settings for various runtime activities. This is accomplished by using the Runtime page of the Preferences dialog box in the Cobol-WOW Designer or by manually adding a [WOWRT] section to the cblwow.ini file. For more information, see page **3**.

Cobol-WOW Thin Client

The Cobol-WOW Thin Client executable program, **tclient.exe**, which is installed on the Windows client workstation, begins the Thin Client session by connecting to the server. It loads the required DLLs (see page **274**) and reads the configuration file, **RpcPlus.ini**. The server, upon receiving this connection request, begins execution of the application on the server. The application runs as a normal RM/COBOL program on the server until a Cobol-WOW function is invoked. All Cobol-WOW functions are intercepted by special logic in the Cobol-WOW runtime, which routes the requests back to the client, where they are executed. This causes the user interface to be presented on the client. When the Cobol-WOW function completes execution, control is returned back to the server. The Thin Client portion of Cobol-WOW is discussed in more detail in Appendix E, *Using Cobol-WOW Thin Client*, beginning on page **273**.

Cobol-WOW Development Process Overview

Note The development process is discussed in more detail in Chapter 4, *Developing with Cobol-WOW*, beginning on page **57**.

You begin the Cobol-WOW development process by creating a project. A project is a development environment provided by Cobol-WOW to facilitate the creation of the multiple forms that make up your application's user interface. A project manages not only the form creation, but also provides the ability to add file access and other code to the rest of your program.

Next, you design forms. The form files that are contained in a project are also known as members. The default extension for a form file is **.wow**. A full range of form types, styles, system colors and fonts is available to create highly stylized forms.

You continue using the Cobol-WOW Designer to populate the form with controls selected from the Toolbox. The Toolbox provides the ability to add Windows intrinsic controls (default) and ActiveX controls to the form. Using the Properties dialog box, Cobol-WOW enables the appearance and functionality of each control to be fully tailored to your needs.

Next, you attach event-handling code to the graphical user interface objects: the form, Windows intrinsic controls, and ActiveX controls. The Cobol-WOW Designer provides a complete list of possible events for each object and includes an Event-Handling Code dialog box that can be used to easily add event-handling code using familiar COBOL statements. In addition, over 150 of the Windows API functions are available, all with parameters that use standard COBOL data types.

Once the event-handling code is complete, you can generate copy files to allow for easy integration of the form into a legacy COBOL application or into a new COBOL program — ready to compile and execute. The compile and execute processes are available from the Project menu in the Cobol-WOW Designer.

Cobol-WOW also makes it easy to test your program and debug your source code.

Windows Graphical Operating Environment

The elements of the Microsoft Windows graphical operating environment allow you to develop Windows applications with Cobol-WOW. These GUI elements are as follows:

- Forms
- Controls
- Properties
- Handles
- IDs
- Functions and Messages

In this section, you will examine the two types of objects used to build your user interface: forms and controls, and two unique identifiers for these objects, handles and IDs. The use of properties to customize the way in which the controls that you place on a form (or the form itself) appear and behave are also discussed, as are functions and messages.

Forms and Controls

In the past, COBOL programmers built user interfaces with two verbs, ACCEPT and DISPLAY. Under Windows, however, programmers build user interfaces with two types of objects. This illustrates the paradigm shift that has occurred in software development. User interface development has shifted from a process described by syntax to an entity built from different objects.

Cobol-WOW and Windows provide you with a wealth of user interface technology, vastly expanding your capabilities beyond anything you could attempt with COBOL ACCEPT and DISPLAY statements. This new approach is more powerful, more flexible, and more easily maintainable than traditional COBOL user-interface development — a true "win-win" situation.

In this section, you will examine the two types of objects used to build your user interface: forms and controls, and two unique identifiers for these objects: handles and IDs. The use of properties to customize the way in which the controls that you place on a form (or the form itself) appear and behave are also discussed.

Forms

Windows with a capital "W" refers to the Microsoft Windows operating system. The term windows with a lowercase letter refer to a displayable, rectangular object that a program asks the operating system to create. The window is the basic building block of

the user interface. Everything you see on the screen is contained in a window. Dialog boxes, command buttons, list boxes, and text boxes are all specialized types of windows.

The Windows operating system provides extensive capabilities to manipulate windows. Most of these capabilities apply equally to a command button or a main window with a title, scroll bars, and a System menu. One of the merits of Windows is that you can manipulate many different types of objects in the same way, since they are all windows.

These different kinds of windows are extremely flexible. They can be visible or invisible. They can be as large as the screen or be 0 pixels wide by 0 pixels high. They can be enabled or disabled. They can be moved and stretched dynamically by the user or the application program. They can even have other windows created inside them.

You can see that window is a very broad and general term. To avoid confusion, Cobol-WOW uses the term "form" to describe the windows you create in the Cobol-WOW Designer. These forms, however, are true Windows windows. Forms are the containers within which you group controls. In traditional programming, you placed fields on the screen or in a pop-up window. With Cobol-WOW, you place fields (that is, controls) in a form.

When a program creates a form, all the controls contained on the form are created. The form is the parent of the controls. If the form is moved, the controls move with it. If the form is hidden, the controls are hidden. If the form is destroyed, the controls are destroyed.

Although forms are quite versatile, most of your programming will be involved with manipulating controls, not forms.

The form is where you create the interface of your application during design time — the time during which you are designing, rather than running, your form. This form looks like a typical window and contains a System-menu box (also known as the Control-menu box), a title bar, a border, a client (or workspace) area, and Minimize and Maximize buttons. The form has only default properties associated with it.

Note The evenly spaced marks that appear on the form at design time are the grid. The grid makes it easier to align, reposition, and resize controls visually. The Show Grid option and the Snap to Grid option, which are enabled by default at design time, cause the edges of each control to align with the nearest grid points. You can, however, disable these commands from the Form menu. To specify the units of measure for the grid points (that is, the X and Y coordinates), choose Edit Preferences on the Options menu to display the Preferences dialog box. Click the Alignment tab.

For more information about forms, see Appendix A, *Setting Properties and Events for Intrinsic Controls and Forms*.

Controls

Controls are the primary mechanism for getting user input and displaying output. Controls replace the fields you used with the COBOL ACCEPT and DISPLAY statements. A large portion of the interface design consists of using controls to customize the forms that make up your application. Tool tips are available on controls at design time.

You have probably seen and used controls in other Windows-based software applications. Although they vary from one another in appearance and function, they are all windows, and, as such, can be manipulated in identical ways. They are all hidden, displayed, enabled, disabled, created, destroyed, moved, and resized in the same manner.

Cobol-WOW supports two broad categories of controls:

- **ActiveX controls**, which exist as separate files with an **.ocx** filename extension. These include controls that are available with 32-bit versions of the Windows operating system, such as the animation, toolbar, or progress bar controls, as well those available from third-party vendors. See Appendix B, *Working with ActiveX Controls*, for more information about ActiveX controls.

    Note Although ActiveX controls may have additional features, we recommend that you use intrinsic controls whenever possible for greater portability.

- **Intrinsic controls**, (or default controls), such as the command button or a check box. The intrinsic controls are the easiest controls to implement, because they are part of the Windows operating system. You do not need to install or distribute any special files to support them. They will work under any version of Windows. Intrinsic controls are always included in the Cobol-WOW Toolbox, unlike ActiveX controls, which can be removed from or added to the Toolbox. See Appendix A, *Setting Properties and Events for Intrinsic Controls and Forms*, for more information about intrinsic controls.

    Note If you are using Cobol-WOW to modify an existing RM/Panels panel library, Cobol-WOW refers to the objects called "data fields" in RM/Panels as "controls." See Appendix D, *Using Cobol-WOW with RM/Panels*, for more information.

Table 1 illustrates the intrinsic controls that appear on the Toolbox in the Cobol-WOW Designer window. These are the basic controls that are common to most dialog boxes in Windows and the ones that you are likely to use most frequently when designing the user interface for your application.

Table 1 — Intrinsic Controls

Note The pointer tool (the first tool in the Toolbox) provides a way to select the form or controls on the form, and move and resize forms and controls. It is not a control.

Icon	Control Name	Description
	Animation	Displays an AVI clip. An AVI clip is a series of bitmap frames that run like a movie. Only AVI files without sound can be played using the animation control.
	Bitmap	Displays bitmap files. The bitmap control acts like a command button when clicked.
	Check Box	Displays a Yes/No, True/False, or On/Off option. You can check any number of check boxes on a form at one time.
	Combo Box	Combines a text box with a list box. Allows a user to type in a selection or select an item from a drop-down list.
	Command Button	Carries out a command or action when a user chooses it.
	Date Time Picker	Allows the user to select a date and time, and to display that date-time in the specified format. **Note:** Not available in this release.
	Edit Box	Provides an area to enter or display text.
	Ellipse Shape	Draws the geometric shape of an ellipse on the form.
	Group Box	Provides a visual and functional container for other controls. It is generally used to enclose related controls (usually check boxes or option buttons).
	IP Address	Allows the user to enter a numeric address in Internet protocol (IP) format. This control also allows the application to obtain the address in numeric form. **Note:** Not available in this release.
	Line Shape	Draws a line on the form.
	List Box	Displays a list of choices from which the user can select one or more items.
	Month Calendar	Displays a monthly calendar. The calendar can display one or more months at a time. **Note:** Not available in this release.
	Option Button	Presents mutually exclusive options in an option control. Option buttons are usually used with the group box control to form groups where only one of the listed buttons can be selected at one time.

Table 1 — Intrinsic Controls (*Cont.*)

Icon	Control Name	Description
	Progress Bar	Displays a pattern of blocks that show the status of a long operation.
	Rectangle Shape	Draws the geometric shape of a rectangle on the form.
	Rounded Rectangle Shape	Draws the geometric shape of a rectangle with rounded corners on the form.
	Scroll Bar (Horizontal and Vertical)	Allows a user to add scroll bars (horizontal and/or vertical) to controls that do not automatically provide them. (These are not the same as the built-in scroll bars that are found with many controls.)
	Static Text	Displays text, such as titles or captions, in regular outlines or filled rectangles, that the user cannot interact with or modify.
	Status Bar	Displays status information in a horizontal window at the bottom of an application window.
	Tab	Acts as a container for other controls and places a series of tabs at the top of the container.
	Timer	Provides a measured time interval that can be tied to events
	Toolbar	Displays a series of buttons that can be placed at the top and/or bottom of a form
	Trackbar	Displays a window containing a slider and optional tick marks used to select a value or a set of consecutive values in a range.
	Updown	Consists of a pair of arrows the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control.

Properties

Forms and controls have a number of configurable characteristics. These characteristics are called properties. Properties are the primary means by which forms and controls are manipulated. Setting properties defines how forms and controls are displayed and how they function in the running application.

The properties of a form and control are initially defined in the Cobol-WOW Designer. During design time, you use the Properties dialog box, which lists each property and its value, to set the default (initial) properties of a selected form or control. That is only half the story, however. Most of those properties can also be altered and retrieved at runtime by the code you enter in the Event-Handling Code dialog box. Think about that for a

second. You have almost the same level of flexibility in customizing your user-interface at runtime that you do at design time.

While setting properties in the Cobol-WOW Designer is achieved through the Properties dialog box, retrieving and setting property values at runtime is accomplished primarily with the CALL statement and two Cobol-WOW functions, WOWSETPROP and WOWGETPROP, which provide a consistent method of getting and setting property values for forms and all types of controls. For more information, see the *Functions and Messages* online Help file.

The following sections introduce you to the WOWSETPROP and WOWGETPROP functions. A sample program demonstrates some of what you can do with properties at runtime.

Setting a Property Value at Runtime

A property value is set at runtime by calling a special Cobol-WOW function, WOWSETPROP. For example:

```
CALL WOWSETPROP USING WIN-RETURN OBJECT-H "PropertyName"
     PROPERTY-VALUE.
```

WIN-RETURN is a numeric field into which a value of 1 is returned if the operation is successful, or a value of 0 if it fails. Any numeric field may be used. WIN-RETURN is a numeric field declared in a Cobol-WOW copy file, **windows.cpy**.

OBJECT-H indicates the handle of the object whose property is to be set. This field could be the handle of a form or a control.

"PropertyName" contains the name of the property to be set. All properties have an alphanumeric name, which is not case-sensitive. This field can be an alphanumeric literal or an alphanumeric data item containing the property name.

PROPERTY-VALUE contains the value to which the property should be set. This field can be an alphanumeric or numeric literal, or a data item.

Getting a Property Value at Runtime

A property value is retrieved at runtime by calling a special Cobol-WOW function, WOWGETPROP. For example:

```
CALL WOWGETPROP USING WIN-RETURN OBJECT-H "PropertyName"
     PROPERTY-VALUE.
```

WIN-RETURN is a numeric field into which a value of 1 is returned if the operation is successful, or a value of 0 if it fails. Any numeric field may be substituted. WIN-RETURN is a numeric field declared in a Cobol-WOW copy file, **windows.cpy**.

OBJECT-H indicates the handle of the object whose property is to be retrieved. This field could be the handle of a form or a control.

"PropertyName" contains the name of the property to be retrieved. All properties have an alphanumeric name, which is not case-sensitive. This field can be an alphanumeric literal, as shown, or an alphanumeric data item containing the property name.

PROPERTY-VALUE is where the value of the property will be stored. It must be a data item, not a literal.

Benefits of Using WOWSETPROP and WOWGETPROP

You will use these two CALL statements frequently as you build your user interface. These calls are to Windows programming what the MOVE statement is to COBOL. Since these two CALL statements are used so extensively, they have three important and helpful characteristics.

1. You can retrieve and set multiple property values in a single CALL statement.

    For example, to retrieve the size and location of any object with one CALL statement:

    ```
    CALL WOWGETPROP USING WIN-RETURN OBJECT-H "TOP" TOP-VALUE
                                       "LEFT" LEFT-VALUE
                                       "WIDTH" WIDTH-VALUE
                                       "HEIGHT" HEIGHT-VALUE.
    ```

    To set the size and location of any object with one CALL statement:

    ```
    CALL WOWSETPROP USING WIN-RETURN OBJECT-H "TOP" TOP-VALUE
                                       "LEFT" LEFT-VALUE
                                       "WIDTH" WIDTH-VALUE
                                       "HEIGHT" HEIGHT-VALUE.
    ```

2. You can retrieve the numeric value of a Text property. The following example sets the text of an edit field to an alphanumeric value that represents a negative decimal number. Then it retrieves that value directly into a signed numeric field with decimal

digits. By doing so, this operation prevents you from having to translate the alphanumeric value into a numeric value within your code.

```
01 DEC-FIELD                 PIC S9(5)V99.

CALL WOWSETPROP USING WIN-RETURN OBJECT-H "Text" 123.45-".
CALL WOWGETPROP USING WIN-RETURN OBJECT-H "TEXT" DEC-FIELD.
```

3.   You can set the value of a Text property directly from a numeric field. For example:

```
01 DEC-FIELD                 PIC S9(5)V99 COMP-3.

MOVE 512.1 TO DEC-FIELD.
CALL WOWSETPROP USING WIN-RETURN OBJECT-H "Text" DEC-FIELD.
```

The edit field will display "512.10".

Sample Program — Setting Properties

The sample project, PROPRTES, demonstrates how some common properties can be set and retrieved at runtime with these two functions. Using the Cobol-WOW Designer, look at the event-handling code attached to the Click event for each of the buttons to see how WOWSETPROP and WOWGETPROP are used. The variables used for retrieving property values are declared in the Common Working Storage area of the form.

Handles

In a Windows graphical interface, a handle is a number that can be used to uniquely identify and access a window's object. While most handles are associated with windows, other types of objects, such as fonts and bitmaps, can also have handles. For example, when a window is created, Windows assigns it a numeric identifier that is specific to that particular window. This number is the window's handle. The handle is then used to identify the window when Windows wants to inform you of activity for the window, or when you tell Windows to take some action on the window.

The handle is a subscript into an internal table of information maintained by Windows. Using this handle, or subscript, to identify the window gives Windows the ability to relocate its internal information without affecting your application program.

A handle is valid from the time the object is created until the time the object is destroyed. Once the object is destroyed, the handle may be reused and assigned to another object. Handles are never saved from one session to another. They must always be stored when the object is created. Cobol-WOW automatically stores all the required handles when it creates objects, so you do not have to worry about this process.

IDs

An ID is a numeric identifier assigned by the developer to a control when it is created. While handles and IDs are both numerical identifiers of a window, there are several important distinctions between the two values. An ID is assigned by the developer; a handle is assigned by the Windows operating system. An ID may or may not be unique; a handle is always unique. An ID is known at design time; a handle is not known until runtime and must, therefore, be stored for use.

Why does Windows support both types of identifiers? The window handle is essential to the functioning of the operating system. It provides a system-wide, unique identifier so that individual windows can be manipulated. Since several applications are running at once under Windows, the identifiers they use for windows must be unique for the entire system.

The window ID is for the developer's use in order to simplify the programming of user interaction in windows with controls. If you assign unique ID numbers to controls, application logic can be simplified. For example, an application program might create a window containing four controls: a name text box, an address text box, an OK command button, and a Cancel command button. The application program could assign ID numbers of 1, 2, 3, and 4, respectively, to these controls. The rest of the application code could use the ID numbers to identify the controls, rather than use their window handles.

Windows always uses the window handle to identify the window when it reports events that have taken place for the window. Sometimes it also provides the window ID. Some of the actions you can take on windows allow you to specify either the window handle or the ID.

Cobol-WOW makes it easy to use both handles and IDs. Data items containing both values are generated in a copy file so you can use the data name to specify the ID or handle in your code.

Functions and Messages

While properties are the primary method for manipulating controls in your programs, there are two other methods of handling controls: functions and messages.

When Windows was developed, functions and messages were the primary way of manipulating controls. In fact, the intrinsic controls do not actually have properties. Cobol-WOW imposes a property interface on top of the controls to give you a consistent method for using intrinsic and ActiveX controls.

Since Windows did not implement properties for the intrinsic controls, it provided hundreds of functions and messages to use with them. This large number of functions and

messages, each with its own unique set of parameters, may seem confusing at first. They do, however, provide a great deal of flexibility that you can use to supplement setting properties, which is the new approach to using these controls. When using properties, you need only remember the property name. The syntax for setting and getting all properties is the same.

Note In most cases, you will use only properties when manipulating ActiveX controls, since these were developed with an emphasis on properties. However, you may use a few functions with ActiveX controls when working with list boxes or combo boxes.

What are Functions?

A function is a callable subroutine, contained either in the Windows or Cobol-WOW runtime, that can be passed COBOL parameters and that will perform some special processing. A function is always executed with a CALL statement. The term "function" is commonly associated with C-language programming. In Cobol-WOW, functions are non-COBOL routines (or subprograms) that allow you to use the capabilities of the Windows operating system. To COBOL applications, functions are non-COBOL callable subprograms. Because most documentation on Windows will refer to these subprograms as functions, that term is used here.

Functions allow you to adjust the initial state of the forms and controls that you create in the Cobol-WOW Designer. A function executes code that can be used to carry out a specific task. Most of the functions that you will use are in the Windows application programming interface (API). Other functions are specifically designed for ActiveX controls; the remainder are provided to address issues exclusive to COBOL.

Cobol-WOW supports Windows API functions, ActiveX control functions, and Cobol-WOW functions. Cobol-WOW has tailored the Windows API to COBOL in order to simplify its use. It has also maintained a close parallel to the C-language syntax. These approaches should allow you to use general reference information on the Windows API from other sources to expand your knowledge of the API. Where substantial differences exist from the standard API functions, the Cobol-WOW documentation notes those differences. For more information, see the *Functions and Messages* online Help file.

What are Messages?

Messages are the means of communicating between your application program and the Windows operating environment. The Windows operating system sends messages to your program to give you an opportunity to respond to events. You send messages to Windows to tell it what you want it to do. (This second use is very similar to executing a Windows function. In fact, many Windows functions simply send messages.)

Windows reports hundreds of messages to your application. We recommend that you allow Cobol-WOW to interpret these messages. Although you can write your own message interpretation code, this is an advanced task that should not be attempted until you have significant experience in developing with Windows.

Since all the messages are Windows messages, they are intended by Windows for use with forms and the intrinsic controls. Messages cannot be sent directly to ActiveX controls.

For more information, see the *Functions and Messages* online Help file.

Using Functions and Messages

Cobol-WOW has a feature that makes it very easy to use the enormous collection of functions and messages. The Code Templates list in the Cobol-WOW Designer's Event-Handling Code dialog box lists code templates of every function and message that can be used with forms and controls.

When you select a name in this list, Cobol-WOW inserts into the Code Entry area a full description of the function or message, the COBOL syntax for its use, and a description of each parameter in your event-handling code. You simply replace the parameter names and values with your own, and the function or message is ready to use. (The manner in which code is displayed in the Event-Handling Code dialog box is configured by default. The Preferences dialog box provides several pages of configuration options that you can modify. To change this option on the Code page of the Preferences dialog box, click **Edit Preferences** on the **Options** menu and then click the **Code** tab.)

While the large number of functions and messages provide an overwhelming amount of functionality, there is a significant amount of overlap between them. For example, the SETWINDOWTEXT function and the WM-SETTEXT message both set the text of a window. When you use the SETWINDOWTEXT function, it merely sends a WM-SETTEXT message to the window.

Sample Program — Using Functions and Messages

The sample project, FUNCMESG, demonstrates the use of functions and messages with a list box control. The list box and combo box have the most dependence on functions and messages of any of the Windows intrinsic controls. Look at the event-handling code attached to each button to see how the function was executed or the message was sent.

Chapter 4: Developing with Cobol-WOW

This chapter is designed to provide essential background information to help you understand what you are doing and why. Then, it looks at how you approach common types of programs under Windows and how you take advantage of Windows' features. These concepts are illustrated by simple, but functional, sample programs.

The topics covered in this chapter include the following:

- Cobol-WOW Projects

- Event-Driven Applications

- Addressing Issues in Data Entry Programs

- Working with Menus

Cobol-WOW Projects

Most of the time, your user interface will consist of multiple forms. After you have created your forms, you will want to add file access and other code to the rest of the program. To provide these capabilities in a seamless environment, Cobol-WOW provides a facility called a project.

By using a project, the Cobol-WOW Designer allows you to do your complete development in an integrated, visual framework. The default extension for Cobol-WOW project filenames is **.wpj** (see page **200**). The **.wpj** file is a text file that contains project configuration information and a list of the forms included in the project.

When you create a project, Cobol-WOW lets you specify the forms that are used in the project. Not only will Cobol-WOW keep track of all the forms that are part of the project, it will create a skeleton COBOL program that creates, operates, and removes all of the forms. Better yet, you can edit any part of this COBOL program from inside the Cobol-WOW Designer. The Event-Handling Code dialog box lists every code section of the COBOL program in the Events/Code Sections list box. You can copy in your file descriptions, declaratives, create additional Working Storage data items — in short, everything — from within the Designer.

Cobol-WOW assumes that you will be working in a project. In the Cobol-WOW Designer window, all the forms in a project are displayed in the project tree. The Project menu provides all the commands necessary for working with the project.

Note Cobol-WOW v3.0 is project-based. If you have a form-based application created with an earlier version of Cobol-WOW, you must create a project and add the form files in the existing application to it.

Event-Driven Applications

Even before Windows came along, COBOL programmers were not the only ones struggling with how to code user-input logic. Everybody else was too. The developers of Windows took a new approach to user input, which is reflected in Cobol-WOW. This new concept is called "event-driven" programming, as opposed to the more traditional method, sequential programming.

In sequential programming, the programmer dictates the exact sequence of events in the program. The user is directed to enter field 1, then field 2, and so forth. With this method, the programmer always knows what is going to happen. In actual use, however, users generally want to be in charge and enter things in whatever manner they wish.

Event-driven programming allows users to have that flexibility. The user is in control and makes the program respond to the user's actions. Every time an action occurs on a field, an event is triggered. The program then responds to those events.

How does this work? First, you tell Windows that you want this field, this field, and that field on the screen. Windows creates these elements. Then Windows allows users to do whatever they want with those fields. Whenever a user does something, Windows tells the developer what is going on by communicating events to use (such as field changed, mouse clicked, and so forth). You attach your program logic (code) to these events.

The following examples compare traditional COBOL programming and event-driven programming implemented under Windows.

Example 1

```
ENTER-CUST-ID
   ACCEPT CUST-ID LINE 4 POSITION 10.
   IF F3-KEY
        PERFORM LOOK-UP-CUST.
   PERFORM VALIDATE-CUSTOMER.
   IF NOT VALID-CUST
        PERFORM BAD-CUST
        GO TO ENTER-CUST-ID.
```

In traditional COBOL programming, the example shown above performs three operations:

1. Accepts the customer ID.

2. Performs a lookup when the F3 key is pressed.

3. Validates the customer number before the user can proceed.

Under Windows, you simply take the same logic and distribute it to the appropriate events. Examine these same three operations when implemented under Windows:

1. The COBOL ACCEPT statement would be eliminated because Windows handles it.

2. The lookup would probably be associated with a button or menu command, rather than the F3 key. You would attach PERFORM LOOK-UP-CUST to one or all of these events.

3. The VALIDATE-CUSTOMER validation would be attached to two events: LostFocus and Click. The first event, LostFocus, occurs when the user finishes entry into a field and moves to another field. The second event, Click, occurs when the user clicks the OK button to signal completion of all information on the window. This validation is important because the user may never even access the CUST-ID control (unless you position him there). If the validation failed, you tell Windows to put the user back into the CUST-ID control.

In one way, this does make it less convenient, because the logic is in several places rather than one.

Example 2

```
ENTER-CUST-STREET-1.
   ACCEPT CUST-STREET-1 LINE 7 POSITION 10 TAB UPDATE NO BEEP.
   IF UP-ARROW
        GO TO GET-CUST-NAME.
   IF DOWN-ARROW
        GO TO GET-CUST-STREET-2.
```

No special processing is associated with this field; the only requirements are the data entry fields and logic to provide keyboard control over what field is entered next. With Windows and Cobol-WOW, however, this processing is all automatic. You do not need to replace the code; you simply discard it.

Take a minute to think about your data entry screens and logic. Instead of writing all the logic to implement those screens, Cobol-WOW enables you to write only the logic to implement special features, thereby substantially reducing the size of an average COBOL program.

Addressing Issues in Data Entry Programs

COBOL is often used to create data entry programs. Data entry programs have unique requirements and issues that are not ordinarily discussed in programming literature. Over the years, COBOL developers have adopted fairly common techniques for addressing these issues in a character-based environment. This section discusses these issues and suggests how they could be addressed under Windows with Cobol-WOW.

While Windows was designed around an exceptional user interface, it was not designed for data entry. However, there are practical ways to address the different sets of issues important to data entry programs, including:

- Handling Data

- Handling Different Types of Data

- Managing User Interaction

- Using Function Keys for Special Options

Handling Data

One set of issues important to data entry programs are those related to the manipulation of data. When data is read out of a file, how does it become displayed? How are numeric and date fields handled? How are the fields formatted? How is data moved from the user interface back to the file? This section, along with the following topics, discusses these issues.

When Windows creates a control, such as an edit box, it allocates its own storage space for the contents of that edit box control. When the user modifies the contents of the edit box control on the screen, Windows stores the new value in its own storage space and sends your program a message that the value changed. If you want the new value, you have to ask Windows for it. Windows will not automatically store the new value in your COBOL data item. The reverse is also true. Windows does not know when the value of your COBOL data item changes and will not automatically update an edit box control to display the new value. You have to send it the new value.

The following two examples show how data is transferred between COBOL data items in Working Storage on record areas and a form created under Windows.

Example 1: Loading a Form with COBOL Data

This example illustrates how to load a form with COBOL data.

In the following code section, the lines of code that contain the COBOL data are highlighted; the lines of code that move the data to the form are not highlighted.

```
01 CUST-FIELDS
   03 CUST-NAME               PIC X(40).
   03 CUST-CITY               PIC X(20).
   03 CUST-ST                 PIC X(2).

CALL WOWSETPROP USING WIN-RETURN CUST-NAME-H "TEXT" CUST-NAME.
CALL WOWSETPROP USING WIN-RETURN CUST-CITY-H "TEXT" CUST-CITY.
CALL WOWSETPROP USING WIN-RETURN CUST-ST-H "TEXT" CUST-ST.
```

The following figure illustrates the three fields on the form (the edit box controls labeled Name, City, and State) that will receive the transferred COBOL data.
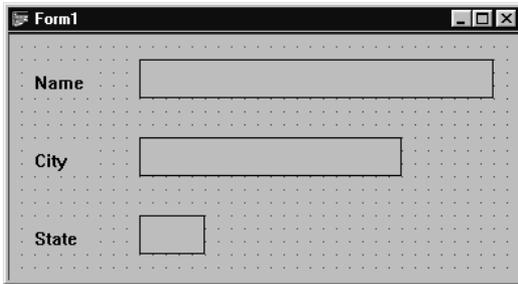
Example 2: Retrieving Information from a Form and Storing It in COBOL Data Items

This example illustrates how to retrieve information from a form and store it in COBOL data items.

In the following code section, the lines of code that contain the COBOL data are highlighted; the lines of code that retrieve the data from the form are not highlighted.

```
01 CUST-FIELDS
   03 CUST-NAME               PIC X(40).
   03 CUST-CITY               PIC X(20).
   03 CUST-ST                 PIC X(2).

CALL WOWGETPROP USING WIN-RETURN CUST-NAME-H "TEXT" CUST-NAME.
CALL WOWGETPROP USING WIN-RETURN CUST-CITY-H "TEXT" CUST-CITY.
CALL WOWGETPROP USING WIN-RETURN CUST-ST-H "TEXT" CUST-ST.
```

Most likely, you will want to create two procedures in your program for each data entry form. Create one procedure to set the value of the controls on the form from your COBOL data. Produce the second procedure to retrieve the value of the controls on the form into your COBOL data.

Let's say you created a form called CUSTFORM, and want to use it to update the contents of your customer file, CUSTFILE. The file and record would both contain fields such as CUST-NAME, CUST-CITY, and CUST-ST. You would create the following two procedures:

```
LOAD-CUST-FORM.
    CALL WOWSETPROP USING WIN-RETURN CUST-NAME-H
        "Text" CUST-NAME.
    CALL WOWSETPROP USING WIN-RETURN CUST-CITY-H
        "Text" CUST-CITY.
    CALL WOWSETPROP USING WIN-RETURN CUST-ST-H
        "Text" CUST-ST.

UNLOAD-CUST-FORM.
    CALL WOWGETPROP USING WIN-RETURN CUST-NAME-H
        "Text" CUST-NAME.
    CALL WOWGETPROP USING WIN-RETURN CUST-CITY-H
        "Text" CUST-CITY.
    CALL WOWGETPROP USING WIN-RETURN CUST-ST-H
        "Text" CUST-ST.
```

LOAD-CUST-FORM sets the Text property of each control based on the data in the file.

UNLOAD-CUST-FORM sets the value of each field in the record based on the Text property of each control.

When you want to update the data on the screen from the record, you execute PERFORM LOAD-CUST-FORM. When you want to update the data in the record from the screen, you execute PERFORM UNLOAD-CUST-FORM.

The handle fields, such as CUST-NAME-H, CUST-CITY-H, and CUST-ST-H, contain the handle of the control. The data fields, such as CUST-NAME, CUST-CITY, and CUST-ST are the COBOL data items. "Text" indicates the control property that stores the value or contents of the control. The property name to use will depend on the control.

In a typical data entry situation, the program works as follows:

1. Read a record from the file.

2. Execute PERFORM LOAD-CUST-FORM to display the values in the form.

3. Let the user modify the values (Windows handles this).

4. When the OK button is pressed:

   a. Execute PERFORM UNLOAD-CUST-FORM to put updated values in the record.

   b. Write/Rewrite the record to the file.

The drawback to this approach is that you have to create two procedures that list each control handle and data field. This means you have to maintain two procedures as you add or remove controls and fields. You can, however, alter the approach and consolidate this information in one procedure.

First, declare a new data item:

```
01 LOAD-FUNC                  PIC X(5).
```

Then rewrite your procedures as follows:

```
LOAD-CUST-FORM.
    MOVE WOWSETPROP TO LOAD-FUNC.
    PERFORM CUST-LOAD-UNLOAD.

UNLOAD-CUST-FORM.
    MOVE WOWGETPROP TO LOAD-FUNC.
    PERFORM CUST-LOAD-UNLOAD.

CUST-LOAD-UNLOAD.
    CALL LOAD-FUNC USING WIN-RETURN CUST-NAME-H
        "Text" CUST-NAME.
    CALL LOAD-FUNC USING WIN-RETURN CUST-CITY-H
        "Text" CUST-CITY.
    CALL LOAD-FUNC USING WIN-RETURN CUST-ST-H "Text"
        CUST-ST.
```

Although this may look unusual, it is actually fairly straightforward. The WOWGETPROP and WOWSETPROP routines (functions) are alphanumeric fields in **windows.cpy**. These routines contain the names of subprograms in the Cobol-WOW dynamic-link library (DLL), **wowrt.dll**. These routines are called by using a data name, not a literal name. Since they are alphanumeric data, you can MOVE them to LOAD-FUNC and CALL LOAD-FUNC instead of calling WOWGETPROP or WOWSETPROP. Because the syntax for WOWGETPROP and WOWSETPROP is identical, you can use the same statement for both. You now have the list of controls and fields in one place.

Handling Different Types of Data

Now that you have a logic structure for loading and unloading the controls in your forms, how do you deal with different types of data?

Alphanumeric data. Not surprisingly, managing alphanumeric data is the easiest. The edit box control and many ActiveX control equivalents have a "Text" (or similarly named) property that contains the alphanumeric data of the control. Generally, you will use some type of edit box control for alphanumeric data entry and then set the Text property of the control.

Numeric data. If you do not require special formatting, managing and supporting numeric data can be just as easy as alphanumeric data. See Example 1 below.

Special formatting of numeric data. Although the approach illustrated in Example 1 provides a simple way to handle basic numeric data, in some circumstances you will want to carefully control the format in which the numeric data is displayed. In these situations, you will need to perform the formatting in your COBOL program, then use the formatted value to set the control text. Example 2, on page **65**, illustrates formatted numeric data.

Some controls, for example scroll bars, are designed to manipulate a numeric value. See Example 3 on page **66**.

Other controls, such as buttons and check boxes, often represent a True or False value. Consequently, these types of controls need a different approach for handling numeric data, as illustrated by Example 4 on page **66**.

Example 1: Basic Numeric Data for an Edit Box Control

The edit box control does not have any special support for numeric data. Cobol-WOW, however, does provide this functionality. When you pass a numeric literal or data item while setting the Text property, Cobol-WOW converts it to a string and passes the string to Windows. When you use a numeric data item while getting the Text property, Cobol-WOW retrieves the text from Windows, converts it to a numeric value, and returns the numeric value. Let's see how this works.

This function call will set the text of the control to 127 because Cobol-WOW takes the numeric value and converts it to a text string.

```
01 COMP-FIELD          PIC 9(5) COMP VALUE 127.
   CALL WOWSETPROP USING WIN-RETURN EDIT-H "TEXT" COMP-FIELD.
```

Cobol-WOW does not provide any flexibility in numeric formatting with WOWSETPROP. It will always zero suppress leading zeros and display all trailing decimal zeros. For example, this function call will set the text of the control to 127.00.

```
01 CUST-BAL              PICS 9(7)V99 VALUE 127.
    CALL WOWSETPROP USING WIN-RETURN EDIT-H "TEXT" CUST-BAL.
```

The numeric capabilities of WOWGETPROP are less interesting to illustrate. For example, the following function call will store the numeric value of the text of the edit box control in CUST-BAL. If the text value contains more than five integers or two decimal digits, the remaining digits are truncated.

```
    CALL WOWGETPROP USING WIN-RETURN EDIT-H "TEXT" CUST-BAL.
```

Example 2: Formatted Numeric Data for an Edit Box Control

When special formatting of numeric data is required, you will need to perform the formatting in your COBOL program, then use the formatted value to set the control text. For example:

```
01 YMD-DATE              PIC 99/99/99.
    MOVE CUST-LAST-PURCHASE TO YMD-DATE.
    CALL WOWSETPROP USING WIN-RETURN EDIT-H "TEXT" YMD-DATE.
```

You can still retrieve the numeric value with the CALL to WOWGETPROP.

```
    CALL WOWGETPROP USING WIN-RETURN EDIT-H "TEXT" YMD-DATE.
```

Notice that the calls to WOWGETPROP and WOWSETPROP are now different, which affects the coding strategy outlined for handling basic numeric data. You now need to modify your approach as follows:

```
LOAD-CUST-FORM.
    MOVE WOWSETPROP TO LOAD-FUNC.
    PERFORM CUST-LOAD-UNLOAD.
    MOVE CUST-LAST-PURCHASE TO YMD-DATE.
    CALL WOWSETPROP USING WIN-RETURN EDIT-H "TEXT" YMD-DATE.

UNLOAD-CUST-FORM.
    MOVE WOWGETPROP TO LOAD-FUNC.
    PERFORM CUST-LOAD-UNLOAD.
    CALL WOWGETPROP USING WIN-RETURN EDIT-H "TEXT" YMD-DATE.

CUST-LOAD-UNLOAD.
    CALL LOAD-FUNC USING WIN-RETURN CUST-NAME-H
        "Text" CUST-NAME.
    CALL LOAD-FUNC USING WIN-RETURN CUST-CITY-H
        "Text" CUST-CITY.
    CALL LOAD-FUNC USING WIN-RETURN CUST-ST-H "Text" CUST-ST.
```

Example 3: Handling Numeric Data with Scroll Bar Controls

Numeric values are easy to handle with scroll bar controls. Simply use any type of numeric field with the desired property as follows:

```
01 NUM-VALUE                 PIC 9(5) COMP-6.
   CALL WOWSETPROP USING WIN-RETURN SCROLLBAR-H "Value" NUM-VALUE.
```

or

```
   CALL WOWGETPROP USING WIN-RETURN SCROLLBAR-H "Value" NUM-VALUE.
```

Example 4: Handling Numeric Data with Check Box Controls

Several types of controls often represent a True and False value: the value True corresponds to a numeric value of 1, and the value False corresponds to a numeric value of 0. These kinds of controls are often used to represent the value of a data item with 88-level condition names. If the data item is numeric and the conditions are 0 and 1, this is very straightforward.

The following example shows that you are using a check box control to indicate whether a customer is active or inactive:

```
01 CUST-ACTIVE                       PIC 9.
   88 CUST-IS-ACTIVE          VALUE 1.
   88 CUST-IS-INACTIVE        VALUE 0.
```

The following function call will set the check box state:

```
   CALL WOWSETPROP USING WIN-RETURN CB-H "State" CUST-ACTIVE.
```

The following function call will retrieve the check box state:

```
   CALL WOWGETPROP USING WIN-RETURN CB-H "State" CUST-ACTIVE.
```

If your data item and condition name are not numeric, or have values other than one and zero, you will have to use logic more like that shown in the following example:

```
01 CUST-WHLSLE                       PIC X.
   88 CUST-IS-WHLSLE          VALUE "Y".
   88 CUST-IS-NOT-WHLSLE      VALUE "N".
```

To set the check box state:

```
   IF CUST-IS-WHLSLE
      CALL WOWSETPROP USING WIN-RETURN CB-H "State" WIN-TRUE
   ELSE
      CALL WOWSETPROP USING WIN-RETURN CB-H "State" WIN-FALSE.
```

To retrieve the check box state:

```
CALL WOWGETPROP USING WIN-RETURN CB-H "State" NUM-VALUE.
IF NUM-VALUE = 1
    SET CUST-IS-WHSLE TO TRUE
ELSE
    SET CUST-IS-WHSLE TO FALSE.
```

Managing User Interaction

Another type of issue that you must deal with in data entry programs involves user interaction. Although there are reasonable approaches to use under Windows to address this issue, COBOL programmers are generally unaccustomed to implementing them. This section covers a range of topics pertaining to this issue and provides examples illustrating how to respond to user actions in your Cobol-WOW applications.

Handling input validation. In data entry programs, it is common to want to validate the contents of a field after it is entered. In character-based applications this process was easy as you did it after the COBOL ACCEPT statement. However, in Windows your Cobol-WOW application would respond to an event, which represents user actions, associated with the field (control) that your application can recognize. Every field (control) has certain events to which it can respond. See Example 1 on page **68**.

Dictating entry order for controls. Character-based data entry programs generally dictate a specific entry order for fields (controls). Although Windows programs usually do not dictate such a specific order, they can easily support one by using the Tab key to move through controls in a default tab order. See Example 2 on page **68**.

Preventing data entry on a control. If you do not want a user to enter data in a particular field (control), you must disable it, as detailed in Example 3 on page **69**.

Switching to another Windows application. Your program needs to be flexible enough to accommodate moving between applications if the user wants to switch to another Windows application. See two cases in point in Example 4, on page **70**.

Disabling and enabling a validated control. When a user completes data entry of a key field (control), such as the customer ID, and the value is validated, you do not want the user to return and change the value of the key control. See Example 5 on page **73**.

Example 1: Handling an Invalid Value

In Windows, your first response to handle field validation might be to watch for users to press the Enter key, indicating they had completed the field. However, Enter is not the key usually used for moving between fields (controls) under Windows. (The Enter key is discussed in Example 2 below.) Such a response also overlooks the use of the mouse: the user might have clicked on another field with the mouse, rather than pressed any key on the keyboard.

In a Cobol-WOW application, there are two reasonable places to perform input validation on a field (control): in the Change event or in the LostFocus event. It is preferable, however, to perform input validation in the LostFocus event rather than in the Change event. You can assume that when the user leaves the control, a value has been entered. The Change event occurs every time the user or your program changes the value of a control, for example, on every keystroke or whenever a WOWSETPROP routine is called. Unless you want to validate at all these times, the LostFocus event is the most feasible strategy as it indicates that the input focus is moving away from the control.

You can get the value of the control and validate it in the LostFocus event. What if the value is invalid? Instead of going back to the ACCEPT statements (as you would in character-based programs), under Windows, you can force the user back to the invalid control with the SETFOCUS function. In this case, the LostFocus event logic is executed as follows:

```
CUST-TYPE-LOSTFOCUS.
    CALL WOWGETPROP USING WIN-RETURN CUST-TYPE-H "TEXT" CUST-TYPE.
    PERFORM VALIDATE-CUST-TYPE.
    IF CUST-TYPE-IS-INVALID
        PERFORM INVALID-CUST-TYPE-MSG
        CALL SETFOCUS USING WIN-RETURN CUST-TYPE-H.
```

The user will not be allowed to leave the CUST-TYPE field until a valid value is entered. The SETFOCUS function solution, however, has implications on switching to another Windows application, as discussed in Case 1 of Example 4 (see page).

Example 2: Dictating Entry Order for Controls

A default order for moving through controls can be assigned in the Cobol-WOW Designer through the TabIndex property. The TabIndex is the order through which the controls should be moved when the user presses the Tab key. Notice that there is also a TabStop property. Windows will stop at controls with TabStop set to True only when the Tab key is pressed. The Enter key is generally used to indicate that the default button on the form should be pressed; it is not used for moving between controls.

In some situations, such as in the preceding input validation example, you may want to position the user on a specific control. This is performed with the SETFOCUS function.

You can use SETFOCUS to override the default tab order (see page **19**) by detecting the Tab key in the KeyDown event and calling the SETFOCUS function. You also can disable automatic tabbing between controls by setting the DialogMotion property of the form to False.

Example 3: Preventing Data Entry on a Control

In character-based applications, it was easy to prevent a user from entering a value into a field. You simply did not ACCEPT it. Under Windows, any enabled control on a form can be accessed by the user. The key word here is "enabled." If you do not want a user to Tab to or click a control, you must disable it.

For example, you have a customer maintenance form with Customer ID as the key control. You want the user to enter the customer identification number, then you will read the file, load the form, and let the user modify the rest of the fields. If you simply present the form with all the controls enabled, there is no way to prevent the user from clicking one of the other controls before completing the Customer ID control. Disabling all the other controls on the form, however, is inconvenient.

Let's look at how you might handle this situation if you use an edit box control with the user entering the customer ID. (A more appropriate solution in this situation, however, would be to use a combo box for the Customer ID control, since it allows the user to either enter a customer ID or select a customer from a list.)

The first issue is positioning the user in the Customer ID control. Cobol-WOW automatically positions the user in the first control (set by the TabIndex property) of the form when the form is created. To avoid destroying and recreating the form every time the user wants to access a different customer, you will use the SETFOCUS function. Add the following code to your OK and Cancel buttons, so that after every completed or canceled maintenance operation, the user will be repositioned in the Customer ID control. Be sure to place this code after the other OK or Cancel command button logic. For example:

```
OK-CLICK.
*Followed by logic to save data.
.
.
.
CALL SETFOCUS USING WIN-RETURN CUST-ID-H.

CANCEL-CLICK.
*Followed by logic to cancel changes.
.
.
.
CALL SETFOCUS USING WIN-RETURN CUST-ID-H.
```

You also may want to add the SETFOCUS call to the Create event for your form. Then your code will not be sensitive to the TabIndex value of the CUST-ID control.

Now that you know the user will start with the CUST-ID control, you need to keep the user there until a valid customer ID is entered. However, the user may switch to another Windows application. How can you handle this? See case 2 in Example 4 on page **71**.

Example 4: Switching to Another Windows Application

Case 1. What if the user, however, wants to switch to another Windows application? Using the logic in Example 1 (see page **68**), that would not be possible. Perhaps using the SETFOCUS function is not the best solution.

Let's say that the customer type control is one of many controls on the form. The user begins to enter the value, then decides to switch to another application. Your LostFocus code is executed and you determine the customer type is invalid. You display a warning message, but do not call the SETFOCUS back to CUST-TYPE. The user moves on to the other application, then switches back to your application by clicking a field other than CUST-TYPE. The customer type control now contains an invalid value. To protect the integrity of your data, you will need validation logic somewhere else in order to detect this response. The OK button would appear to be an ideal place, as presumably your user will click the OK button to save the data. Then, you could set focus back to the CUST-TYPE field if the value is invalid, as shown in the following example:

```
OK-CLICK.
   PERFORM VALIDATE-CUST-TYPE.
   IF CUST-TYPE-IS-INVALID
        PERFORM INVALID-CUST-TYPE-MSG
        CALL SETFOCUS USING WIN-RETURN CUST-TYPE-H.
```

This is, of course, a matter of personal preference. Windows applications should be as flexible as possible. From a programming viewpoint, it would be simpler to include the SETFOCUS in the LOSTFOCUS logic, although it would inconvenience your users. Without the SETFOCUS, the LOSTFOCUS logic looks like the following:

```
CUST-TYPE-LOSTFOCUS.
   CALL WOWGETPROP USING WIN-RETURN CUST-TYPE-H "TEXT" CUST-TYPE.
   PERFORM VALIDATE-CUST-TYPE.
   IF CUST-TYPE-IS-INVALID
        PERFORM INVALID-CUST-TYPE-MSG.
```

Case 2. If the user clicks on another control on the form, you want to keep them on CUST-ID, as discussed in Example 3. If they click another application, however, you want to let them move on to that program. Is there a simple way you can tell if they are moving to another application?

Windows provides a function called ISCHILD, which tells you whether a control is a child of a form. You can use this function to determine whether the user has clicked on another control on the form. Here is an example of the logic:

```
CUST-ID-LOSTFOCUS.
*Get the cust ID and validate it, as described previously.
.
.
.
IF CUST-ID-INVALID
    CALL GETFOCUS USING CURRENT-H
    CALL ISCHILD USING WIN-RETURN DATANTRY-H CURRENT-H
    IF WIN-RETURN = WIN-TRUE
        CALL SETFOCUS USING WIN-RETURN CUST-ID-H.
```

First, use the GETFOCUS function to determine what form or control has focus. Then, use the ISCHILD function to determine whether that form or control (CURRENT-H) is a child control of the form (DATANTRY-H). If it is, set focus back to the CUST-ID control. Otherwise, you can let the focus go to wherever the user places it.

That process, however, solves only half the problem. What happens when the user clicks back on the same form, but to a different control? You need to catch that event too and force the user to the CUST-ID control.

Whenever a form or control gets focus, the GetFocus event occurs. When a control on an inactive form gets focus, the GetFocus event occurs for both the form and the control. If the user switches back to the form after switching to some other application, no matter what control is clicked on, the GetFocus event will occur for the form. You can add a SETFOCUS call to the form's GetFocus event to make sure the user goes back to the CUST-ID control.

There is one more detail. If the user has already completed the CUST-ID field, you do not want to force the user back to it. You could determine whether to force the user to the CUST-ID control by validating CUST-ID again, but that might disrupt the file position or some data value in the record. Instead, modify the LostFocus code to set a flag as shown in the following example:

```
CUST-ID-LOSTFOCUS.
*Get the cust id and validate it, as described previously.
.
.
.
    IF CUST-ID-INVALID
          CALL GETFOCUS USING CURRENT-H
          CALL ISCHILD USING WIN-RETURN DATAENTRY-H CURRENT-H
          IF WIN-RETURN = WIN-TRUE
                CALL SETFOCUS USING WIN-RETURN CUST-ID-H
          ELSE
                SET FORCE-FOCUS TO TRUE
          END-IF
    ELSE
          SET FORCE-FOCUS TO FALSE.
```

Now you can add this code to the GetFocus event for the form:

```
DATANTRY-GETFOCUS.
    IF FORCE-FOCUS
          CALL SETFOCUS USING WIN-RETURN CUST-ID-H.
```

The user will have to enter a valid customer ID before anything else can be done on the form.

Example 5: Disabling and Enabling a Validated Control

To prevent a user from returning and changing a value after it has been validated, you need to make one more change to the LostFocus code:

```
CUST-ID-LOSTFOCUS.
*Get the cust id and validate it, as described previously.
.
.
.
   IF CUST-ID-INVALID
          CALL GETFOCUS USING CURRENT-H
          CALL ISCHILD USING WIN-RETURN DATANTRY-H CURRENT-H
          IF WIN-RETURN = WIN-TRUE
                 CALL SETFOCUS USING WIN-RETURN CUST-ID-H
          ELSE
                 SET FORCE-FOCUS TO TRUE
          END-IF
   ELSE
          CALL ENABLEWINDOW USING WIN-RETURN CUST-ID-H WIN-FALSE
          SET FORCE-FOCUS TO FALSE.
```

The ENABLEWINDOW function, when used with the argument WIN-FALSE, disables the control. In order to enable it, return to the logic for the OK and Cancel buttons:

```
OK-CLICK.
   *Followed by logic to save data.
   CALL ENABLEWINDOW USING WIN-RETURN CUST-ID-H WIN-TRUE.
   CALL SETFOCUS USING WIN-RETURN CUST-ID-H.

CANCEL-CLICK.
   *Followed by logic to cancel changes.
   CALL ENABLEWINDOW USING WIN-RETURN CUST-ID-H WIN-TRUE.
   CALL SETFOCUS USING WIN-RETURN CUST-ID-H.
```

You need to make sure you enable the control before you set focus to it. You cannot set focus to a disabled control.

Using Function Keys for Special Options

Another technique commonly used in character-based data entry programs is that of using function keys for special options. This also can be accomplished under Windows, although before we describe how it is done, let's examine some more Windows programming principles.

Windows, a feature-rich, flexible environment, allows you to develop software that will work virtually in any capacity you wish. That being said, you need to do things the Windows way. Not because it is necessarily better, or because it is an industry standard, but because it will make your coding easier. While Windows is very flexible, it was designed with a certain orientation, which was not function-key nor data-entry-program driven.

It is important to know how to do under Windows what you could do in a character-based environment. This is the skill set and basic approach to software development you have perfected over the years. However, mirroring that approach exactly under Windows will be more difficult than transitioning to a more Windows-like approach for you and your users both.

Function keys are a good example of this point. The KeyDown and KeyUp events, provided on virtually all controls, return the value of the key pressed, thereby making function key detection possible. The Windows approach to software design, however, mandates the use of pulldown menus or command buttons for executing the type of functionality you have been used to assigning to function keys. We recommend that you give serious consideration to implementing these approaches before implementing function keys.

Implementing Function Keys in Cobol-WOW

Note The following description on how to detect function keys applies only for Windows intrinsic controls.

When any key is pressed, the KeyDown event is triggered. When it is released, the KeyUp event is triggered. If the key that was pressed and released was an ASCII key, the KeyPress event is also triggered.

All of these events return a value identifying the key in WIN-WPARAM. Cobol-WOW automatically moves this value to WIN-KEY, which is a numeric field that is redefined to include a one-byte, alphanumeric field, WIN-CHAR. You can, therefore, examine the key as numeric or alphanumeric data.

If the value is that of an ASCII character, WIN-CHAR will contain the alphanumeric character value. Otherwise, WIN-KEY will contain a numeric value identifying the key.

This value is called a virtual key code. The value in WIN-KEY can be compared to the virtual key values defined in **windows.cpy**. The names of these values more or less correspond to the key names.

Now you are ready to detect function keys (remember, however, that this is not the Windows approach). If you want to use F7 key to trigger a customer lookup in the CUST-ID field, add the following code to the KeyDown event for the CUST-ID field:

```
CUST-ID-KEYDOWN.
   IF WIN-KEY = VK-F7
         PERFORM CUSTOMER-LOOKUP.
```

What if you added several special key actions to the same event? In this case, you might want to switch to the EVALUATE statement, although this step is not recommended:

```
CUST-ID-KEYDOWN.
   EVALUATE WIN-KEY
         WHEN VK-F7          PERFORM CUSTOMER-LOOKUP
         WHEN VK-F20         PERFORM …
         WHEN VK-NUMLOCK     PERFORM …
         WHEN VK-ADD         PERFORM …
         WHEN VK-NUMPAD3     PERFORM …
         WHEN VK-PRINT       PERFORM …
   END-EVALUATE.
```

This key detection will be active only for the CUST-ID field. What if you want to assign a global function key action that applies to every field on the form? The KeyPress event for the form, however, is triggered only under one of the following conditions:

- When the form is active.

- No control on the form has focus.

- A key is pressed.

When a control has focus and a key is pressed, the form KeyPress event is not triggered. You can, however, simulate it by adding the following code to the KeyPress event for each control on the form:

```
PERFORM FORMNAME-KEYPRESS.
```

Then, the EVALUATE or IF statement used for key detection could be placed in the form's KeyPress event and would be executed when any key is pressed in any control, providing global detection. This behavior is very non-Windows-like. Windows provides accelerators for buttons and menu commands.

Sample Program

The sample project, DATANTRY, demonstrates all of the techniques discussed in this section except function key detection.

DATANTRY is a very simple data entry program that allows maintenance of a file with only one record in it. It is a customer record with a key value of 000001. Enter the key value and press Tab or click another field. The file will be read and the data displayed. Make any changes you want and press OK to save them or Cancel to discard them.

This sample is not intended to demonstrate how to design your user interface under Windows. You should use some type of list box or combo box for entering customer numbers. You might want to support Add options in your customer maintenance program. This program simply illustrates how to implement the types of approaches we used to use in the character-based world under Windows.

Working with Menus

Menus provide a simple, consistent, and intuitive way to inform users of options (menu items) available when running a program. The Cobol-WOW Designer contains a Menu Editor dialog box (see page **13**) that makes menu creation easy. (See "Creating a Menu" in Chapter 2, *Tutorial*, for further information.)

A menu is another type of object you add to your form. A menu has two parts: the horizontal bar at the top of the form, which is always present, and the vertical menus that appear when a top-level item is selected. The line at the top of the form is called the top-level menu. The menus that "pop up" when a top-level item is selected are called pop-up menus. The top-level menu is actually constructed from the titles of the pop-up menus.

Menus are another built-in part of Windows, similar to intrinsic controls. You do not need to distribute any special files to support menus.

Using Menus

Menus are one of the simplest objects to use in your programs. They have one purpose: to indicate to the program that the user has selected an option (menu item).

A menu item can be selected from a menu in one of three ways. First, the user can select the menu item by clicking it. Second, by pressing the Alt key, the user can highlight the menu, use the arrow keys to move to a menu item, and press Enter to select it. Third, the

user can press an accelerator key that is associated with the menu item. Accelerator keys are assigned in the Menu Editor dialog box.

When a menu item is selected, the Click event for that menu item is executed. This is the only event available for menu items. There are no data associated with menu items. Menus are very similar to command buttons, in that they are a request for action.

Menus do not have properties like controls do, but there are some characteristics of menu items you may want to manipulate in your programs. The most common ones to use are the Checked/Unchecked and Enabled/Disabled characteristics. You may also want to support pop-up menus (see page **78**).

Checking and Unchecking Menu Items

Menu options (items) can be checked and unchecked in much the same way as check boxes. Menu items let the user select whether or not to activate a certain feature that affects program execution. The checked state of the menu item is toggled every time the item is selected (clicked). Since this is not done automatically by Windows, it must be done at runtime in your programs using the CHECKMENUITEM function (see also the *Functions and Messages* online Help file).

Note For information about displaying a check mark on a menu item at design time using the Menu Editor dialog box, see page **13**.

A menu item is checked with the following code:

```
INITIALIZE MENU-FLAGS.
SET MF-BYCOMMAND MF-CHECKED TO TRUE.
CALL CHECKMENUITEM USING WIN-RETURN MENU-H ITEM-ID MENU-FLAGS.
```

MENU-FLAGS is a collection of options that affect menus. By first initializing MENU-FLAGS, all the options are unset so that options can be selected.

MF-BYCOMMAND supplies the ID of the menu item to be checked.

MF-CHECKED indicates that the menu item should be checked.

WIN-RETURN returns 1 if the menu item was already checked, 0 if it was not.

MENU-H is the handle of the menu containing the item to be checked. If the item is on a pop-up menu, MENU-H should be the handle of the pop-up menu, rather than the top-level menu.

ITEM-ID is the ID number of the item to check.

The menu item is unchecked in the same manner, but MF-UNCHECKED is used in place of MF-CHECKED.

Enabling and Disabling Menu Items

Menu options (items) can be enabled and disabled at runtime in the same manner as controls, although this is not done with an Enabled property, but rather by using the ENABLEMENUITEM function. (For more information, see the *Functions and Messages* online Help file.)

Note For information about enabling or disabling a menu item at design time using the Menu Editor dialog box, see page .

A menu item is disabled with the following code:

```
INITIALIZE MENU-FLAGS.
SET MF-BYCOMMAND MF-DISABLED MF-GRAYED TO TRUE.
CALL ENABLEMENUITEM USING WIN-RETURN MENU-H ITEM-ID MENU-FLAGS.
```

Most of the parameters for the ENABLEMENUITEM are the same as defined for the CHECKMENUITEM function, described in the previous section. MF-DISABLED and MF-GRAYED are new options that respectively disable and gray out the option. Disabling the option does not automatically gray it out as is the case with controls. Graying must be explicitly requested with the MF-GRAYED option.

To enable the menu item, use the following code:

```
INITIALIZE MENU-FLAGS.
SET MF-BYCOMMAND MF-ENABLED TO TRUE.
CALL ENABLEMENUITEM USING WIN-RETURN MENU-H ITEM-ID MENU-FLAGS.
```

Notice that it is not necessary to specify "un-gray" when the option is enabled. That characteristic is enabled by default. The following code causes the option to be grayed out even when it is enabled:

```
INITIALIZE MENU-FLAGS.
SET MF-BYCOMMAND MF-ENABLED MF-GRAYED TO TRUE.
CALL ENABLEMENUITEM USING WIN-RETURN MENU-H ITEM-ID MENU-FLAGS.
```

This behavior would, however, be unlike the expected Windows behavior.

Popping Up Menus

One interesting technique that can be used with menus is to have the program pop up a menu on the display without the user having selected it from the top-level menu. This type of menu is usually referred to as a context-sensitive pop-up menu. Such menus provide an efficient way to access frequently used commands without the need to navigate a menu bar. They also can include commands that logically apply to the limited context of the selected object. For example, when input focus moves to a customer number field, the program could pop up a menu listing functions related to customer number entry and

place the menu next to the field. The user can select an option from the menu. By clicking outside the pop-up menu (or a specified area), the menu can be dismissed. If the user selects an option from the menu, the Click event associated with that menu option is triggered.

Use the TRACKPOPUPMENU function (see also the *Functions and Messages* online Help file) to accomplish this feature. A call to the TRACKPOPUPMENU function appears as follows:

```
CALL TRACKPOPUPMENU USING WIN-RETURN MENU-H 0 X Y 0 WND-H RECT
```

WIN-RETURN returns 1 if the menu was displayed, 0 if it was not.

MENU-H is the handle of the pop-up menu to display. This cannot be the handle of the top-level menu.

The two 0s are unused parameters for some future functionality.

X and Y are the pixel coordinates at which the top left corner of the menu should be displayed. These coordinates are relative to the entire screen, not the form. You may need to use the CLIENTTOSCREEN function (described in the *Functions and Messages* online Help file) to help you calculate this position.

RECT is an optional parameter. By default, Windows erases the pop-up menu if the user clicks outside the menu. This behavior can be achieved by passing 0 for this parameter instead of RECT. However, RECT can be filled with values and passed to define a specific area of the screen the user should be allowed to click without erasing the menu. This action overrides the default behavior.

Chapter 5: Debugging

Cobol-WOW makes Windows programming fairly straightforward, but as your application grows in complexity, you will need to test your program and debug your source code. This chapter discusses three different approaches to debugging a Windows-based application created with Cobol-WOW:

- Debugging with COBOL DISPLAY Statements

- Debugging with the RM/COBOL Interactive Debugger

- Debugging with CodeWatch (Liant's standalone source-level debugger)

Note It is possible to enable messages that aid in debugging a Cobol-WOW application at runtime by adding the following entry to the **cblwow.ini** file (see page **3**):

```
[WOWRT]
DevelopmentMode=True
```

Debugging with COBOL DISPLAY Statements

The RM/COBOL runtime system creates a window to use for supporting the standard COBOL user interface. Cobol-WOW programs create their own windows, which makes it easy to use the COBOL main window for debugging. The first way you might use this window is by inserting DISPLAY statements in your programs. An example form named SHOWME illustrates this process. The project name is **showme.wpj**; the executable program name is **showme.cbl**. (For procedures on executing the SHOWME program, see page **82**.)

The SHOWME form contains a number of different controls. Every event associated with every control on the form, as well as the form itself, has a DISPLAY statement associated with it. When you run the example, you will see the SHOWME form and the COBOL main window displayed. As you use the form and controls, you will see the result of the DISPLAY statements scrolling by in the COBOL main window.

Since Cobol-WOW programs are event-driven, rather than sequential, you may wonder if certain events occurred, or if certain sections of logic were executed. You cannot assume that because your program is at point C, you passed points A and B. If you insert DISPLAY statements at key points in the program, you will know whether or not those points have been reached.

Unlike traditional COBOL code, event-driven coding associates instructions with a particular event on a particular control. This, in turn, means that if there is a syntax error at compile time, the source of the error might not be immediately apparent. To avoid this problem, compile your project every time you put code against an event. In this way, if there is a compile error, you know exactly where it came from. Additionally, the compilation will save the program, which is a good safeguard against system crashes.

It is good practice to test almost as often as you compile. Text fragments as you develop, rather than waiting to test until you've finished coding the entire program. Testing fragments allows you to isolate errors in logic.

Executing the SHOWME Program

Compile and run the SHOWME program. Just starting the program generates a number of events: Create events, Size events, even Change events when the default text is set.

Then see how many different events you can generate by working with the form and the various controls. Windows reports lots of events, giving you many opportunities to customize the behavior of your programs.

How the SHOWME Program Works

Every available event in the Cobol-WOW Designer for each control has a DISPLAY statement associated with it. When the event occurs, the DISPLAY statement displays the name of the control for which the event occurred and the name of the event.

Debugging with the RM/COBOL Interactive Debugger

If you have been using RM/COBOL very long, you probably have used the Interactive Debugger in the runtime system. While it is unable to display your source code as you debug, it can be a straightforward way of quickly checking out isolated problems if you have a listing file conveniently available.

The Interactive Debugger works better with Cobol-WOW programs than it does with DOS, UNIX, or non-Cobol-WOW Windows programs. Because the Debugger has exclusive access to the COBOL main window, it does not have to share it with the program that is executing. This prevents the Debugger from being limited to operating in a single line or from scrolling the other contents of the display out of view.

The BREAK program (located in the cobolwow\samples folder) demonstrates how the Debugger works with a Cobol-WOW program. The project name is **break.wpj**; the executable program name is **break.cbl**. When you run the BREAK program, you will set a breakpoint on the event-handling code for the Size event in the form. When you reach that breakpoint, you can use the Debugger to display the value of the Height, Left, Top, and Width properties of the form. See the following section for procedures on how to execute the BREAK program.

Executing the BREAK Program

Compile the BREAK program with the L and Y RM/COBOL Compile Command options. Look at the listing file and find the line number of the following line of code:

```
CALL WOWGETPROP USING WIN-RETURN BREAK-H
    "Left" LEFT-VALUE
    "Top"  TOP-VALUE
    "Width" WIDTH-VALUE
    "Height" HEIGHT-VALUE.
```

Remember the line number because you will need it to set a breakpoint.

Next, run the program with the D RM/COBOL Runtime Command option to start the program and enable the Interactive Debugger. The COBOL main window will display and the debug prompt will be displayed in the lower-left corner of the COBOL main window.

Type the following command:

```
B NNNNN
```

where *NNNNN* is the line number of the code identified above, and press **Enter**. This action sets a breakpoint at the specified line. Now, every time the runtime system is ready to execute this line of code, execution will stop and the debug prompt will be displayed.

Type the letter **R** and press **Enter** to run the program. The breakpoint is reached immediately because the breakpoint is associated with the Size event. A Size event is generated when a form is created.

At this point, something very interesting happens. Based on your experience in character-based environments, you would expect the Cobol-WOW Designer window to freeze and no longer respond to user input. This, however, is not the case. Since Windows — not the application program — is controlling the form, the form and controls will continue to respond to user input events. These events are placed in a queue and will wait for the application to retrieve them from the queue.

When the debug prompt is displayed, the COBOL main window is inactive. To work with the Debugger, you must click the mouse in the COBOL main window or press Alt+Tab until you see the COBOL main window listed. Do this now so that the COBOL main window becomes active.

The runtime system is ready to execute the WOWGETPROP function.

Type the letter **S** and press **Enter** to tell the Debugger to step through the execution of this line. The debug prompt is immediately redisplayed and you can examine the values.

To display the value of the fields, type the following Debug commands, pressing Enter after each command:

```
LEFT-VALUE
TOP-VALUE
WIDTH-VALUE
HEIGHT-VALUE
```

Then type **R** and press **Enter** to resume execution of the program. If you happened to resize the form while the runtime system was paused for debugging, you will immediately go back to the breakpoint again. Try resizing the form several times and see how the values change.

How the BREAK Program Works

The BREAK program works by using the COBOL main window for the Debugger and the Cobol-WOW Designer window for the user interface. This is an ideal situation, since the two windows do not interfere with each other.

However, there are two things to remember when debugging in this manner. First, one window is always active, either the Debugger or the form. To work with one or the other, you must click or press Alt+Tab to the desired window. When the Debugger reaches a breakpoint, it will automatically display the debug prompt, but it will not make the COBOL main window active. To type Debug commands, you must make the debug window active.

Secondly, while the runtime system is stopped at a debug breakpoint, the form will continue to respond to user input. You can move and resize a form, press buttons, and even type data into input fields. The COBOL program will not, however, be made aware of any of these events until the runtime execution is resumed. The events are not lost and will then be processed in the order they occurred.

Debugging with CodeWatch

CodeWatch, Liant's standalone source-level debugger, also can be used to debug your program. To use CodeWatch, follow these steps:

1. From the **Options** menu in the Cobol-WOW Designer window, click **Edit Preferences**.

2. On the **Tools** tab of the Preferences dialog box, enter **Y=3** in the Compiler area Command Tail text box.

   The Y=3 setting places both the symbol table and the debug line table (used by CodeWatch to display the source program) in the object file. Additionally, the Y=3 setting includes the allocation map and cross-reference information in the debug line table if the A and/or X Options are also specified.
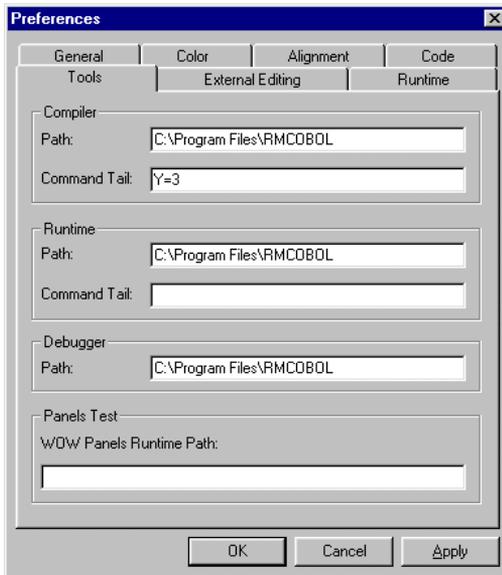
Preferences Dialog Box – Tools Page

3. Set the **L** Option in the Tools tab of the Preferences dialog box in the Compiler Command Tail text box to get an automatic listing file to help identify the area with the problem.

   Note If you set the L Option in this manner, you must remove the entry E L from the Compiler Command Line Arguments text box in the Project Options dialog box. Failing to do so will cause the E L entry to take precedence over the L entry. (To

open the Project Options dialog box for the current project, click **Options** on the **Project** menu.)

4. Click the **Run Debugger** toolbar button to invoke CodeWatch.

Make sure that the CodeWatch environment can find the **wowrt32.dll** that Cobol-WOW needs at runtime. You can do this either by changing the settings in the Windows Registry or by adding the DLL in the CodeWatch wizard as you start it.

Once you have your application loaded in CodeWatch, you can step through it in the normal manner. As you run the program, the additional code that Cobol-WOW has generated is displayed as it is executed. If you do not wish to see this code, set a breakpoint at a relevant place in an event code and disable animation until you reach the breakpoint.

Appendix A: Setting Properties and Events for Intrinsic Controls and Forms

This appendix describes the properties and events of each of the intrinsic controls used in the Cobol-WOW programming system as well as the properties and events for forms.

Manipulating Properties at Runtime

Unless otherwise stated, properties for intrinsic controls and forms can be manipulated at runtime using the WOWSETPROP and WOWGETPROP functions, as described in the examples shown on pages **51** and **52**, respectively.

Intrinsic Controls

Intrinsic controls are part of the Windows operating system. They are always included in the Cobol-WOW Toolbox when you first install Cobol-WOW, unlike ActiveX controls (see Appendix B), which can be removed from or added to the Toolbox. During design time, intrinsic control properties are displayed and modified through the Properties dialog box (see page **8**).

The intrinsic controls include the following:

- Animation Control. Displays an AVI clip. An AVI clip is a series of bitmap frames that run like a movie. Only AVI files without sound can be played using the animation control.

- Bitmap Control. Displays bitmap files. The bitmap control acts like a command button when clicked.

- Check Box Control. Displays a Yes/No, True/False, or On/Off option. You can check any number of check boxes on a form at one time.

- Combo Box Control. Combines a text box with a list box. Allows a user to type in a selection or select an item from a drop-down list.

- Command Button Control. Carries out a command or action when a user chooses it.

- Date Time Picker Control. Allows the user to select a date and time, and to display that date-time in the specified format. **Note:** Control is not available in this release.

- Edit Box Control. Provides an area to enter or display text.

- Ellipse Shape. Draws the geometric shape of an ellipse on the form.

- Group Box Control. Provides a visual and functional container for other controls. It is generally used to enclose related controls (usually check boxes or option buttons).

- IP Address Control. Allows the user to enter a numeric address in Internet protocol (IP) format. This control also allows the application to obtain the address in numeric form rather than in text form. **Note:** Control is not available in this release.

- Line Shape. Draws a line on the form.

- List Box Control. Displays a list of choices from which the user can select one or more items.

- Month Calendar Control. Displays a monthly calendar. The calendar can display one or more months at a time. **Note:** Control is not available in this release.

- Option Button Control. Presents mutually exclusive options in an option control. Option buttons are usually used with the group box control to form groups where only one of the listed buttons can be selected at one time.

- Progress Bar Control. Displays a pattern of blocks that show the status of a long operation.

- Rectangle Shape. Draws the geometric shape of a rectangle on the form.

- Rounded Rectangle Shape. Draws the geometric shape of a rectangle with rounded corners on the form.

- Scroll Bar Controls. Allow a user to add scroll bars (horizontal and/or vertical) to controls that do not automatically provide them. (These are not the same as the built-in scroll bars that are found with many controls.)

- Static Text Control. Displays text, such as titles or captions, in regular outlines or filled rectangles, that the user cannot interact with or modify.

- Status Bar Control. Displays status information in a horizontal window at the bottom of an application window.

- Tab Control. Acts as a container for other controls and places a series of tabs at the top of the container.

- Timer Control. Provides a measured time interval that can be tied to events.

- Toolbar Control. Displays a series of buttons that can be placed at the top and/or bottom of a form.

- Trackbar Control. Displays a window containing a slider and optional tick marks used to select a value or a set of consecutive values in a range.

- Updown Control. Consists of a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control.

Note The description of properties and events for forms begins on page **176**.

Animation Control

An animation control is used to display an AVI clip. An AVI clip is a series of bitmap frames that run like a movie. Only AVI files without sound can be played using the animation control.

To add an animation control to a form, click **Animate** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

At the current time, properties unique to the animation control (AnimationFile, AutoPlay, Center, Play, and Transparent) can be manipulated only in the Cobol-WOW Designer. The runtime functions, WOWGETPROP and WOWSETPROP, will not recognize these properties. Runtime handling of the animation control can be accomplished by using the ACM- messages listed in the Event-Handling Code dialog box.

Properties

*AnimationFile	Enabled	Name	*Transparent
*AutoPlay	Height	*Play	Visible
*Border	Left	TabIndex	Width
*Center	Locked	Top	ZOrder

Events

*Start	*Stop

AnimationFile Property

The AnimationFile property specifies the name of the AVI file containing the animation to play in the control.

AutoPlay Property

The AutoPlay property determines when the animation will begin playing.

The following table lists the possible values of the AutoPlay property:

Value	Description
False	Causes the control to wait until it receives an ACM-PLAY message to begin playing (the default).
True	Causes the animation to begin playing as soon as the control is created (and an animation file is specified).

Border Property

The Border property determines whether or not a border is displayed around the animation.

The following table lists the possible values of the Border property:

Value	Description
False	Does not display a border around the animation.
True	Displays a border around the animation (the default).

Center Property

The Center property determines whether or not the animation is centered in the control.

The following table lists the possible values of the Center property:

Value	Description
False	Does not center the animation (the default).
True	Centers the animation.

Play Property

The Play property determines when the animation starts or stops playing.

The following table lists the possible values of the Play property:

Value	Description
False	Causes the animation to stop playing (the default).
True	Causes the animation to start playing.

Transparent Property

The Transparent property determines whether the animation will be drawn with a transparent background. Currently, this property does not work properly for all animations.

The following table lists the possible values of the Transparent property:

Value	Description
False	Causes the animation to be drawn with an opaque background (the default).
True	Causes the animation to be drawn with a transparent background.

Start Event

The Start event notifies an animation control's parent window that the associated AVI clip has started playing. This notification message is sent in the form of a WM-COMMAND message.

Stop Event

The Stop event notifies an animation control's parent window that the associated AVI clip has stopped playing. This notification message is sent in the form of a WM-COMMAND message.

Bitmap Control

The bitmap control is used to display bitmapped images. The image can be displayed in several ways, including being tiled or scaled to fit the size of the control. A bitmap defines an image or picture as a pattern of dots (or pixels) and has the file extension **.bmp**. Even though Windows does not implement or package a bitmap control, it does provide bitmap handling. Cobol-WOW adds the bitmap control to provide a convenient way to use bitmaps on a form.

To add a bitmap control to a form, click **Bitmap** from the Toolbox.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

BackColor	Enabled	Name	Width
*Bitmap	Height	TabIndex	*Xoffset
*BitmapMode	Left	Top	*Yoffset
*Border	Locked	Visible	ZOrder

Event

Click

Note The bitmap control reports one event, Click, if the mouse is clicked inside the control. Since this control recognizes the Click event, you can use it anywhere you would use a command button. Grouping several bitmap controls together horizontally across the top of the screen, usually within a group box control, allows you to create a toolbar in your application. Unlike command buttons, however, bitmap controls do not appear pushed in when clicked, thereby providing no visual cue that the "button" is being pushed.

Bitmap Property

The Bitmap property specifies the bitmap image that is displayed on the control. (The BitmapMode property setting, described in the following section, determines the bitmap's appearance. If you set the Bitmap property for a form, the bitmap you select is displayed on the background of the form, behind any controls you have placed on the form.)

Note The value of this property must be the complete name of a bitmap file. If a full pathname is specified or if the file is in the working directory, the file will be opened. If the file is not located, Cobol-WOW will attempt to open the bitmap file using each of the directories specified in the RUNPATH environment variable. If the bitmap is not in the working directory or in a directory specified in the RUNPATH environment variable, a pathname is also required.

BitmapMode Property

The BitmapMode property determines how the bitmap is displayed in a control.

The following table list the possible values of the BitmapMode property:

Value	Description
0	Displays the bitmap in its original size (the default). If the bitmap is smaller than the control, the remaining space is filled with the background color. If the bitmap is larger than the control, only the portion of the bitmap that fits inside the control is displayed.
1	Scales bitmap to fit exactly within the control. This may result in some distortion of the bitmap image, especially if the size difference between the bitmap and the control is substantial.
2	Tiles bitmap to fit the control. If BitmapMode is set to Tile, the bitmap, if smaller than the control, is displayed in a tiled pattern multiple times within the control.
3	Sizes the control automatically to fit the specified bitmap exactly.

Note Changing the value of the BitmapMode property to 1, 2, or 3 at design time or runtime will set the values of the Xoffset and Yoffset properties to 0.

Border Property

The Border property determines whether or not a border is displayed around the bitmap.

The following table lists the possible values of the Border property:

Value	Description
False	Does not display a border around the bitmap.
True	Displays a border around the bitmap (the default).

Xoffset Property

The Xoffset property determines how far, in pixels, from the left edge of the control the bitmap is displayed. This value is reset to zero whenever the BitmapMode property settings change.

The Xoffset value must be in the range of 0 to the width of the control.

Yoffset Property

The Yoffset property determines how far, in pixels, from the top of the control the bitmap is displayed. This value is reset to zero whenever the BitmapMode property settings change.

The Yoffset value must be in the range of 0 to the height of the control.

Check Box Control

The check box control displays an option that can be turned on or off. The check box control is similar to the command button, in that the primary method of operation is clicking it. The check box control, however, represents data, not a request for action.

The check box solves a programming situation that has always been challenging: one in which a user must choose between True/False, Yes/No, or On/Off options. Since check boxes work independently of each other, a user can select any number of check boxes at the same time. While these seem like trivial items, creating a character-based implementation that includes validation, good user feedback, and convenient operation are certainly not insignificant. The check box control makes these tasks effortless.

 To add a check box control to a form, click **Check Box** from the Toolbox.

Note If you are working with the check box field/control in an RM/Panels panel library, see page **218**.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

3D	FontItalic	Height	Top
*Alignment	FontName	Left	*Value
*AutoCheck	FontSize	Locked	Visible
BackColor	FontStrikethru	Name	Width
Caption	FontUnderline	TabIndex	ZOrder
Enabled	ForeColor	TabStop	
FontBold	Group	*ThreeState	

Events

Click	KeyDown	KeyUp
GotFocus	KeyPress	LostFocus

Note The user can change the state of a check box in two ways: by clicking with the mouse or by pressing the Spacebar while the check box has input focus. With either method, the Click event for the check box is triggered. You may want to add event-handling code to this event to enable or disable other controls based on the new state of the check box.

Alignment Property

The Alignment property controls the position of the text in a check box control. By default, the caption of a check box displays to the right of the box. The text may be moved to the left of the box with the Alignment property. When using the 3D property, however, the text must always be on the right.

The following table lists the possible values of the Alignment property:

Value	Description
0	Displays text to the right of the check box (the default).
1	Displays text to the left of the check box.

AutoCheck Property

The AutoCheck property determines whether the state of a check box control is automatically changed when clicked.

The following table lists the possible values of the AutoCheck property:

Value	Description
False	Check box will not automatically check or uncheck when clicked.
True	Check box will automatically check or uncheck when clicked (the default).

ThreeState Property

The ThreeState property determines whether a check box control can be cycled through two or three states.

When you create the check box, you assign it a caption that describes the option for which the user is selecting the state (for example, Tax Exempt or Drop Ship). Initially, a check box control has two states, checked and unchecked. These are intuitively On/Off, Yes/No, or True/False selections of whatever the caption describes. The user toggles the check box to the desired state. When the user presses OK, you simply check the state of the button to see what condition to store as data.

You can determine whether you want your check box to have two states or three with the ThreeState property. The third state (grayed) is considered to be no choice made or undefined.

The following table lists the possible values of the ThreeState property:

Value	Description
False	Check box has two states, checked or unchecked (the default).
True	Check box has three states, checked, unchecked and grayed.

Value Property

The Value property determines the state of a check box control.

The following table lists the possible values of the Value property:

Value	Description
0	Check box is not checked (the default).
1	Check box is checked.
2	Check box is grayed (displays only if ThreeState property is set to True).

Combo Box Control

Many times, you may want to combine the list selection capability of a list box with the edit box's ability to type in a value. Alternatively, to save screen space, you may wish to show only a portion of the list box's selections. And, there may be instances when you would like to display the currently selected item in a static edit box area when the entire list is not displayed. The combo box control can satisfy all these conditions since it combines the features of a edit box (also known as an edit field) and a list box. Use this control to give the user the choice of typing in the edit box area or selecting an item from the list portion of the control. Combo boxes can save space on a form.

In addition, if you know how to use a edit box and a list box, you know how to use a combo box. The properties and events available for a combo box are a composite of those present in the edit box and list box controls. The messages you use with a combo box also parallel those that you use with edit boxes and list boxes. These messages, however, begin with a CB- prefix instead of an LB- or EM- prefix.

To add a combo box control to a form, click **Combo Box** from the Toolbox.

Note If you are working with the combo box field/control in an RM/Panels panel library, see page **218**.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

3D	Enabled	FontUnderline	Name	TabIndex
*AutoHScroll	FontBold	ForeColor	*OEMConvert	TabStop
BackColor	FontItalic	Group	ScrollBar	Top
*Count	FontName	Height	*SelText	Visible
*CurSel	FontSize	Left	*Sort	Width
*DisableNoScroll	FontStrikethru	Locked	*Style	ZOrder

Events

Click	*DropDown	GotFocus	KeyPress	LostFocus
DblClick	*EditChange	KeyDown	KeyUp	*NoSpace

Note The DblClick and DropDown and events are not supported if the Style property is set to a value of 0 (Simple, standard combo box). The DropDown event occurs when the user double-clicks the drop-down arrow in the text portion of the drop-down combo box and in the drop-down list box.

AutoHScroll Property

The AutoHScroll property indicates whether the edit portion of a combo box control will automatically scroll horizontally as text is entered. If the value of this property is set to 0, the user will not be allowed to enter more text than fits within the width of the control.

The following table lists the possible values of the AutoHScroll property:

Value	Description
False	Disables horizontal scrolling.
True	Enables horizontal scrolling of text when typed (the default).

Count Property

The Count property is a runtime-only property that lets you determine how many items are included in the list box portion of the combo box. To get the number of items in the list box:

```
CALL WOWGETPROP USING WIN-RETURN MYCOMBO-H "COUNT" COUNT-FIELD.
```

CurSel Property

The CurSel property is a runtime-only property that represents the current selection in the list box portion of the combo box. This value can be queried to determine which item in the list box is selected, or set to move the selection to a different item.

DisableNoScroll Property

The DisableNoScroll property determines whether a scroll bar is displayed when the list box portion of a combo box control is not completely full.

The following table lists the possible values of the DisableNoScroll property:

Value	Description
False	Scroll bar disappears if combo box is not full (the default).
True	Scroll bar is disabled if combo box is not full.

OEMConvert Property

The OEMConvert property converts characters entered in the edit box portion of a combo box control from the ANSI character set to the OEM character set and then back to ANSI. Use this property for combo box controls that are used to enter filenames. When a character is converted from the ANSI character set to the OEM character set and back to ANSI, the resulting character is not always the same as the original character; however, subsequent conversions from ANSI to OEM to ANSI do result in the same character.

The following table lists the possible values of the OEMConvert property:

Value	Description
False	The characters are not converted (the default).
True	The characters are converted from ANSI to OEM and back to ANSI.

SelText Property

The SelText property is a runtime-only property that lets you retrieve the text of the currently selected list box item. If no item is selected, the value returned is space.

Sort Property

The Sort property determines whether the entries in a combo box control are automatically sorted.

The following table lists the possible values of the Sort property:

Value	Description
False	Entries are not sorted.
True	Entries are sorted (the default).

Style Property

The Style property determines what type of combo box is created. The three types of combo boxes are specified with the Style property. The possible values for this property include simple (standard) combo box, drop-down combo box, and drop-down list box.

A standard combo box always displays an edit box and list box. A drop-down combo box always displays an edit box, but only displays the list box when the drop-down arrow displayed beside the edit box is clicked. A drop-down list box always displays a static edit box control containing the current selection, but, like the drop-down combo box, only displays the list box when the drop-down arrow beside the static text control is clicked.

You might question why the drop-down list box is a style for combo boxes but is not a style for list boxes. This is the way Windows built this control; it should not cause you any problems. Windows simply implements these three styles as one control because they all combine two types of controls into one.

You work with the list box portion of a combo box in exactly the same way you work with a list box. You use messages with a CB- prefix and supply the combo box handle. Windows knows what part of the combo box to change.

For the edit box portion, work with the combo box properties, events, and messages as you would an edit box remembering to use the CB-prefix.

The following table lists the possible values of the Style property:

Value	Description
0	Simple (standard combo box). The edit box (edit field) and list box portions are always displayed.
1	Drop-down combo box. The edit box portion is always displayed but the list box area is only displayed when the drop-down arrow is clicked.
2	Drop-down list box. The edit box portion is always displayed, however, it only displays the value of the selection. The edit box portion will not accept user input. The list box portion is only displayed when the drop-down arrow is clicked.

DropDown Event

The DropDown event occurs when the user double-clicks the left mouse button on the drop-down arrow in the edit box portion of the drop-down combo box and drop-down list box.

Note This event is not supported if the Style property value is set to a value of 0 (Simple, standard combo box).

EditChange Event

The EditChange event occurs whenever the text displayed in the edit box portion of the combo box is changed.

NoSpace Event

The NoSpace event occurs when Windows cannot allocate enough internal space to store the contents of the combo box.

Command Button Control

The command button (also known as push button) control causes an action to occur when the user either clicks the button or presses a key.

The command button control is simple to use for both the user and the developer. When you place a command button on a form, the user can perform one action: push. Unlike other controls, the command button does not represent any data. It represents a request for action. When a command button is pushed, an action is carried out immediately.

To add a command button control to a form, click **Command Button** from the Toolbox.

Note If you are working with the command button field/control in an RM/Panels panel library, see page **220**.

Cobol-WOW offers a user several ways to push a command button:

- Clicking it with the mouse.

- Pressing the Spacebar when the command button has input focus.

- Pressing an accelerator key for the command button while any control on the form has input focus.

- Pressing the Enter key while any control on the form has input focus.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

*Accelerator	FontItalic	Height	Top
*Bitmap	FontName	Left	Visible
Caption	FontSize	Locked	Width
*Default	FontStrikethru	Name	ZOrder
Enabled	FontUnderline	TabIndex	
FontBold	Group	TabStop	

Events

Click	KeyDown	KeyUp
GotFocus	KeyPress	LostFocus

Accelerator Property

The Accelerator property determines what key, if any, should simulate the pressing of the command button. This property cannot be modified or retrieved at runtime. An accelerator key is defined for the command button by selecting one of the available keys for the Accelerator property listed in the Properties dialog box. Function keys are acceptable as accelerator keys. You must include the name of the accelerator key in the text of the command button so that the user knows it is available.

Bitmap Property

The Bitmap property specifies the bitmap image that is displayed on the command button control. (If you set the Bitmap property for a form, the bitmap you select is displayed on the background of the form, behind any controls you have placed on the form.)

Note The value of this property must be the complete name of a bitmap file. If a full pathname is specified or if the file is in the working directory, the file will be opened. If the file is not located, Cobol-WOW will attempt to open the bitmap file using each of the directories specified in the RUNPATH environment variable. If the bitmap is not in the working directory or in a directory specified in the RUNPATH environment variable, a pathname is also required.

Default Property

The Default property indicates that a command button control should be pressed when the Enter (or Return) key is pressed while input focus is anywhere on the form. A command button with the Default property set to True is displayed with a heavy border. Only one command button on a form should be set with the Default property.

Note The value of the Default property cannot be set at runtime. The value can, however, be retrieved at runtime.

The following table lists the possible values of the Default property:

Value	Description
False	Button is not a default button (the default).
True	Button is a default button.

Date Time Picker Control (Not Available in this Release)

The date time picker control allows the user to select a date and time, and to display that date-time in the specified format. An embedded month calendar control (see page **126**) displays a monthly calendar.

The date time picker control is based on the Gregorian calendar, which was introduced in 1753. It will not calculate dates that are consistent with the Julian calendar that was in use prior to 1753.

To add a date time picker control to a form, click **Date Time Picker** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

Enabled	*Format	*MCFontBold	*RightAlign	*UpDown
FontBold	Height	*MCFontItalic	*ShortDateCentury Format	Visible
FontItalic	Left	*MCFontName	*ShowNone	Width
FontName	Locked	*MCFontSize	TabIndex	ZOrder
FontSize	*LongDateFormat	*MCFontStrikeThru	TabStop	
FontStrikethru	MCColor	*MCFontUnderline	*TimeFormat	
FontUnderline	MCColorIndex	Name	Top	

Event

*Change

Format Property

The Format property determines the date-time format in which the date is displayed. The date-time format is based on the user's regional settings in their operating system.

Date and time format elements will be replaced by the actual date and time. They are defined by the following groups of characters:

Value	Description
"d"	The one- or two-digit day.
"dd"	The two-digit day. Single-digit day values are preceded by a zero.
"ddd"	The three-character weekday abbreviation.
"dddd"	The full weekday name.
"h"	The one- or two-digit hour in 12-hour format.
"hh"	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
"H"	The one- or two-digit hour in 24-hour format.
"HH"	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
"m"	The one- or two-digit minute.
"MM"	The two-digit minute. Single-digit values are preceded by a zero.
"MMM"	The three-character month abbreviation.
"MMMM"	The full month name.
"t"	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
"tt"	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").
"yy"	The last two digits of the year (that is, 1996 would be displayed as "96").
"yyyy"	The full year (that is, 1996 would be displayed as "1996").

To make the information more readable, you can add body text to the format string by enclosing it in single quotes. Spaces and punctuation marks do not need to be quoted.

Note Non-format characters that are not delimited by single quotes will result in unpredictable display by the date time picker control.

For example, to display the current date with the format "'Today is: 04:22:31 Tuesday Mar 23, 1996", the format string is "'Today is: 'hh':'m':'s dddd MMM dd', 'yyyy". To include a single quote in your body text, use two consecutive single quotes. For example,

"'Don"t forget' MMM dd',' yyyy" produces output that looks like: Don't forget Mar 23, 1996. It is not necessary to use quotes with the comma, so "'Don"t forget' MMM dd, yyyy" is also valid, and produces the same output.

LongDateFormat Property

The LongDateFormat property, when set to a value of True, causes the date to display in day, month, date, year format. For example: "Friday, April 19, 2002".

The following table lists the possible values of the LongDateFormat property:

Value	Description
False	The date is displayed in short date format, for example, "4/19/02" (the default).
True	The date is displayed in long date format, for example, "Friday, April 19, 2002".

MCFontBold Property

The MCFontBold property determines whether the associated text for the month calendar is displayed in bold font format.

The following table lists the possible values of the MCFontBold property:

Value	Description
False	Text is not displayed bold (the default).
True	Text is displayed bold.

MCFontItalic Property

The MCFontItalic property determines whether the associated text of the month calendar is displayed in italic font format.

The following table lists the possible values of the MCFontItalic property:

Value	Description
False	Text is not displayed in italics (the default).
True	Text is displayed in italics.

MCFontName Property

The MCFontName property determines the font used to display text in the month calendar. The font specified must be present on the system.

MCFontSize Property

The MCFontSize property determines the size of the font to be used for text displayed in the month calendar. The size specified must be supported by the font. If the size is not supported by the font, the system will substitute the nearest supported value.

MCFontStrikeThru Property

The MCFontStrikeThru property determines whether the associated text for the month calendar is displayed in a strikethrough font style.

The following table lists the possible values of the MCFontStrikethru property:

Value	Description
False	No strikeout is used (the default).
True	Strikeout is used.

MCFontUnderline Property

The MCFontUnderline property determines whether the associated text for the month calendar is displayed in an underlined font format.

The following table lists the possible values of the MCFontUnderline property:

Value	Description
False	Text is not underlined (the default).
True	Text is underlined.

RightAlign Property

The RightAlign property determines whether the drop-down month calendar will be right-aligned or left-aligned with the date time picker control.

The following table lists the possible values of the RightAlign property:

Value	Description
False	The drop-down month calendar will be left-aligned with the control (the default).
True	The drop-down month calendar will be right-aligned with the control.

ShortDateCenturyFormat Property

The ShortDateCenturyFormat property, when set to a value of True, causes the date to display in the MM/DD/YYYY format. For example: "4/19/2002".

The following table lists the possible values of the ShortDateCenturyFormat property:

Value	Description
False	The date is displayed in short date format, for example, "4/19/02" (the default).
True	The date is displayed in short date century format, for example, "4/19/2002".

ShowNone Property

The ShowNone property determines whether the control displays a check box.

The following table lists the possible values of the ShowNone property:

Value	Description
False	No check box is displayed (the default).
True	A check box is displayed.

TimeFormat Property

The TimeFormat property determines whether the time will display instead of the date. When set to a value of True, the TimeFormat property causes the time to display in HH/MM/SS AM or PM format. For example: "5:31:42 PM".

The following table lists the possible values of the TimeFormat property:

Value	Description
False	The time is not displayed (the default).
True	The time is displayed in HH/MM/SS AM or PM format. For example, "5:31:42 PM".

UpDown Property

The UpDown property determines whether the control displays an arrow button. If the user clicks the arrow button, an embedded month calendar control (see page **126**) drops down. The user can select a specific date by clicking an area of the calendar.

The following table lists the possible values of the UpDown property:

Value	Description
False	An arrow button is displayed (the default).
True	An arrow button is not displayed.

Change Event

The Change event occurs when a change has occurred within the date time picker control.

Edit Box Control

The edit box control provides an area to input or display text. This control replaces the COBOL ACCEPT statement. The user can enter any type of alphanumeric data in an edit box, including numeric data. Because no formatting is provided, numbers are entered in the same manner as text. (The use of edit box controls is illustrated in Chapter 2, *Tutorial*.)

`ab` To add an edit box control to a form, click **Edit Box** from the Toolbox.

Note If you are working with the edit box field/control in an RM/Panels panel library, see page **223**.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

3D	FontBold	Height	*Password	Visible
*Alignment	FontItalic	Left	*PasswordChar	*WantReturn
*AutoHScroll	FontName	Locked	*ReadOnly	Width
*AutoVScroll	FontSize	*MaxChars	*ScrollBars	ZOrder
BackColor	FontStrikethru	*Multiline	TabIndex	
*Border	FontUnderline	Name	TabStop	
*Case	ForeColor	*NoHideSel	*Text	
Enabled	Group	*OEMConvert	Top	

Events

*Change	*HScroll	KeyPress	LostFocus	*NoSpace
GotFocus	KeyDown	KeyUp	*MaxText	*VScroll

Alignment Property

The Alignment property determines how text is positioned in an edit box control.

Note The Alignment property has an affect only when the Multiline property (see page **112**) has a value of 1 (True).

The following table lists the possible values of the Alignment property:

Value	Description
0	Normal – Performs no justification (the default).
1	Left justifies all text.
2	Center justifies all text.
3	Right justifies all text.

AutoHScroll Property

The AutoHScroll property indicates whether an edit box control will automatically scroll horizontally as text is entered. If the value of this property is set to False, the user will not be allowed to enter more text than fits within the width of the control.

The following table lists the possible values of the AutoHScroll property:

Value	Description
False	Disables horizontal scrolling.
True	Enables horizontal scrolling of text when typed (the default).

AutoVScroll Property

The AutoVScroll property determines whether an edit box control will scroll vertically as text is entered. If the value of AutoVScroll property is set to False, the user will not be allowed to enter more text than the control will display.

Note The AutoVScroll property has an affect only when the Multiline property (see page **112**) has a value of True.

The following table lists the possible values of the AutoVScroll property:

Value	Description
False	Disables vertical scrolling of text when typed (the default).
True	Enables vertical scrolling of text when typed.

Border Property

The Border property determines whether a border is displayed around an edit box control.

The following table lists the possible values of the Border property:

Value	Description
False	A border is not displayed (the default).
True	A border is displayed.

Case Property

The Case property determines the case conversion of alphabetic characters entered into an edit box control.

The following table lists the possible values of the Case property:

Value	Description
0	Mixed – text case is not altered; accepted as typed (the default).
1	Converts all text to lowercase.
2	Converts all text to uppercase.

MaxChars Property

The MaxChars property determines how many characters can be entered into an edit box control. A value of 0 will not set any limit.

Multiline Property

The Multiline property determines whether an edit box control should support single or multiple lines of text.

The following table lists the possible values of the Multiline property:

Value	Description
False	Control has only one line of text (the default).
True	Control can have multiple lines of text.

NoHideSel Property

The NoHideSel property determines whether the selected text remains highlighted when an edit box control loses the input focus.

The following table lists the possible values of the NoHideSel property:

Value	Description
False	Selected text does not remain highlighted when the edit box control loses input focus (the default).
True	Selected text remains highlighted when the edit box control loses input focus.

OEMConvert Property

The OEMConvert property converts characters entered in an edit box control from the ANSI character set to the OEM character set and then back to ANSI.

This property should be used for edit box controls that are used to enter filenames. When a character is converted from the ANSI character set to the OEM character set and back to ANSI, the resulting character is not always the same as the original character; however, subsequent conversions from ANSI to OEM to ANSI do result in the same character.

The following table lists the possible values of the OEMConvert property:

Value	Description
False	The characters are not converted (the default).
True	The characters are converted from ANSI to OEM and back to ANSI.

Password Property

The Password property determines whether the text of an edit box control or the password character is displayed (see "PasswordChar Property" in the following section).

The following table lists the possible values of the Password property:

Value	Description
False	Text of the control is displayed (the default).
True	The password character is displayed instead of the text.

PasswordChar Property

The PasswordChar property determines the character that is displayed if an edit box control has the Password property set.

Set the value of the PasswordChar property with any alphanumeric character, including space.

ReadOnly Property

The ReadOnly property determines whether the contents of an edit box control can be modified by the user.

The following table lists the possible values of the ReadOnly property:

Value	Description
False	Contents may be modified (the default).
True	Contents may not be modified.

ScrollBars Property

The ScrollBars property determines whether one or more scroll bars are included in an edit box control.

Note Vertical scroll bars should only be used with edit box controls when the Multiline property (see page **112**) is set to a value of 1 (True).

The following table lists the possible values of the ScrollBars property:

Value	Description
0	No scroll bars are added (the default).
1	A vertical scroll bar is added.
2	A horizontal scroll bar is added.
3	Both vertical and horizontal scroll bars are added.

Text Property

The Text property specifies the text associated with an edit box control.

Set the value of the Text property with any alphanumeric character, including space.

WantReturn Property

The WantReturn property, used in combination with the Multiline property, specifies that a carriage return be inserted when the user presses the Enter (or Return) key while entering text into a multi-line edit box control in a dialog box. When the user presses Enter in a multi-line edit box control that omits this property, the dialog box's default command button is pressed.

The following table lists the possible values of the WantReturn property:

Value	Description
False	A carriage return is not inserted when the user presses the Enter key during text entry (the default).
True	A carriage return is inserted when the user presses the Enter key during text entry.

Change Event

The Change event occurs when the value of the text in an edit box control changes. Any of the following actions will cause this event to occur:

- A character is typed in the edit box control.

- The WOWSETPROP function (see page **52**) is used to set the text.

- The edit box control is created with a text value assigned in the Designer.

HScroll Event

The HScroll event occurs when the user clicks the horizontal scroll bar for the edit box control.

MaxText Event

The MaxText event occurs when the user attempts to enter more characters than the edit box control will allow. This event only occurs if the AutoHScroll property is not set, or a MaxChars property is not equal to 0.

NoSpace Event

The NoSpace event occurs when the internal Windows memory used to store the text of the edit box control has been depleted.

VScroll Event

The VScroll event occurs when the user clicks the vertical scroll bar for the edit box control.

Ellipse Shape

The ellipse shape is used to draw the geometric shape of an ellipse on the form. A 32-bit Windows-based application uses filled shapes in a variety of ways. Spreadsheet applications, for example, use filled shapes to construct charts and graphs.

Technically, an ellipse is a closed curve defined by two fixed points such that the sum of the distances from any point on the curve to the two fixed points is constant. When calling ellipse, an application supplies the coordinates of the upper-left and lower-right corners of the ellipse's bounding rectangle. A bounding rectangle is the smallest rectangle completely surrounding the ellipse. When the system draws the ellipse, it excludes the right and lower sides if no world transformations are set. Therefore, for any rectangle measuring x units wide by y units high, the associated ellipse measures x–1 units wide by y–1 units high.

To add an ellipse shape control to a form, click **Ellipse** from the Toolbox.

Note This shape is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this shape will not be displayed on the Cobol-WOW Toolbox.

All of the properties for this shape are listed in the following table. For detailed information on these properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

BackBrushHatch	Fill	Left	PenSize	Top
BackBrushStyle	ForeColor	Locked	PenStyle	Width
BackColor	Height	Name	TabIndex	ZOrder

Note Because the ellipse shape allows no user interaction, no events are associated with it.

Group Box Control

The group box control (sometimes called a group box control) is a specialized box that is used to group other controls, such as check boxes and option (or radio) buttons.

The group box control cannot be modified or operated on by the user. Windows implements this control in much the same way as check boxes and option buttons, and it is commonly used to group these types of controls.

There is no need to retrieve the text of a group box, and situations in which you would want to change its text are hard to imagine, but possible. To change the text of a group box control at runtime with the WOWSETPROP function:

```
CALL WOWSETPROP USING WIN-RETURN CTL-H "CAPTION" NEW-TEXT.
```

CTL-H is the handle of the group box. "CAPTION" is the name of the property. NEW-TEXT is the new text of the control.

To add a group box control to a form, click **Group Box** from the Toolbox.

Note If you are working with the group box field/control in an RM/Panels panel library, see page **224**.

All of the properties for this control are listed in the following table. Note that none of the properties for this control are unique. For information on the properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

3D	FontItalic	ForeColor	Name	Width
BackColor	FontName	Group	TabIndex	ZOrder
Caption	FontSize	Height	TabStop	
Enabled	FontStrikethru	Left	Top	
FontBold	FontUnderline	Locked	Visible	

Note Because the group box control allows no user interaction, no events are associated with it.

IP Address Control (Not Available in this Release)

The IP address control allows the user to enter a numeric address in Internet protocol (IP) format. This format consists of four three-digit fields. Each field is treated individually. The field numbers are zero-based and proceed from left to right. This control also allows the application to obtain the address in numeric form rather than in text form.

The IP address control allows only numeric text to be entered in each of the fields. Once three digits have been entered in a given field, keyboard focus is automatically moved to the next field. If filling the entire field is not required by the application, the user can enter fewer than three digits. For example, if the field should only contain 21, typing 21 and pressing the Right Arrow key will take the user to the next field.

To add an IP address control to a form, click **IP Address** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

BackColor	FontBold	FontUnderline	Name	Visible
Enabled	FontItalic	ForeColor	TabIndex	Width
*FieldIndex	FontName	Height	TabStop	ZOrder
*FieldMax	FontSize	Left	Top	
*FieldMin	FontStrikethru	Locked	*Value	

Events

*Change	GotFocus	LostFocus

FieldIndex Property

The FieldIndex property controls the currently selected field of the IP address. The value is a zero-based index to the four three-digit address fields, where 0 indicates the first field, 1 indicates the second field, 2 indicates the third field, and 3 indicates the fourth field.

FieldMax Property

The FieldMax property specifies the maximum range for address field. The value of the FieldIndex property (see above) determines which field is affected. The possible range for each field is 0 to 255, but the range can be set to any values between those limits. The default value is 255.

FieldMin Property

The FieldMin property specifies the minimum range for address field. The value of the FieldIndex property (see above) determines which field is affected. The possible range for each field is 0 to 255, but the range can be set to any values between those limits. The default value is 0.

Value Property

The Value property specifies the value of the IP address and should be in the range specified by the settings of the FieldMin and FieldMax properties (described above).

Change Event

The Change event occurs when the value of the address fields in an IP address control changes.

Line Shape

The line shape is used to draw a line on the form.

To add a line shape control to a form, click **Line** from the Toolbox.

Note This shape is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this shape will not be displayed on the Cobol-WOW Toolbox.

All of the properties for this shape are listed in the following table. For detailed information on these properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

BackBrushHatch	Fill	Left	PenSize	Top
BackBrushStyle	ForeColor	Locked	PenStyle	Width
BackColor	Height	Name	TabIndex	ZOrder

Note Because the line shape allows no user interaction, no events are associated with it.

List Box Control

The list box control allows the selection of one or several items from a list of items. It is a simple, yet versatile control. You load a list box with items at runtime. When enabled, the user can select an item by clicking with the mouse or moving the selection bar with the arrow keys. You send the list box a message to find out which item in the list box is selected. For more information on using functions and messages with list boxes and the procedures on how to use list boxes, see page **124**.

To add a list box control to a form, click **List Box** from the Toolbox.

Note If you are working with the list box field/control in an RM/Panels panel library, see page **226**.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

3D	*ExtendedSel	Group	ScrollBar	Visible
BackColor	FontBold	Height	*SelText	*WantKeyboard
*Border	FontItalic	Left	*Sort	Width
*ColumnWidth	FontName	Locked	*Standard	ZOrder
*Count	FontSize	*MultipleSel	TabIndex	
*CurSel	FontStrikethru	Name	TabStop	
*DisableNoScroll	FontUnderline	*NoIntegralHeight	Top	
Enabled	ForeColor	*NoRedraw	*UseTabStops	

Events

Click	GotFocus	KeyPress	LostFocus
DblClick	KeyDown	KeyUp	

Note The event to which you are most likely to respond with a list box is the Click event. This is the event that occurs whenever a selection changes, either by mouse click or keyboard press, or when the Standard property is set to True. However, list boxes generally do not take action on a selection change. A DblClick event occurs when a list box item is double-clicked. This event is often expected to trigger some immediate program response.

Border Property

The Border property determines whether a border is displayed around a list box control.

The following table lists the possible values of the Border property:

Value	Description
False	A border is not displayed.
True	A border is displayed (the default).

ColumnWidth Property

The ColumnWidth property determines the width, in pixels, of the columns in a list box control with multiple columns. If you specify a non-zero value for the ColumnWidth property, the list box will display multiple columns.

Set the ColumnWidth property with any positive value greater than 0 but less than the value specified in the Width property for the list box control.

Count Property

The Count property is a runtime-only property that lets you determine how many items are contained in the list box.

Note This property can only be retrieved, not set, at runtime.

CurSel Property

The CurSel property is a runtime-only property that represents the current selection in the list box. This value can be queried to determine which item in the list box is selected, or set to move the selection to a different item. If no item is selected, this property has the value LB-ERR.

DisableNoScroll Property

The DisableNoScroll property determines whether a scroll bar is displayed when a list box control is not completely full.

The following table lists the possible values of the DisableNoScroll property:

Value	Description
False	Scroll bar disappears if list box is not full (the default).
True	Scroll bar is disabled if list box is not full.

ExtendedSel Property

The ExtendedSel property allows selections in a multiple selection list box control by using the mouse and the Shift key.

The following table lists the possible values of the ExtendedSel property:

Value	Description
False	No extended selection (the default).
True	Extended selection allowed.

MultipleSel Property

The MultipleSel property allows more than one item in a list box control to be selected.

The following table lists the possible values of the MultipleSel property:

Value	Description
False	No multiple selection allowed (the default).
True	Multiple selection allowed.

NoIntegralHeight Property

The NoIntegralHeight property determines whether the height of a list box control is adjusted to contain an even number of items.

The following table lists the possible values of the NoIntegralHeight property:

Value	Description
False	List box height is adjusted (the default).
True	List box height is not adjusted.

NoRedraw Property

The NoRedraw property allows a list box control to be created without updating the screen when entries are loaded. After entries are loaded, the property can be changed to update the screen display.

Note The value of this property cannot be retrieved at runtime. The value can, however, be set at runtime with WOWSETPROP (see page **52**).

The following table lists the possible values of the NoRedraw property:

Value	Description
False	List box is redrawn (the default).
True	List box is not redrawn.

SelText Property

The SelText property is a runtime-only property that lets you retrieve the text of the currently selected list box item. If no item is selected, the value returned is space.

Note This property can only be retrieved, not set, at runtime.

Sort Property

The Sort property determines whether the entries in a list box control are automatically sorted.

The following table lists the possible values of the Sort property:

Value	Description
False	Entries are not sorted.
True	Entries are sorted (the default).

Standard Property

The Standard property, when turned on, causes a list box control to be sorted and the Click event to occur every time the selection changes.

The following table lists the possible values of the Standard property:

Value	Description
False	No sorting or Click event.
True	Entries sorted and Click event on selection (the default).

UseTabStops Property

The UseTabStops property determines whether tab characters are interpreted as a spacing technique by a list box control.

The following table lists the possible values of the UseTabStops property:

Value	Description
False	Tabs are not expanded (the default).
True	Tabs are expanded.

WantKeyboard Property

The WantKeyboard property determines whether keystroke events are reported to the form containing a list box control.

The following table lists the possible values of the WantKeyboard property:

Value	Description
False	Keystroke events are not reported to the form (the default).
True	Keystrokes are reported to the form.

Using Functions and Messages with List Boxes

There are several functions and many messages that you can use with a list box. The functions that deal with adding and removing items in a list box are WOWADDITEM, WOWREMOVEITEM, and WOWCLEAR. Respectively, these functions add an item to a list box, remove an item from a list box, and remove all items from a list box. The messages you can use with a list box are too numerous to list here, but each begins with the prefix LB-. Comprehensive information about messages can be found in the *Functions and Messages* online Help file. We recommend that you take a few minutes and browse through these topics to get an idea of the kinds of capabilities that messages can provide.

Using a List Box

The following sections outline the basic procedures involved in using a list box.

Loading the List Box

The list box is loaded at runtime, one item at a time, with the WOWADDITEM function. This function can be used to insert an item in a list box at a specific position or to append

it to the end of the list. To add an item to the list box, use the WOWADDITEM function as follows:

```
CALL WOWADDITEM USING WIN-RETURN CTL-H NEW-ITEM INDEX.
```

WIN-RETURN returns 0 if the function call is successful. CTL-H is the handle of the list box. NEW-ITEM must be an alphanumeric data item or literal that contains the item to be added to the list box. INDEX is an optional, zero relative index of the position at which the item should be added.

Operating the List Box

Once the list box is loaded, Windows takes care of the operation of the list box. If the Standard property is set to True, the Click event is executed every time the user makes a selection. Otherwise, no event is associated with making a selection. In general, no action is taken when a selection is made, but the user should press a command button or select a menu option to take an action. In some cases, you may want to display information related to the selection in another part of the form as the selection changes.

Determining the Selection

At some point, you will want to determine what selection was made in the list box. This is accomplished by checking the value of the list box's CurSel property as follows:

```
CALL WOWGETPROP USING WIN-RETURN CTL-H "CURSEL" SEL-VALUE.
```

SEL-VALUE returns the 0 relative index of the selected item. CTL-H is the handle of the list box. If there is no selection, SEL-VALUE will equal LB-ERR. Note that LB-ERR is a -1 value, so SEL-VALUE must be a signed field to properly return this value.

Finding an Item

To find a specific item in a list box, use the LB-FINDSTRING or LB-FINDSTRINGEXACT message. The LB-FINDSTRING message finds the first entry in the list box that begins with the specified value. The LB-FINDSTRINGEXACT message finds the first entry in the list box that exactly matches the specified value. The messages are sent in the same manner:

```
CALL SENDMESSAGE USING WIN-RETURN CTL-H LB-FINDSTRING
     START-POS SEARCH-VALUE.
```

WIN-RETURN is the relative position of the item if found, or LB-ERR if an item is not found. CTL-H is the handle of the list box. START-POS is the zero-relative position at which the search should begin. SEARCH-VALUE is the alphanumeric literal or data item for which to search.

Selecting an Item

Occasionally, you will want to set the selection from inside your programs. This is accomplished by setting the value of the list box's CurSel property as follows:

```
CALL WOWSETPROP USING WIN-RETURN CTL-H "CURSEL" SEL-VALUE.
```

SEL-VALUE is the 0 relative index of the item to select. CTL-H is the handle of the list box. SEL-VALUE must not be greater than the number of items in the list box - 1 (since the value is zero relative). The number of items in the list box can be determined by checking the value of the list box's Count property.

Retrieving the Selection

You will undoubtedly want to retrieve the text of the selected list box item. This is accomplished by retrieving the value of the list box's SelText property as follows:

```
CALL WOWGETPROP USING WIN-RETURN CTL-H "SELTEXT" SEL-TEXT.
```

SEL-TEXT returns the value of the selected list box item. If no item is selected, space is returned. CTL-H is the handle of the list box.

Removing One or All Items from the List Box

You may want to clear one or all items from the list box during the use of the form containing the list box. To remove a single item from a list box:

```
CALL WOWREMOVEITEM USING WIN-RETURN CTL-H INDEX
```

WIN-RETURN returns 0 if the function is successful. CTL-H is the handle of the list box. INDEX is a numeric data item or literal that specifies the zero-relative index of the item to delete.

To remove all items from a list box, use the WOWCLEAR function as follows:

```
CALL WOWCLEAR USING WIN-RETURN CTL-H.
```

WIN-RETURN returns 0 if the function is successful. CTL-H is the handle of the list box.

Month Calendar Control (Not Available in this Release)

The month calendar control displays a monthly calendar. The calendar can display one or more months at a time. When a user clicks on the name of a month, a pop-up menu appears that lists all of the months of the year. A user can select a month by clicking its name on the menu. A user who is using the date time picker control (see page) can

use the Alt+Down Arrow key combination to activate the month calendar control. A user can scroll the displayed months backward or forward, respectively, either by clicking the left arrow or the right arrow at the top of the control, or by pressing the PageUp or the PageDown key on the keyboard. When a user clicks the year that is displayed at the top of the calendar next to the month, an updown control appears. A user can use this control to change the year. A user also can use the Ctrl+PageUp or the Ctrl+PageDown key combination to scroll from one year to another. A user can press keys on the keyboard to navigate; the arrow keys scroll between days, the Home key moves to the beginning of a month, and the End key moves to the end of a month. Unless the calendar has the NoToday property (see page **129**) set to False, a user can return to the current day by tapping the "Today" label at the bottom of the month calendar control.

To add a month calendar picker control to a form, click **Month Calendar** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

Enabled	FontUnderline	*MonthDelta	Top
*FirstDayOfWeek	Height	*MultiSelect	Visible
FontBold	Left	Name	*WeekNumbers
FontItalic	Locked	*NoToday	Width
FontName	*MaxSelCount	*NoTodayCircle	ZOrder
FontSize	MCColor	TabIndex	
FontStrikethru	MCColorIndex	TabStop	

Event

*Change

FirstDayOfWeek Property

The FirstDayOfWeek property specifies the first day of the week for a month calendar control.

The following table lists the possible values of the FirstDayOfWeek property:

Value	Description
0	Monday (the default)
1	Tuesday
2	Wednesday
3	Thursday
4	Friday
5	Saturday
6	Sunday

MaxSelCount Property

The MaxSelCount property sets the maximum number of days that can be selected in a month calendar control. The default value is 7 (one week).

MonthDelta Property

The MonthDelta property determines the scroll rate for a month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. The default value is 1.

MultiSelect Property

The MultiSelect property allows the user to select a range of dates within the control. By default, the maximum range is one week. You can change the maximum range that can be selected by using the MaxSelCount property (described above).

The following table lists the possible values of the MultiSelect property:

Value	Description
False	The user cannot select a range of dates (the default).
True	The user can select a range of dates.

NoToday Property

The NoToday property determines whether or not the month calendar control will not display the "today" date at the bottom of the control.

The following table lists the possible values of the NoToday property:

Value	Description
False	Displays the "today" date at the bottom of the control (the default).
True	The "today" date does not display at the bottom of the control.

NoTodayCircle Property

The NoTodayCircle property specifies that the month calendar control will not circle the "today" date when the NoToday property (see above) is set to False.

The following table lists the possible values of the NoTodayCircle property:

Value	Description
False	The "today" date, if displayed, is circled (the default).
True	The "today" date, if displayed, is not circled.

WeekNumbers Property

The WeekNumbers property displays week numbers (1-52) to the left of each row of days. Week 1 is defined as the first week that contains at least four days. The default value is False.

The following table lists the possible values of the WeekNumbers property:

Value	Description
False	Week numbers are not displayed to the left of each row of days (the default).
True	Week numbers are displayed to the left of each row of days.

Change Event

The Change event occurs when a change has occurred within the month calendar control.

Option Button Control

The option button (also known as radio button) control displays an option that can be turned on or off. Option buttons are usually used in groups where turning one button on turns the others off. For more information on how to group option buttons, see page **131**.

To add an option button control to a form, click **Option Button** from the Toolbox.

Note If you are working with the option button field/control in an RM/Panels panel library, see page **232**.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties and events, see "Common Intrinsic Control Properties" on page **166** and "Common Intrinsic Control Events" on page **175**.

Properties

3D	Enabled	FontStrikethru	Left	Top
*Alignment	FontBold	FontUnderline	Locked	*Value
*AutoPress	FontItalic	ForeColor	Name	Visible
BackColor	FontName	Group	TabIndex	Width
Caption	FontSize	Height	TabStop	ZOrder

Events

Click	KeyDown	KeyUp
GotFocus	KeyPress	LostFocus

Note The user can change the state of an option button in two ways: by clicking with the mouse or by pressing the Spacebar while the option button has input focus. With either method, the Click event for the option button is triggered. You may want to add event-handling code to this event in order to enable/disable other controls based on the new state of the option button.

Alignment Property

The Alignment property controls the position of the text in an option button control.

By default, the caption of an option button displays to the right of the box. The text may be moved to the left of the button with the Alignment property. When using the 3D property, however, the caption must be on the right.

The following table lists the possible values of the Alignment property:

Value	Description
False	Displays text to the right of the option button (the default).
True	Displays text to the left of the option button.

AutoPress Property

The AutoPress property determines whether the state of an option button control is automatically changed when pressed. This behavior is similar to the AutoCheck property of the check box control.

The following table lists the possible values of the AutoPress property:

Value	Description
False	Option button state will not automatically change when pressed.
True	Option button state will automatically change when pressed (the default).

Value Property

The Value property determines the state of an option button control.

The following table lists the possible values of the Value property:

Value	Description
False	Option button is not pushed (the default).
True	Option button is pushed.

Grouping Option Buttons

At first glance, the option button control seems similar to the check box control. Because it has two states, pushed and unpushed, you might think that it would also be used for True/False type conditions. However, this is not the case.

The option button is almost always used in a group with other option buttons. Together, these option buttons represent a group of mutually exclusive choices. When one option button is selected, it deselects whatever other button in the group was previously selected. Only one button in the group can be selected at any time.

This control also solves another tedious programming problem very easily, that of choosing one of a limited number of exclusive options. Since only one option button can be selected at a time, you do not have to validate any user input. You only need to determine which option button is selected. A group of option buttons is very similar to a list box, which is discussed on page **119**.

When you create a group of option buttons, you must indicate to Windows that they are a group. For example, let's say you are creating two groups of option buttons, each with three buttons in a group. Windows needs to know which buttons go together, so that it does not treat all six as one big group.

To group the option buttons, you use two properties together, the TabIndex and Group properties. The TabIndex property determines the input order of controls. Option buttons in a group must have sequential input order. If the first option button in a group has a TabIndex setting of 3, the next option button must have a TabIndex of 4, and the next one 5.

The Group property indicates that a control is the first control in a group. The first option button in a group must have the Group property set to True. The other option buttons in that group must have the Group property set to False. The first control that follows a group of controls, that is, the control whose input order (TabIndex) is subsequent to the last one in the group, should have its Group property set to True so that Windows knows where the group ends.

If you have two groups of three option buttons each, the Group and TabIndex properties should be set in the following manner:

First Group	Set Group Property to	Set TabIndex Property to
Button 1	True	x
Button 2	False	x + 1
Button 3	False	x + 2
Second Group		
Button 4	True	y
Button 5	False	y + 1
Button 6	False	y + 2

Progress Bar Control

A progress bar control consists of a patterned block which can be used to show the status of a long operation.

To add a progress bar control to a form, click **Progress Bar** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

All of the properties for this control is listed in the following table. Properties that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For more information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

Height	*Maximum	TabIndex	Visible
*Increment	*Minimum	Top	Width
Left	Name	*Value	ZOrder
Locked			

Note Because the progress bar control allows no user interaction, no events are associated with it.

Increment Property

The Increment property value is used to increment the progress bar when it receives a PBM-STEPIT message.

Maximum Property

The Maximum property specifies the maximum allowable value for the progress bar and is used in determining how much of the progress bar should be filled.

Minimum Property

The Minimum property specifies the minimum allowable value for the progress bar and is used in determining how much of the progress bar should be filled.

Value Property

The Value property specifies the value of the progress bar and should be in the range specified by the settings of the Minimum and Maximum properties.

Rectangle Shape

The rectangle shape is used to draw the geometric shape of a rectangle on the form. Rectangles are used for the cursor clipping region, the invalid portion of the client area, an area for displaying formatted text, or the scroll area. Your applications can also use rectangles to fill, frame, or invert a portion of the client area with a given brush, and to retrieve the coordinates of a window or a window's client area.

To add a rectangle shape control to a form, click **Rectangle** from the Toolbox.

Note This shape is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this shape will not be displayed on the Cobol-WOW Toolbox.

All of the properties for this shape are listed in the following table. For detailed information on these properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

BackBrushHatch	Fill	Left	PenSize	Top
BackBrushStyle	ForeColor	Locked	PenStyle	Width
BackColor	Height	Name	TabIndex	ZOrder

Note Because the rectangle shape allows no user interaction, no events are associated with it.

Rounded Rectangle Shape

The rounded rectangle shape is used to draw the geometric shape of a rectangle with rounded corners on the form. Rectangles are used for the cursor clipping region, the invalid portion of the client area, an area for displaying formatted text, or the scroll area. Your applications can also use rectangles to fill, frame, or invert a portion of the client area with a given brush, and to retrieve the coordinates of a window or a window's client area.

To add a rounded rectangle shape control to a form, click **Rounded Rectangle** from the Toolbox.

Note This shape is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this shape will not be displayed on the Cobol-WOW Toolbox.

All of the properties for this shape are listed in the following table. Properties that apply only to this shape, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For detailed information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

BackBrushHatch	Height	PenStyle	Width
BackBrushStyle	Left	*RoundnessX	ZOrder
BackColor	Locked	*RoundnessY	
Fill	Name	TabIndex	
ForeColor	PenSize	Top	

Note Because the rounded rectangle shape allows no user interaction, no events are associated with it.

RoundnessX Property

The RoundnessX property specifies the width of the ellipse used to draw the rounded corners.

RoundnessY Property

The RoundnessY property specifies the height of the ellipse used to draw the rounded corners.

Scroll Bar Controls

A vertical scroll bar displays a vertical bar that can be used to scroll information. A horizontal scroll bar displays a horizontal bar that can be used to scroll information. For more information on using scroll bars, see page **139**.

To add a scroll bar control to a form, click either **Horizontal Scroll Bar** or **Vertical Scroll Bar** from the Toolbox.

Note If you are working with the scroll bar field/control in an RM/Panels panel library, see page **234**.

All of the properties and events for both these controls are listed in the following tables. Properties and events that apply only to these controls, or that require special

consideration when used with them, are marked with an asterisk (*). These particular
items are documented in the following sections. For information on the remaining
properties, see ".

Properties

Enabled	Locked	TabIndex	Width
Group	*Maximum	TabStop	ZOrder
Height	*Minimum	Top	
Left	Name	*Value	
*LineChange	*PageChange	Visible	

Events

*EndScroll	*LineRight (Horizontal)	*PageLeft (Horizontal)	*ThumbPos
*LineDn (Vertical)	*LineUp (Vertical)	*PageRight (Horizontal)	*ThumbTrk
*LineLeft (Horizontal)	*PageDn (Vertical)	*PageUp (Vertical)	

Note There are a number of events associated with the scroll bar, related to the different
ways in which the thumb can be moved. No matter how the thumb is moved, the
EndScroll event is always generated when the user has finished moving the thumb.
Unless the contents of some part of the form are to be scrolled while the thumb is being
dragged, the EndScroll event is the best place to respond to changes in thumb position.

LineChange Property

The LineChange property determines the change in position of a scroll bar control when
the mouse is clicked on the arrows at the end of the scroll bar.

Set the LineChange property with any value greater than 0 but less than the difference
specified between the Minimum and Maximum property values. In addition, note that the
LineChange setting should be less than the value specified in the PageChange property.

Maximum Property

The Maximum property determines the highest value allowed for a scroll bar position.

Set the Maximum property with any value from 0 to 65535. Note that this value should
be greater than the value specified in the Minimum property.

Minimum Property

The Minimum property determines the lowest value allowed for a scroll bar position.

Set the Minimum property with any value from 0 to 65535. Note that this value should be less than the value specified in the Maximum property.

PageChange Property

The PageChange property determines the amount the position of a scroll bar control changes when the mouse is clicked on the scroll bar.

Set the PageChange property with any value greater than 0 but less than the difference specified between the Minimum and Maximum property values. In addition, note that the PageChange setting should be greater than the value specified in the LineChange property.

Value Property

The Value property, a numeric value, determines the position of the scroll bar thumb. This value will never be lower than the value of the Minimum property, or greater than the value of Maximum property. If the thumb is positioned at the top or left of the scroll bar, the Value property is equal to the Minimum property. If the thumb is positioned at the bottom or right of the scroll bar, the Value property is equal to the Maximum property. If the thumb is positioned somewhere between the ends of the scroll bar, the value is proportional to the position of the thumb, within the numeric range established by the Minimum and Maximum properties.

Set the Value property with any value from that of the Minimum property to the value of the Maximum property.

EndScroll Event

The EndScroll event occurs after every change in the scroll bar thumb position.

LineLeft Event (Horizontal)

The LineLeft event occurs when the mouse is clicked on the arrow at the left of the horizontal scroll bar.

LineRight Event (Horizontal)

The LineRight event occurs when the mouse is clicked on the arrow at the right of the horizontal scroll bar.

LineDn Event (Vertical)

The LineDn event occurs when the mouse is clicked on the arrow at the bottom of the vertical scroll bar.

LineUp Event (Vertical)

The LineUp event occurs when the mouse is clicked on the arrow at the top of the vertical scroll bar.

PageLeft Event (Horizontal)

The PageLeft event occurs when the mouse is clicked on the bar to the left of the thumb on a horizontal scroll bar.

PageRight Event (Horizontal)

The PageRight event occurs when the mouse is clicked on the bar to the right of the thumb on a horizontal scroll bar.

PageDn Event (Vertical)

A PageDn event occurs when the mouse is clicked on the bar to the right of or below the thumb on a vertical scroll bar.

PageUp Event (Vertical)

A PageUp event occurs when the mouse is clicked on the bar to the left of or above the thumb on a vertical scroll bar.

ThumbPos Event

A ThumbPos event occurs when the mouse is released after being clicked on the scroll bar thumb.

ThumbTrk Event

A ThumbTrk event occurs when the mouse is pressed on the scroll bar thumb.

Using Scroll Bars

The scroll bar control is used to allow a numeric value to be manipulated as a thumb position on a bar. By specifying the minimum and maximum, the value can be viewed relative to a range of possible values. This value and the scroll bar are often used to scroll the display of other information on a form.

For example, let's say a form is used for order entry and displays five lines of a possible 100 on an order. The scroll bar could be used to scroll the view to include the other lines on the order. In this case, by specifying the minimum value as 0 and the maximum value as 95, the scroll bar value could be used directly as the offset between the displayed order line and the actual order line.

Although scroll bars can be vertical or horizontal, they function in the same manner. The thumb on the scroll bar can be dragged to a desired position with the mouse. The thumb also can be moved by clicking the bar on either side of the thumb, or by clicking one of the arrows at either end of the bar.

Clicking the body of the scroll bar or on the arrows moves the thumb in different, configurable increments. Clicking the body of the scroll bar moves the thumb by the increment specified in the Page property. Clicking the arrows at either end of the scroll bar moves the thumb by the increment specified in the Line property. The Page increment, by convention, should be larger than the Line increment. Considering the order entry situation described previously, the Line property should be one and the Page property should be equal to five, which is the number of lines of the order displayed on the form at one time.

Static Text Control

The static text control is used to display text, rectangular outlines, or filled rectangles. These features could reasonably be implemented as several different types of objects, but Windows combines them into one since they have the same properties. See page **142** for special considerations when using static text controls.

You use the static text control most often to display text the user is not allowed to modify, such as labels for other controls. The static text control is also used to draw rectangles or outlines to highlight parts of a form, group controls, or even create a design.

There is rarely a need to retrieve the contents of static text controls since the user cannot change them. However, you may need to change the text of a static text control at runtime. To change the text of a static text control at runtime with the WOWSETPROP function:

```
    CALL WOWSETPROP USING WIN-RETURN CTL-H "TEXT" NEW-TEXT.
```

CTL-H is the handle of the static text control. "TEXT" is the name of the property. NEW-TEXT is the new text of the control.

To add a static text box control to a form, click **Static Text** from the Toolbox.

Note If you are working with the static text field/control in an RM/Panels panel library, see page **236**.

All of the properties for this control are listed in the following table. Properties that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

3D	FontBold	ForeColor	*NoPrefix	*WordWrap
*Alignment	FontItalic	Group	TabIndex	ZOrder
BackColor	FontName	Height	Top	
Caption	FontSize	Left	*Transparent	
*Effect	FontStrikethru	Locked	Visible	
Enabled	FontUnderline	Name	Width	

Note Because the static text control allows no user interaction, no events are associated with it.

Alignment Property

The Alignment property determines how text is positioned in a static text control. The Alignment property allows the text of any static text control, not just multiline controls, to be aligned to the right, left, or center of the control.

The following table lists the possible values of the Alignment property:

Value	Description
0	Normal – Performs no justification (the default).
1	Left justifies text.
2	Centers text.
3	Right justifies text.

Effect Property

The Effect property changes a static text control into an empty rectangle or a colored group box without text. The color names actually designate one of the Windows configuration options and may not match the color name used.

The Effect property is used to determine the type of static text control that is displayed: text, outline, or rectangle. It is important to note that the text of a static text control is not displayed when the outline or rectangle effect is selected. When the 3D property is set to True, the Effect property also has different appearances.

The following table lists the possible values of the Effect property:

Value	Description
0	None – Text is displayed (the default).
1	Draws a rectangle with the window group box color, usually black.
2	Draws a rectangle with the desktop background color, usually gray.
3	Draws a rectangle with the parent window's background, usually white.
4	Draws a black group box.
5	Draws a gray group box.
6	Draws a white group box.

NoPrefix Property

The NoPrefix property determines whether the ampersand (&) character causes the subsequent character to be underlined in a static text control.

The following table lists the possible values of the NoPrefix property:

Value	Description
False	The ampersand character (&) causes next character to be underlined (the default).
True	The ampersand character (&) character is displayed.

Transparent Property

The Transparent property determines whether the background of the form, or the underlying control, will show through.

The following table lists the possible values of the Transparent property:

Value	Description
False	Causes the background of the form or the underlying control not to show through (the default).
True	Causes the background of the form or the underlying control to show through.

WordWrap Property

The WordWrap property determines whether text is wrapped to multiple lines on a static text control.

The following table lists the possible values of the WordWrap property:

Value	Description
False	Text is wrapped (the default).
True	Text is not wrapped.

Special Considerations for Static Text Controls

Windows displays all disabled static text controls with gray text. While you may never need to disable a static text control (since they do not have any events attached to them), if you were to do so, the text would appear as gray. If the control is displayed on a form with the default gray background, the control will not be visible.

Status Bar Control

A status bar control display status information in a horizontal window at the bottom of an application window. Status bars are often divided into sections, called panes, and each pane displays different status information.

When the status bar shows only one pane, it is in "simple mode." When the text of the window is set, the window is invalidated, but it is not redrawn until the next WM-PAINT message. Waiting for the message reduces screen flicker by minimizing the number of times the window is redrawn. A simple mode status bar is useful for displaying Help text for menu items while the user is scrolling through the menu.

To add a status bar control to a form, click **Status Bar** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

All of the properties for this control is listed in the following table. Properties that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For more information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

*CurSection	*SectionNoBorders	*SimpleNoBorders	Visible
Height	*SectionPopOut	*SimplePopOut	Width
Left	*Sections	*SimpleStatus	ZOrder
Locked	*SectionStatus	TabIndex	
Name	*SectionWidth	Top	

Note Because the status bar control allows no user interaction, no events are associated with it.

CurSection Property

The CurSection property controls the currently selected section (or pane) in the status bar. The value is a zero-based index to the status bar panes, where 0 indicates the first pane, 1 indicates the second pane, and so on. The number of panes is controlled by the Sections property.

SectionNoBorders Property

The SectionNoBorders property specifies whether or not the text in the specified pane of a status bar is drawn without borders. The value of CurSection determines which pane is affected.

The following table lists the possible values of the SectionNoBorders property:

Value	Description
False	The text in the status bar pane is drawn with borders (the default).
True	The text in the status bar pane is drawn without borders.

SectionPopOut Property

The SectionPopOut property determines whether the text in the specified pane of a status bar is drawn with a border to appear higher than the plane of the status bar. The value of CurSection determines which pane is affected.

The following table lists the possible values of the SectionPopOut property:

Value	Description
False	The text is not drawn with a border to appear higher than the plane of the status bar (the default).
True	The text is drawn with a border to appear higher than the plane of the status bar.

Sections Property

The Sections property indicates the number of panes into which the status bar is divided. The number of sections cannot be greater than 256.

SectionStatus Property

The SectionStatus property specifies the text that appears in the specified pane of the status bar. The value of CurSection determines which pane is affected.

SectionWidth Property

The SectionWidth property is a pointer to an integer array. The number of elements is specified in the Sections property. Each element specifies the position, in client coordinates, of the right edge of the corresponding part. If an element is -1, the right edge of the corresponding part extends to the border of the window.

SimpleNoBorders Property

The SimpleNoBorders property specifies whether or not the text in the status bar is drawn without borders when the status bar is in simple mode, that is, when only one pane is visible.

The following table lists the possible values of the SimpleNoBorders property:

Value	Description
False	The text in the status bar pane is drawn with borders (the default).
True	The text in the status bar pane is drawn without borders.

SimplePopOut Property

The SimplePopOut property determines whether the text in the status bar is drawn with a border to appear higher than the plane of the status bar when the status bar is in simple mode, that is, when only one pane is visible.

The following table lists the possible values of the SimplePopOut property:

Value	Description
False	The text is not drawn with a border to appear higher than the plane of the status bar (the default).
True	The text is drawn with a border to appear higher than the plane of the status bar.

SimpleStatus Property

The SimpleStatus property specifies the text that appears in the status bar when it is in simple mode.

Tab Control

A tab control is a container control, meaning it allows other controls to be placed inside it. The tab control has several tabs at the top of the control. When a control is added to the tab, it is attached to the tab that is currently selected. When another tab is selected, the controls for the other tabs are hidden and the controls for the selected tab are displayed. This is an excellent way to organize controls by category, rather than placing a large number of controls in a single window.

All the controls on the tab are created when the tab control is created and destroyed when the tab control is destroyed. The controls are not created and destroyed as different tabs are selected. This means that the controls can be initialized once when the tab is created, and the control values retrieved once, before the tab is destroyed. There is no need to initialize or read from the controls just because a new tab is being selected.

 To add a tab control to a form, click **Tab** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

At the current time, certain properties unique to the tab control (Buttons, FixedWidth, ForceLabelLeft, GetFocus, Multiline, RightJustify, and Tabs) can be manipulated only in the Cobol-WOW Designer. The runtime functions, WOWGETPROP and WOWSETPROP, will not recognize these properties.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

*Buttons	FontName	*GetFocus	Name	Top
*CurTab	FontSize	Height	*RightJustify	Visible
*FixedWidth	FontStrikethru	Left	TabIndex	Width
FontBold	FontUnderline	Locked	*Tabs	ZOrder
FontItalic	*ForceLabelLeft	*Multiline	*TabText	

Events

*KeyDown	*SelChange	*SelChanging

Buttons Property

The Buttons property controls the way the tabs are displayed. Setting its value to True makes tabs appear as buttons. This implies that the application should take immediate action when one of the buttons is pressed.

The following table lists the possible values of the Buttons property:

Value	Description
False	Causes tabs to appear as tabs (the default).
True	Causes tabs to appear as buttons.

CurTab Property

The CurTab property controls the currently selected tab in the Cobol-WOW Designer. Change this value to select the desired tab before adding controls to it, and before setting the TabText property, which applies to each tab individually. The value is a zero-based index to the tabs, where 0 indicates the first tab, 1 indicates the second tab, and so on.

FixedWidth Property

The FixedWidth property allows all tabs to be the same width.

The following table lists the possible values of the FixedWidth property:

Value	Description
False	Tabs are displayed with varying widths (the default).
True	Tabs are displayed in the same width.

ForceLabelLeft Property

The ForceLabelLeft property determines whether or not tab static texts are forced to the left. If the ForceLabelLeft property is set, the FixedWidth property must be set to True.

The following table lists the possible values of the ForceLabelLeft property:

Value	Description
False	Tabs are not forced to the left (the default).
True	Tabs are forced to the left.

GetFocus Property

The GetFocus property determines whether or not the text of the selected tab has input focus. Setting the GetFocus property to False on a tab control prevents input focus from going to the text of the selected tab. It does not prevent focus from going to any of the controls on the tab. When the tab control receives focus, the text of the tab itself gets selected with a box.

The following table lists the possible values of the GetFocus property:

Value	Description
False	Prevents input focus from going to the text of the selected tab (the default).
True	Text of the selected tab has input focus.

Multiline Property

The Multiline property determines whether the tabs will occupy multiple lines if the tab control is too narrow for all the tabs to be displayed on a single line.

The following table lists the possible values of the Multiline property:

Value	Description
False	Prevents the tabs from occupying multiple lines (the default).
True	Allows the tabs to occupy multiple lines.

RightJustify Property

Not implemented.

Tabs Property

The Tabs property determines how many tabs are displayed on the control.

TabText Property

The TabText property controls the text of each tab. The value of CurTab determines which tab is affected.

KeyDown Event

The KeyDown event notifies a tab control's parent window that a key has been pressed. This message is sent in the form of a WM-NOTIFY message.

SelChange Event

The SelChange event notifies a tab control's parent window that the currently selected tab has changed.. This message is sent in the form of a WM-NOTIFY message.

SelChanging Event

The SelChanging event notifies a tab control's parent window that the currently selected tab is about to change. This message is sent in the form of a WM-NOTIFY message.

Timer Control

The timer control provides a measured time interval that can be tied to events.

To add a time control to a form, click **Timer** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

Enabled	Left	TabIndex	Width
Height	Locked	Top	ZOrder
*Interval	Name	Visible	

Event

*Timer

Interval Property

The Interval property specifies the length of time between timer ticks in milliseconds.

Timer Event

The Timer event enables or disables one event per timer tick (interval).

Toolbar Control

A toolbar control consists of a series of buttons that can be placed at the top and/or bottom of a form. You can put two toolbars on a form, one at the top and one at the bottom. Event-handling code can be attached to each button in the toolbar. Each button in the toolbar can contain a bitmap and/or text.

The interaction of button groups and the wrapping properties of the toolbar are somewhat obscure. Liant Software Corporation has not fully isolated the interaction of all of these properties, and documentation from Microsoft is sketchy.

To add a toolbar control to a form, click **Toolbar** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

At the current time, properties unique to the toolbar control (AlignTop, BitmapHeight, BitmapWidth, BtnBitmap, BtnHidden, BtnStyle, BtnWrap, ButtonHeight, Buttons, ButtonWidth, Larger, Rows, and Wrapable) can be manipulated only in the Cobol-WOW Designer.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Note The properties that begin with the prefix "Btn" refer to a single button on the toolbar. The button being referred to is controlled by the setting of the CurButton property. All other properties apply to the entire toolbar.

Properties

*AlignTop	*BtnState	*ButtonWidth	Name	*Wrapable
*BitmapHeight	*BtnStyle	*CurButton	*Rows	ZOrder
*BitmapWidth	*BtnText	Height	TabIndex	
*BtnBitmap	*BtnWrap	Larger	Top	
*BtnEnabled	*ButtonHeight	Left	Visible	
*BtnHidden	*Buttons	Locked	Width	

Event

*Button-0

AlignTop Property

The AlignTop property determines the placement of the toolbar on the form.

The following table lists the possible values of the AlignTop property:

Value	Description
False	Places the toolbar at the bottom of the form.
True	Places the toolbar at the top of the form (the default).

BitmapHeight Property

All bitmaps placed in the toolbar must be the same size. The BitmapHeight property specifies the height of the bitmaps to be placed on the toolbar. This is not only the height at which bitmaps are displayed, but also the height of the bitmaps as they were created.

BitmapWidth Property

All bitmaps placed in the toolbar must be the same width. The BitmapWidth property specifies the width of the bitmaps to be placed on the toolbar. This is not only the width at which the bitmaps are displayed, but also the width of the bitmaps as they were created.

BtnBitmap Property

The BtnBitmap property is an optional bitmap that will be displayed in the button. An example of such a bitmap is the scissors in the Cut button.

BtnEnabled Property

The BtnEnabled property controls whether or not the button can be clicked at runtime. Cobol-WOW provides runtime support for the BtnEnabled property using WOWGETPROP and WOWSETPROP, which allows the enabled state of the toolbar button to be set or retrieved at runtime. Before getting or setting the BtnEnabled property value, the CurButton property must be set to the zero-based index of the desired button. Setting the CurButton property has no effect on the user interface.

The following table lists the possible values of the BtnEnabled property:

Value	Description
False	The toolbar button cannot be clicked at runtime.
True	The toolbar button can be clicked at runtime (the default).

BtnHidden Property

The BtnHidden property determines whether or not the button is displayed.

The following table lists the possible values of the BtnHidden property:

Value	Description
False	The toolbar button is displayed (the default).
True	The toolbar button is not displayed.

BtnState Property

The BtnState property determines the initial state of the button. Cobol-WOW provides runtime support for the BtnState property using WOWGETPROP and WOWSETPROP, which allows the state of the toolbar button to be set or retrieved at runtime. Before getting or setting the BtnState property value, the CurButton (see page **153**) property must be set to the zero-based index of the desired button. Setting the CurButton property has no effect on the user interface.

The following table lists the possible values of the BtnState property:

Value	Description
0	Normal – The button accepts user input.
1	Checked – The button is being clicked.
2	Pressed – The button is being clicked.
3	Indeterminate – The button is grayed.

BtnStyle Property

The BtnStyle property determines the style of the button. The check style creates a button that stays pressed. Group and checkgroup are normal and check buttons, respectively, that begin a group of buttons that work together. The separator style creates a button that looks like a space between buttons and that cannot be pressed.

The following table lists the possible values of the BtnStyle property:

Value	Description
0	Button – Creates a standard button.
1	Check – Creates a dual-state push button that toggles between the pressed and nonpressed states each time the user clicks it. The button has a different background color when it is in the pressed state.
2	Group – Creates a button that stays pressed until another button in the group is pressed.
3	CheckGroup – Creates a button that stays pressed until another button in the group is pressed, similar to option buttons.
4	Separator – Creates a separator, providing a small gap between button groups. A button that has this style does not receive user input.

BtnText Property

The BtnText property allows optional text to display on the button.

BtnWrap Property

The BtnWrap property will allow the toolbar to wrap to the next line after the current button. Wrapping is also done at separators, but will not be done within a group.

The following table lists the possible values of the BtnWrap property:

Value	Description
False	The toolbar is wrapped (the default).
True	The toolbar is not wrapped.

ButtonHeight Property

The ButtonHeight property determines the displayed height of the buttons. If this value is set less than the height required by the button's bitmap or text, this value will be ignored.

Buttons Property

The Buttons property determines the number of buttons on the toolbar.

ButtonWidth Property

The ButtonWidth property determines the displayed width of the buttons. If this value is set less than the width required by the button's bitmap or text, this value will be ignored.

CurButton Property

The CurButton property specifies which button's properties are displayed in are accessible through the Btn-prefixed property values. Setting the CurButton property has no effect on the user interface. Before getting or setting the either the BtnState (see page **152**) or the BtnEnabled (see page **151**) property value, the CurButton property must be set to the zero-based index of the desired button.

Larger Property

The Larger property allows the size of the toolbar to be increased.

The following table lists the possible values of the Larger property:

Value	Description
False	The toolbar occupies the number of rows indicated by the Rows property.
True	Allows the toolbar to occupy more rows than indicated by the Rows property (the default).

Rows Property

The Rows property indicates how many rows can be used to display the toolbar. This property can be ignored, based on the grouping and separation of buttons.

Wrapable Property

The Wrapable property indicates that a toolbar may be wrapped to subsequent lines if it is too long.

Button-0 Event

The Button-0 event indicates that the user clicked on the specified button on the toolbar.

Trackbar Control

A trackbar control displays a window containing a slider and optional tick marks used to select a value or a set of consecutive values in a range. The trackbar control can be oriented either horizontally or vertically. Trackbars are useful when you want the user to select a discrete value or a set of consecutive values in a range. For example, you might use a trackbar to allow the user to set the repeat rate of the keyboard by moving the slider to a given tick mark.

To add a trackbar control to a form, click **Trackbar** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

At the current time, certain properties unique to the trackbar control (AutoTicks, BothTicks, EnableSelRange, LeftTicks, NoThumb, NoTicks, TopTicks, SelEnd, SelStart,

and Vertical) can be manipulated only in the Cobol-WOW Designer. The runtime functions, WOWGETPROP and WOWSETPROP, will not recognize these properties.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

*AutoTicks	*LeftTicks	*NoThumb	TabStop	Visible
*BothTicks	*LineChange	*NoTicks	*TickFreq	Width
Enabled	Locked	*PageChange	Top	ZOrder
*EnableSelRange	*Maximum	*SelEnd	*TopTicks	
Height	*Minimum	*SelStart	*Value	
Left	Name	TabIndex	*Vertical	

Events

*Bottom	*LineDown	*PageDown	*ThumbPos	*Top
*EndTrack	*LineUp	*PageUp	*ThumbTrk	

AutoTicks Property

The AutoTicks property determines whether or not the trackbar control has tick marks for each increment in its range of values.

The following table lists the possible values of the AutoTicks property:

Value	Description
False	The trackbar control does not have a tick mark for each increment in its range of values.
True	The trackbar control has a tick mark for each increment in its range of values (the default).

BothTicks Property

The BothTicks property determines whether or not tick marks are displayed on both sides of the trackbar control.

The following table lists the possible values of the BothTicks property:

Value	Description
False	The trackbar control does not display tick marks on both sides of the control (the default).
True	The trackbar control displays tick marks on both sides of the control.

EnableSelRange Property

The EnableSelRange property determines whether or not the trackbar control displays a selection range. A "selection range" restricts the user to a specified portion of the total range. The logical units do not change, but only a subset of them are available for use. The trackbar highlights the available range and displays triangular tick marks at the start and end. Typically, an application handles the trackbar's notification messages and sets the trackbar's selection range according to the user's input.

The following table lists the possible values of the EnableSelRange property:

Value	Description
False	The trackbar control does not display a selection range (the default).
True	The trackbar control displays a selection range only. The tick marks at the starting and ending positions of a selection range are displayed as triangles (instead of vertical dashes), and the selection range is highlighted.

LeftTicks Property

The LeftTicks property determines whether or not tick marks are displayed to the left of the trackbar control.

The following table lists the possible values of the LeftTicks property:

Value	Description
False	The trackbar control does not display tick marks to the left of the control.
True	The trackbar control displays tick marks to the left of the control (the default).

LineChange Property

The LineChange property determines the change in position of a trackbar control when the mouse is clicked on the arrows at the end of the scroll bar.

Set the LineChange property with any value greater than 0 but less than the difference specified between the Minimum and Maximum property values. In addition, note that the LineChange setting should be less than the value specified in the PageChange property.

Maximum Property

The Maximum property determines the highest value allowed for a scroll bar position.

Set the Maximum property with any value from 0 to 65535. Note that this value should be greater than the value specified in the Minimum property.

Minimum Property

The Minimum property determines the lowest value allowed for a scroll bar position.

Set the Minimum property with any value from 0 to 65535. Note that this value should be less than the value specified in the Maximum property.

NoThumb Property

The NoThumb property determines whether or not the trackbar control displays a slider.

The following table lists the possible values of the NoThumb property:

Value	Description
False	The trackbar control displays a slider (the default).
True	The trackbar control does not display a slider.

NoTicks Property

The NoTicks property determines whether or not the trackbar control displays tick marks.

The following table lists the possible values of the NoTicks property:

Value	Description
False	The trackbar control displays tick marks (the default).
True	The trackbar control does not display any tick marks.

PageChange Property

The PageChange property determines the amount the position of a trackbar control changes when the mouse is clicked on the trackbar.

Set the PageChange property with any value greater than 0 but less than the difference specified between the Minimum and Maximum property values. In addition, note that the PageChange setting should be greater than the value specified in the LineChange property.

SelEnd Property

The SelEnd property sets the ending position of the selection range when the EnableSelRange property is set to True.

SelStart Property

The SelStart property sets the beginning position of the selection range when the EnableSelRange property is set to True.

TickFreq Property

The TickFreq property determines the number of tick marks to display on the control in a range of 1 through 100. The default is 10.

TopTicks Property

The TopTicks property determines whether or not tick marks are displayed above the control.

The following table lists the possible values of the TopTicks property:

Value	Description
False	The trackbar control does not display tick marks above the control.
True	The trackbar control displays tick marks above the control (the default).

Value Property

The Value property specifies the value of the trackbar and should be in the range specified by the settings of the Minimum and Maximum properties.

Vertical Property

The following table lists the possible values of the Vertical property:

Value	Description
False	The trackbar control is not oriented vertically (the default).
True	The trackbar control is oriented vertically.

Bottom Event

The Bottom event occurs when the user interacts with trackbar control the using the End key.

EndTrack Event

The EndTrack event occurs when the user stops interacting with the trackbar control, whether by the keyboard or with the mouse.

LineDown Event

The LineDown event occurs when the user depresses the Down Arrow or PgDn keys.

LineUp Event

The LineUp event occurs when the user depresses the Up Arrow or PgUp keys.

PageDown Event

The PageDown event occurs when the user clicks the area below or to the right of the slider with the mouse or moves to that area using the keyboard.

PageUp Event

The PageUp event occurs when the user clicks the area above or to the left of the slider with the mouse or moves to that area using the keyboard..

ThumbPos Event

The ThumbPos event occurs when the user drags the slider and releases the mouse.

ThumbTrk Event

The ThumbTrk event occurs when the user drags the slider.

Top Event

The Top event occurs when the user interacts with trackbar control the using the Home key.

Updown Control

An Updown control is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control. The value associated with an updown control is called its current position.

An updown control is most often used with a companion control, which is called a buddy window. To the user, an updown control and its buddy window often look like a single control. You can specify that an updown control automatically position itself next to its buddy window and that it automatically set the caption of the buddy window to its current position. For example, you can use an updown control with an edit box control to prompt the user for numeric input.

An updown control without a buddy window functions as a sort of simplified scroll bar. For example, a tab control sometimes displays an updown control to enable the user to scroll additional tabs into view.

To add an updown control to a form, click **Updown** from the Toolbox.

Note This control is not recognized by RM/Panels. If you use the Cobol-WOW Designer to enhance a panel, this control will not be displayed on the Cobol-WOW Toolbox.

At the current time, certain properties unique to the updown control (Accelerators, AccelIncrement, AccelSeconds, AlignLeft, AlignRight, ArrowKeys, BuddyInteger, CurAccel, NoThousands, and Wrapable) can be manipulated only in the Cobol-WOW Designer. The runtime functions, WOWGETPROP and WOWSETPROP, will not recognize these properties.

All of the properties and events for this control are listed in the following tables. Properties and events that apply only to this control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Intrinsic Control Properties" on page **166**.

Properties

*Accelerators	*Base	*Horizontal	*NoThousands	Width
*AccelIncrement	*Buddy	Left	TabIndex	*Wrapable
*AccelSeconds	*BuddyInteger	Locked	TabStop	ZOrder
*AlignLeft	*CurAccel	*Maximum	Top	
*AlignRight	Enabled	*Minimum	*Value	
*ArrowKeys	Height	Name	Visible	

Events

*EndScroll	*ThumbPos

Accelerators Property

The Accelerators property determines the rate at which the current position changes when the up or down arrow is clicked.

AccelIncrement Property

The AccelIncrement property specifies the position change increment to use after the time specified by the AccelSeconds property elapses. The value of CurAccel determines which accelerator is affected.

AccelSeconds Property

The AccelSeconds property specifies the amount of elapsed time, in seconds, before the position change increment specified by the AccelIncrement property is used. The value of CurAccel determines which accelerator is affected.

AlignLeft Property

The AlignLeft property determines whether or not the updown control is aligned with the left edge of its buddy window. The width of the buddy window is decreased to accommodate the width of the updown control.

The following table lists the possible values of the AlignLeft property:

Value	Description
False	The updown control is not aligned with the left edge of its buddy window (the default).
True	The updown control is aligned with the left edge of its buddy window.

AlignRight Property

The AlignRight property determines whether or not the updown control is aligned with the right edge of its buddy window. The width of the buddy window is decreased to accommodate the width of the updown control.

The following table lists the possible values of the AlignRight property:

Value	Description
False	The updown control is not aligned with the right edge of its buddy window (the default).
True	The updown control is aligned with the right edge of its buddy window.

ArrowKeys Property

The ArrowKeys property provides a keyboard interface for an updown control. If this property is set to True, the control processes the Up Arrow and Down Arrow keys. The control also subclasses the buddy window so that it can process these keys when the buddy window has the focus.

The following table lists the possible values of the ArrowKeys property:

Value	Description
False	The updown control does not process the Up Arrow and Down Arrow keys (the default).
True	The updown control processes the Up Arrow and Down Arrow keys.

Base Property

The Base property specifies he radix base for an updown control. The base value determines whether the buddy window displays numbers in decimal or hexadecimal digits. Hexadecimal numbers are always unsigned, and decimal numbers are signed.

The following table lists the possible values of the Base property:

Value	Description
0	The updown control's buddy window displays numbers in decimal digits.
1	The updown control's buddy window displays numbers in hexadecimal digits.

Buddy Property

The Buddy property specifies the buddy window for an updown control.

The following table lists the possible values of the Buddy property:

Value	Description
0	No buddy window
1	A trackbar is the buddy window.
2	A check box is the buddy window.
6	A status bar is the buddy window.

BuddyInteger Property

The BuddyInteger property causes the updown control to set the text of the buddy window (using the WM-SETTEXT message) when the position changes. The text consists of the position formatted as a decimal or hexadecimal string.

The following table lists the possible values of the BuddyInteger property:

Value	Description
False	The text of the buddy window is not set when its position changes.
True	The text of the buddy window is set when its position changes (the default).

CurAccel Property

The CurAccel property controls the currently selected accelerator for the updown control. Change this value to select the desired accelerator before setting the AccelSeconds and AccelSeconds properties, which apply to each accelerator individually. The value is a zero-based index to the accelerator, where 0 indicates the first accelerator, 1 indicates the second accelerator, and so on.

Horizontal Property

The Horizontal property determines whether or not the updown control is used for horizontal scrolling. If this property is set to True, the updown control's arrows point left and right instead of up and down.

The following table lists the possible values of the Horizontal property:

Value	Description
False	The updown control is not used for horizontal scrolling.
True	The updown control is used for horizontal scrolling (the default).

Maximum Property

The Maximum property sets the maximum position (range) for an updown control. The maximum position can be less than the minimum position. Clicking the up arrow button moves the current position closer to the maximum position, and clicking the down arrow button moves towards the minimum position.

Minimum Property

The Minimum property sets the minimum position (range) for an updown control. The maximum position can be less than the minimum position. Clicking the up arrow button moves the current position closer to the maximum position, and clicking the down arrow button moves towards the minimum position.

NoThousands Property

The NoThousands property determines whether or not the updown control inserts a thousands separator between every three digits of a decimal string.

The following table lists the possible values of the NoThousands property:

Value	Description
False	A thousands separator is not inserted between every three digits of a decimal string (the default).
True	A thousands separator is not inserted between every three digits of a decimal string.

Value Property

The Value property specifies the value of the updown control and should be in the range specified by the settings of the Minimum and Maximum properties.

Wrapable Property

The Wrapable property causes the position of the updown control to wrap if it is incremented or decremented beyond the ending or beginning of the range. By default, the current position does not change if the user attempts to increment it or decrement it beyond the maximum or minimum value. You can change this behavior by using the Wrapable property, so the position wraps to the opposite extreme. For example, incrementing past the upper limit wraps the position back to the lower limit.

The following table lists the possible values of the Wrapable property:

Value	Description
False	The current position of the updown control does not change if the user attempts to increment it or decrement it beyond the maximum or minimum value (the default).
True	The current position of the updown control changes if the user attempts to increment it or decrement it beyond the maximum or minimum value.

EndScroll Event

The EndScroll event occurs when the user stops scrolling.

ThumbPos Event

The ThumbPos event occurs when the user drags the slider and releases the mouse.

Common Intrinsic Control Properties

This section summarizes the common properties that may be implemented in an intrinsic control. Refer to the specific control in the preceding sections to determine the unique properties available for the control.

The following properties are used by several types of intrinsic controls.

Properties

3D	Fill	FontUnderline	MCColor	TabIndex
BackBrushHatch	FontBold	ForeColor	MCColorIndex	TabStop
BackBrushStyle	FontItalic	Group	Name	Top
BackColor	FontName	Height	PenSize	Visible
Caption	FontSize	Left	PenStyle	Width
Enabled	FontStrikethru	Locked	ScrollBar	ZOrder

3D Property

The 3D property controls the appearance of a control. If this property is set to True, the control will have a three-dimensional effect.

The following table lists the possible values of the 3D property:

Value	Description
False	A three-dimensional control is not displayed (the default).
True	A three-dimensional control is displayed.

Note Setting the 3D property to a value of True for the check box (see page **94**) and option (radio) button (see page **130**) controls is compatible only if the Alignment property for these particular controls is set to the default. (The default setting displays text to the right of the check box or option button, respectively.) The 3D property is not available for the command button control because the three-dimensional effect is already

implemented by Windows. Windows always displays check box and option button controls in 3D, regardless of the property settings.

The form 3D property settings of 1 (All 3D) and 2 (No 3D) will override the 3D property settings of individual controls. (For more information, see the 3D property description on page **177**.)

BackBrushHatch Property

The BackBrushHatch property specifies the hatch style of the brush used to paint the interior of the geometric shape (ellipse, line, rectangle, or rounded rectangle control).

The following table lists the possible values of the BackBrushHatch property:

Value	Description
0	Horizontal hatch
1	Vertical hatch
2	Forward diagonal (a 45-degree downward, left-to-right hatch)
3	Backward diagonal (a 45-degree upward, left-to-right hatch)
4	Horizontal and vertical cross-hatch (the default)
5	45-degree diagonal cross-hatch

BackBrushStyle Property

The BackBrushStyle property specifies the style of the brush used to paint the interior of the geometric shape (ellipse, line, rectangle, or rounded rectangle control).

The following table lists the possible values of the BackBrushStyle property:

Value	Description
0	Solid brush
1	Hollow brush
2	Hatched brush

BackColor Property

The BackColor property determines the background color of a control. The property is a numeric value with nine digits specifying colors as RRR,GGG,BBB.

In the RGB color model, valid red, green, and blue values are in the range from 0 through 255, with 0 indicating the minimum intensity and 255 indicating the maximum intensity. Set the BackColor property with any value in the range from 000 to 255255255.

When you click on the value area of the property, an ellipsis appears. Clicking on the ellipsis causes a variation of the standard Windows Color dialog box to open so that you can define the basic colors, custom colors, and system colors for the foreground color of the control(s).

Caption Property

The Caption property specifies the caption (or static text) associated with a control.

Set the value of the Caption property with any alphanumeric character, including space.

Enabled Property

The Enabled property determines whether the control can respond to user-generated input (or events).

The following table lists the possible values of the Enabled property:

Value	Description
False	Control is disabled for user input.
True	Control is enabled for user input (the default).

Fill Property

The Fill property determines whether the geometric shape (ellipse, line, rectangle, or rounded rectangle control) is filled by the current brush.

The following table lists the possible values of the Fill property:

Value	Description
False	The shape is not filled by the brush.
True	The shape is filled by the brush (the default).

FontBold Property

The FontBold property determines whether the associated text for the control is displayed in bold font format.

The following table lists the possible values of the FontBold property:

Value	Description
False	Text is not displayed bold (the default).
True	Text is displayed bold.

FontItalic Property

The FontItalic property determines whether the associated text of the control is displayed in italic font format.

The following table lists the possible values of the FontItalic property:

Value	Description
False	Text is not displayed in italics (the default).
True	Text is displayed in italics.

FontName Property

The FontName property determines the font used to display text in a control. The font specified must be present on the system.

FontSize Property

The FontSize property determines the size of the font to be used for text displayed in a control. The size specified must be supported by the font. If the size is not supported by the font, the system will substitute the nearest supported value.

FontStrikethru Property

The FontStrikethru property determines whether the associated text for the control is displayed in a strikethrough font style.

The following table lists the possible values of the FontStrikethru property:

Value	Description
False	No strikeout is used (the default).
True	Strikeout is used.

FontUnderline Property

The FontUnderline property determines whether the associated text for the control is displayed in an underlined font format.

The following table lists the possible values of the FontUnderline property:

Value	Description
False	Text is not underlined (the default).
True	Text is underlined.

ForeColor Property

The ForeColor property determines the color of text in a control. The property is a numeric value with nine digits specifying colors as RRR,GGG,BBB.

In the RGB color model, valid red, green, and blue values are in the range from 0 to 255, with 0 indicating the minimum intensity and 255 indicating the maximum intensity. Set the ForeColor property with any value in the range from 000 to 255255255.

When you click on the value area of the property, an ellipsis appears. Clicking on the ellipsis causes a variation of the standard Windows Color dialog box to open so that you can define the basic colors, custom colors, and system colors for the foreground color of the control(s).

Group Property

The Group property determines whether a control is the start of a group.

The following table lists the possible values of the Group property:

Value	Description
False	Control is not the start of a group (the default).
True	Control is the start of a group.

Height Property

The Height property determines, in pixels, the height of the control.

Set the Height property with any value from 0 to the value specified in the Height property of the form less the value specified in the Top property of the control.

Left Property

The Left property determines, in pixels, the location of the left side of the control. This value is relative to the client area of the form containing the control.

Set the Left property with any value from 0 to the value specified in the Width property for the form.

Locked Property

The Locked property determines whether or not a lock is placed on the control in order to prevent the control from being moved accidentally on the form.

The following table lists the possible values of the Locked property:

Value	Description
False	Control is not locked (the default).
True	Control is locked.

MCColor Property

The MCColor property determines the color of various background or text areas of the month calendar control, based on the value specified in the MCColorIndex property (see below). The property is a numeric value with nine digits specifying colors as RRR,GGG,BBB.

In the RGB color model, valid red, green, and blue values are in the range from 0 through 255, with 0 indicating the minimum intensity and 255 indicating the maximum intensity. Set the BackColor property with any value in the range from 000 to 255255255.

When you click on the value area of the property, an ellipsis appears. Clicking on the ellipsis causes a variation of the standard Windows Color dialog box to open so that you can define the basic colors, custom colors, and system colors for the foreground color of the control(s).

MCColorIndex Property

The MCColorIndex property determines the color of the various background or text areas of the month calendar control. The color is specified as an index value into the color selected in the MCColor property (see above).

The following table lists the possible values of the MCColorIndex property:

Value	Description
0	Returns or sets the background color behind the calendar (the default).
1	Returns or sets the color of the calendar text.
2	Returns or sets the background color of the calendar title.
3	Returns or sets the color of the calendar title text.
4	Returns or sets the background color of the calendar text.
5	Returns or sets the color of the trailing text in the calendar.

Name Property

The Name property identifies the control to the underlying program, and is the name shown in your code. Because every control in a form must have a unique name, Cobol-WOW assigns default names and numbers them sequentially as you add them to a form. For example, if you add three check boxes to a form, Cobol-WOW names them CB1, CB2, and CB3.

Note When you have more than one form in a project, and the same control name exists within more than one of those forms, you must distinguish those names in the event-handling code in the following manner:

control-name1 of form-name1, control-name1 of form-name2, and so forth.

We recommend that you change the Name property so that it describes the control's function, rather than simply accepting the default name. You cannot set or retrieve the value of this property at runtime (that is, while the application is running).

PenSize Property

The PenSize property specifies the width of the pen used to draw the outline of the geometric shape (ellipse, line, rectangle, or rounded rectangle control) in logical units. The default value is 1.

PenStyle Property

The PenStyle property specifies the style of the pen used to draw the outline of the geometric shape (ellipse, line, rectangle, or rounded rectangle control).

The following table lists the possible values of the PenStyle property:

Value	Description
0	The pen is solid.
1	The pen is dashed.
2	The pen is dotted.
3	The pen has alternating dashes and dots.
4	The pen has dashes and double dots.

ScrollBar Property

The ScrollBar property determines whether a scroll bar is included on a combo box or list box control.

The following table lists the possible values of the ScrollBar property:

Value	Description
False	No scroll bar is included.
True	A scroll bar is included (the default).

TabIndex Property

The TabIndex property determines the tab order, that is the order in which Tab and Shift+Tab key presses will move input focus between controls (see page **19** for more information). Controls that have the same TabIndex property value will have undefined tab sequencing. Set the TabIndex property to a value of 1 or greater.

Note The TabIndex property cannot be changed or retrieved at runtime (with the WOWGETPROP and WOWSETPROP functions) and can only be set in the Cobol-WOW Designer.

TabStop Property

The TabStop property determines whether a user can use the Tab key to set the focus to a control in a form.

The following table lists the possible values of the TabStop property:

Value	Description
False	Control is not a tab stop.
True	Control is a tab stop (the default).

Top Property

The Top property determines, in pixels, the location of the top of the control. This value is relative to the client area of the form containing the control.

Set the Top property with any value from 0 to the value specified in the Height property of the form.

Visible Property

The Visible property determines whether the control is visible or hidden at runtime.

The following table lists the possible values of the Visible property:

Value	Description
False	Control is hidden.
True	Control is visible (the default).

Width Property

The Width property determines, in pixels, the width of the control.

Set the Width property with any value from 0 to the value specified in the Width property of the form less the value specified in the Left property of the control.

ZOrder Property

The ZOrder property determines and changes the control stacking order, that is, the order in which controls are created. The controls with the smaller numbers are stacked "behind" the controls with the larger numbers. The controls with the larger numbers are "on top" of all the other controls. The ZOrder property can be manipulated using the Bring To Front and Send To Back commands on the Control menu.

The value is a one-based index to the z-order of the controls, where 1 indicates the first control, 2 indicates the second control, and so on. Cobol-WOW initially sets the z-order for each control to correspond to the order in which they were added to the form. You can also change the z-order by choosing the Z-Order command on the Control menu. (For more information, see page **20** or the *Designer* online Help file.)

Note The ZOrder property cannot be changed or retrieved at runtime (with the WOWGETPROP and WOWSETPROP functions) and can only be set in the Cobol-WOW Designer.

Common Intrinsic Control Events

The following common events are implemented by one or more of the intrinsic controls.

Events

Click	GotFocus	KeyPress	LostFocus
DblClick	KeyDown	KeyUp	

Click Event

The Click event occurs when the user clicks a mouse button on a control.

DblClick Event

The DblClick event occurs when the user double-clicks a mouse button on a control.

Note This event has an affect for a combo box control only when the Style property (see page **100**) is set to a value of 0 (Simple, standard combo box).

GotFocus Event

The GotFocus event occurs when the control receives the focus.

KeyDown Event

The KeyDown event occurs when the user presses a key while the control has input focus. The value of the key pressed is contained in the WIN-CHAR variable declared in **windows.cpy**.

KeyPress Event

The KeyPress event occurs when the user presses and releases a key (or the key is held down for repeat) while the control has input focus. The value of the key pressed is contained in the WIN-CHAR variable declared in **windows.cpy**.

KeyUp Event

The KeyUp event occurs when the user releases a key while the control has input focus. The value of the key pressed is in the WIN-CHAR variable declared in **windows.cpy**.

LostFocus Event

The LostFocus event occurs when the control loses input focus, either by user action, such as tabbing to or clicking another control, or by changing the focus in code.

Forms

Forms are the containers within which you group controls. In traditional programming, you place fields on the screen or in a pop-up window. With Cobol-WOW, you place fields (that is, controls) in a form. Although forms are quite versatile, most of your programming will be involved with manipulating controls, not forms. The form has only default properties associated with it.

Note

All the properties and events for a form are listed in the following tables and are documented in the following sections (details on the events begin on page **187**).

Properties

3D	Caption	Icon	Parent	Title
AllowEventFilter	ClipControls	IconIndex	ScrollBars	Top
BackColor	Cursor	Left	ShowState	Visible
Bitmap	DialogMotion	MaxButton	Style	Width
BitmapMode	Enabled	MinButton	SysKeyMode	
Border	Height	Modal	SystemMenu	

Events

Activate	GetFocus	LButtonDown	MButtonUp	Show
Close	KeyDown	LButtonUp	Paint	
Create	KeyPress	LoseFocus	RButtonDown	
Enable	KeyUp	MButtonDown	RButtonUp	

3D Property

The 3D property controls the three-dimensional appearance of intrinsic controls in a form.

The following table lists the possible values of the 3D property:

Value	Description
0	Mixed – Allows two-dimensional and three-dimensional settings of individual intrinsic controls in a form (the default).
1	All 3D – Forces all intrinsic controls to a three-dimensional appearance.
2	No 3D – Forces all intrinsic controls to a two-dimensional appearance.

Note The form 3D property settings of 1 (All 3D) or 2 (No 3D) will override the 3D property settings of individual controls (see page **166**).

AllowEventFilter Property

The AllowEventFilter property determines whether Cobol-WOW performs filtering on events returned to the COBOL program.

Cobol-WOW returns the messages generated by Windows to the COBOL program to be handled as events by the form and controls. Windows generates many messages, and in

most cases a small minority of these messages are actually acted upon by the COBOL program. To maximize performance, particularly in networked environments, Cobol-WOW filters the messages (events) returned to the COBOL program. This can be done because the Cobol-WOW Designer knows which events have code associated with them. If the AllowEventFilter property is set to True, this filtering is performed for the form and all controls on it.

In some cases, you may add code to your COBOL program that acts on additional messages. Since Cobol-WOW is not aware of this code, it would filter out the associated messages and the code would never be invoked. To prevent this, set the AllowEventFilter property to False when adding additional message handling code directly to your programs.

Note The AllowEventFilter property can be overridden at runtime by selecting the Do Not Filter Events option on the Runtime page of the Preferences dialog box or by customizing the [WOWRT] section (see page **3**) of the **cblwow.ini** file.

The following table lists the possible values of the AllowEventFilter property:

Value	Description
False	Filtering is not performed for the form and all controls on it.
True	Filtering is performed for the form and all controls on it (the default).

BackColor Property

The BackColor property determines the background color of a form. The property is a numeric value with nine digits specifying colors as RRR,GGG,BBB.

In the RGB color model, valid red, green, and blue values are in the range from 0 through 255, with 0 indicating the minimum intensity and 255 indicating the maximum intensity. Set the BackColor property with any value in the range from 000 to 255255255.

When you click on the value area of the property, an ellipsis appears. Clicking on the ellipsis causes a variation of the standard Windows Color dialog box to open so that you can define the basic colors, custom colors, and system colors for the foreground color of the control(s).

Bitmap Property

The Bitmap property specifies that a bitmap is displayed as the background of the form. The BitmapMode property setting, described in the following section, determines the bitmap's appearance. All controls on the form will be displayed on top of the bitmap.

Note The value of this property must be the complete name of a bitmap file. If the bitmap is not in the working directory or in a directory specified in the RUNPATH environment variable, a pathname is also required.

BitmapMode Property

The BitmapMode property determines how a bitmap is displayed in a form. Very rarely will the size of a form and bitmap match exactly. The bitmap can be displayed in its original size, which may not completely fill the form or may truncate part of the bitmap. The bitmap can also be scaled to match the exact size of the form. You can choose the most appropriate technique. Results will vary depending on the original size of the bitmap, the size of the form, and the nature of the bitmap. (See also "Bitmap Property" described in the previous section.)

The following table lists the possible values of the BitmapMode property:

Value	Description
0	Displays the bitmap in its original size (the default). If the bitmap is smaller than the form, the bitmap will be displayed in the upper-left corner of the form, and the remainder of the form will be filled with the background color of the form. If the bitmap is larger than the form, only the portion of the bitmap that will fit in the form will be displayed.
1	Scales the bitmap to fit the exact size of the form. This setting may distort the bitmap image, especially if the size difference between the bitmap and the form is dramatic.
2	Arranges (tiles) the bitmap in multiple images side by side on the form.

Border Property

The Border property determines whether the form displays a border.

The following table lists the possible values of the Border property:

Value	Description
0	Form does not have a border (the default).
1	Form has a thin border.
2	Form has a thick, sizable border.
3	Form has a thick, sizable, dialog-box-style border.

Caption Property

The Caption property determines whether a form has a title bar.

The following table lists the possible values of the Caption property:

Value	Description
False	Form does not have a title (the default).
True	Form has a title bar.

ClipControls Property

The ClipControls property determines whether child controls can extend past the boundaries of a form.

The following table lists the possible values of the ClipControls property:

Value	Description
False	Child controls can extend outside the form (the default).
True	Child controls cannot extend outside the form.

Cursor Property

The Cursor property sets the default state (shape) of the cursor to display as the mouse pointer moves over the form. Each form can have one cursor shape. This value can be set and retrieved at runtime.

The following table lists the possible values of the Cursor property:

Value	Description
0	ARROW — Cursor shape is a diagonal arrow.
1	IBEAM — Cursor shape is an I-bar, indicating editable text.
2	WAIT — Cursor shape is an hourglass, indicating that the program is busy and the user should wait.
3	CROSS — Cursor shape is a simple crosshair.
4	UPARROW — Cursor shape is an up arrow.
5	SIZENWSE — Cursor shape is arrows with a diagonal bar separating them, indicating the northwest and southeast edges of the form are to be resized.
6	SIZENESW — Cursor shape is arrows with a diagonal bar separating them, indicating the northeast and southwest edges of the form are to be resized.
7	SIZEWE — Cursor shape is arrows pointing left and right with a horizontally bar separating them, indicating the form is to be resized horizontally.
8	SIZENS — Cursor shape is arrows pointing up and down with a vertical bar separating them, indicating the form is to be resized vertically.
9	SIZEALL — Cursor shape is arrows with a diagonal bar separating them, indicating the northeast and southwest edges of the form are to be resized.
10	NO — Cursor shape is a circle with a slash through it.
11	APPSTARTING — Cursor shape is an arrow with an hourglass.
12	HELP — Cursor shape is an arrow with question mark, indicating help is available.

DialogMotion Property

The DialogMotion property determines whether Tab key motion between fields and arrow key motion within groups should automatically be performed for a form.

The following table lists the possible values of the DialogMotion property:

Value	Description
False	Dialog motion should not automatically be performed.
True	Dialog motion should automatically be performed (the default).

Enabled Property

The Enabled property determines whether a form can respond to user-generated input (or events).

The following table lists the possible values of the Enabled property:

Value	Description
False	Form is not enabled for user input.
True	Form is enabled for user input (the default).

Height Property

The Height property determines the height, in pixels, of a form.

Set the Height property with any value from 0 to the height of the screen display less the value specified in the Top property.

Icon Property

The Icon property determines the icon to be used for a form when the form is minimized. This property cannot be retrieved or modified at runtime.

Note The Icon property must be specified in the Designer, and it must be the complete name of an icon (**.ico**) file. If the icon file is not in the working directory or in a directory specified in the RUNPATH environment variable, a pathname is also required.

IconIndex Property

The IconIndex property determines the icon to be used for a form when the form is minimized and when more than one icon exists in the icon (**.ico**) file. The value is a zero-based index to the icons, where 0 indicates the icon, 1 indicates the second icon, and so on.

Left Property

The Left property determines, in pixels, the location of the left side of a form. This value is relative to the entire desktop area.

Set the Left property with any value from 0 to the width of the screen display.

MaxButton Property

The MaxButton property determines whether a Maximize button is included on a form.

The following table lists the possible values of the MaxButton property:

Value	Description
False	Form does not have a maximize button (the default).
True	Form has a maximize button.

MinButton Property

The MinButton property determines whether a Minimize button is included on a form.

The following table lists the possible values of the MinButton property:

Value	Description
False	Form does not have a minimize button (the default).
True	Form has a minimize button.

Modal Property

The Modal property determines whether or not the form is the only form the user can operate for the application. If the value of the Modal property is set to True, all other forms will be unavailable to the user until the form is removed, or the value of the Modal property is set to False, or another modal form is displayed.

The following table lists the possible values of the Modal property:

Value	Description
False	The form is not modal (the default).
True	The form is modal.

Parent Property

The Parent property designates the form that serves as the parent of the current form. This property cannot be set or retrieved at runtime.

The Parent property value should be specified in the Designer, using the name of another form in the project. Leaving the value blank indicates that there is no parent form. If a parent is specified, the current form will be positioned relative to the parent and minimized with the parent.

ScrollBars Property

The ScrollBars property determines whether one or more scroll bars are attached to a form.

The following table lists the possible values of the ScrollBars property:

Value	Description
0	No scroll bars are added (the default).
1	A vertical scroll bar is added.
2	A horizontal scroll bar is added.
3	Both vertical and horizontal scroll bars are added.

ShowState Property

The ShowState property determines the manner in which a form is displayed.

The following table lists the possible values of the ShowState property:

Value	Description
0	Form is displayed normally (the default).
1	Form is displayed as maximized, that is, it fills the entire desktop area.
2	Form is displayed as an icon.

Style Property

The Style property is used to determine the type of a form.

The following table lists the possible values of the Style property:

Value	Description
1	Specifies the form as overlapped, which means that the form is a top-level window that has a title bar, border, and client area. It is meant to serve as an application's main window. It can also have a menu, minimize and maximize buttons, and scroll bars. Overlapped windows may own other top-level windows or be owned by other top-level windows or both. (This is the default value.)
2	Specifies the form as a child, which means that the form has a parent. The parent-child relationship determines where a window can be drawn on the screen. A child window can be drawn only within its parent's client area, and is destroyed along with its parent.
3	Specifies the form as a pop-up, which means that the form is a pop-up window. A pop-up window does not have a parent by default (although a parent can be set for it); a pop-up window can be drawn anywhere on the screen. The main differences between a pop-up window and an overlapped window is that a pop-up window can be displayed outside the border of its parent window.

SysKeyMode Property

The SysKeyMode property specifies the way in which a Cobol-WOW application processes WM-SYSKEY messages (controlled on a form-by-form basis). The Windows operating system makes a distinction between system keystrokes and non-system keystrokes. System keystrokes produce system keystroke messages, such as WM-SYSKEYDOWN and WM-SYSKEYUP. Non-system keystrokes produce non-system keystroke messages, such as WM-KEYDOWN and WM-KEYUP. Windows generates a WM-KEYDOWN or a WM-SYSKEYDOWN message when the user presses a key. When the user releases a key, the system generates a WM-KEYUP or a WM-SYSKEYUP message.

The following table lists the possible values of the SysKeyMode property:

Value	Description
0	WantSysKey — The application receives WM-SYSKEY system messages (the default).
1	WantKey — The application receives WM-KEY messages that have been translated from WM-SYSKEY messages.
2	WantKeyandSysKey — The application receives both WM-SYSKEY and WM-KEY messages.
3	None — The application receives neither WM-SYSKEY nor WM-KEY messages.

SystemMenu Property

The SystemMenu property determines whether a form contains a System menu.

The following table lists the possible values of the SystemMenu property:

Value	Description
False	Form does not contain a System menu (the default).
True	Form contains a System menu.

Title Property

The Title property determines whether a form contains a title in the title bar.

Set the Title property with any alphanumeric characters, including spaces.

The title will be displayed only if the value of the Caption property is set to True. See Caption property on page .

Top Property

The Top property determines, in pixels, the location of the top of a form. This value is relative to the entire desktop area.

Set the Top property with any value from 0 to the height of the screen display.

Visible Property

The Visible property determines whether a form is hidden or visible at runtime.

The following table lists the possible values of the Visible property:

Value	Description
False	Form is hidden.
True	Form is visible (the default).

Width Property

The Width property determines, in pixels, the width of a form.

Set the Width property with any value from 0 to the width of the screen display less the value specified in the Left property.

Activate Event

The Activate event occurs whenever the form becomes active or inactive.

Close Event

The Close event occurs when the form is destroyed.

Create Event

The Create event occurs when the form is created.

Enable Event

The Enable event occurs when the form is enabled or disabled.

GetFocus Event

The GetFocus event occurs when the form gets input focus.

KeyDown Event

The KeyDown event occurs when the form has input focus and a key is pressed down. This event does not occur if a control on the form has input focus. The value of the key pressed is contained in the WIN-CHAR variable declared in **windows.cpy**.

KeyPress Event

The KeyPress event occurs when the form has input focus and a key is pressed and released. This event does not occur if a control on the form has input focus. The value of the key pressed is contained in the WIN-CHAR variable declared in **windows.cpy**.

KeyUp Event

The KeyUp event occurs when the form has input focus and a key is released. This event does not occur if a control on the form has input focus. The value of the key pressed is contained in the WIN-CHAR variable declared in **windows.cpy**.

LButtonDown Event

The LButtonDown event occurs when the form has input focus and the left mouse button is depressed.

LButtonUp Event

The LButtonUp event occurs when the form has input focus and the left mouse button is released.

LoseFocus Event

The LoseFocus event occurs when the form loses input focus.

MButtonDown Event

The MButtonDown event occurs when the form has input focus and the middle mouse button is depressed.

MButtonUp Event

The MButtonUp event occurs when the form has input focus and the middle mouse button is released.

Paint Event

The Paint event occurs when the form receives a WM-PAINT message (see the *Functions and Messages* online Help file). Although Cobol-WOW and the COBOL runtime together automatically draw whatever image is required within the form, if you want to dynamically draw something else, the Paint event provides notification that it is permissible to do so.

RButtonDown Event

The RButtonDown event occurs when the form has input focus and the right mouse button is depressed.

RButtonUp Event

The RButtonUp event occurs when the form has input focus and the right mouse button is released.

Show Event

The Show event occurs when the form is hidden or displayed.

Appendix B: Working with ActiveX Controls

This appendix describes special considerations for using ActiveX controls with Cobol-WOW.

ActiveX Controls and Cobol-WOW

Wouldn't it be nice if you weren't limited to using the controls built into the Windows operating system? Wouldn't it be great if you could license specialized controls and plug them right into your development environment, using them as if they were a part of Windows?

That idea has been pursued with varying degrees of success for many years. The latest implementation of this idea is ActiveX controls, and with Cobol-WOW you can use ActiveX controls on 32-bit platforms. What's more, you can use them just like the intrinsic Windows controls.

History of ActiveX Controls

ActiveX controls have an interesting history. They were preceded by VBX controls. VBX controls were a successful implementation of component technology for 16-bit Microsoft Visual Basic. VBX controls could be created by third-party developers, but used within Visual Basic just like the intrinsic Windows controls. This idea of "plug-in" components sparked the creation of hundreds of third-party controls, and contributed significantly to the popularity of Visual Basic.

But VBX controls have two shortcomings. The first is that they are tied closely to a 16-bit architecture, which prevents moving VBX control technology to the 32-bit environment. The second problem is that VBX controls are tied very closely to Visual Basic. This makes it difficult to provide support for VBX controls in other systems.

The designers at Microsoft set out to solve both problems with a new specification for the creation of third-party controls. They started by using two technologies: COM (Component Object Model) and OLE (Object Linking and Embedding). Based on these technologies, Microsoft came up with a specification for OLE Controls, which were later

renamed to OCX controls. With the popularity of the Internet came another modification to the specification and a final rename: ActiveX controls.

Microsoft provides the COM and OLE technologies used by ActiveX controls as part of Windows, but ActiveX is really a specification of how the ActiveX control is created and how it interfaces with the software that uses the control. The real "magic" is in this specification. By knowing the specification, a program that uses an ActiveX control (called a container) can work with the control without having prior knowledge of the control. The application can learn what it needs to know about the control at runtime.

Adding and Removing ActiveX Controls to the Cobol-WOW Designer

The first step in using ActiveX controls with Cobol-WOW is adding them to the Toolbox. To add ActiveX controls to the Toolbox, click the Select ActiveX Controls command on the Cobol-WOW Designer's Options menu to display the Select ActiveX Controls dialog box..

The Select ActiveX Controls dialog box lists the ActiveX controls that are installed on your system and that appear to be compatible with Cobol-WOW. Cobol-WOW determines what controls to list here by searching the Windows Registry entries on the computer to find the registered ActiveX controls. When Cobol-WOW finds a control, it looks to see if the necessary Registry entries are there to allow it to use the control, and also checks to see if the control requires any features not provided by Cobol-WOW. If an expected control does not appear the dialog box list, see the troubleshooting tips on page **193**.

To add any listed control to your Toolbox, just click on the control to select it. When you have selected all the controls you want, click OK. The Toolbox will be reformatted to display the controls you have selected. The controls also will be recorded in the **cblwow.ini** file (see page **3**). An ActiveX control is added to a form in the same manner as an intrinsic control. Simply select the control in the Toolbox, and then click and drag it on the form.

In some cases, several controls will be added to the Toolbox by selecting a single entry in the list box. This is because some controls are distributed and registered as a group.

To remove a control from the Toolbox, deselect the control in the Select ActiveX Controls dialog box and click OK.

Troubleshooting Tips

If an expected control does not appear in the list box, there are several possible reasons:

1. The control has not been registered. It is not enough simply to copy the control's implementation file (.OCX, .DLL) to the system. The control <u>must</u> be described through Windows Registry entries. This is what allows OLE and COM to work with the control. Check the documentation for your control to see how it should be registered. Most controls will be registered by their installation software. The RegEdit program also provides facilities for registering controls.

2. The control's Registry information is incomplete. The following entries are required for the control under HKEY_CLASSES_ROOT\CLSID:

    CLSID

    ProgID

    Control

    TypeLib

    In addition, the TypeLib must also be registered under HKEY_CLASSES_ROOT\TypeLib.

3. The control has a RequiredCategories entry in its Registry entry. This will prevent Cobol-WOW from displaying the control.

Using ActiveX Controls on a Form

An ActiveX Control is added to a form in the same manner as a intrinsic control. Simply select the control in the Toolbox and click and drag on the form to establish the outline of the control.

ActiveX Control Properties

ActiveX control properties are displayed and modified through the Properties dialog box (see page **8**) in the same manner as intrinsic controls. Font, color, and True/False properties work just like intrinsic controls. Other properties have values that are described in the documentation for the control.

ActiveX control properties can be queried and modified at runtime using the WOWSETPROP and WOWGETPROP functions (see pages **51** and **52**, respectively). Note that True/False properties of ActiveX controls have slightly different values. With an ActiveX control, False is zero, but True is -1.

ActiveX Indexed Properties

Some ActiveX control properties occur multiple times and are described as indexed.

Two special functions must be used to get and set indexed properties for ActiveX controls. These functions are AXGETINDEXPROP and AXSETINDEXPROP. They are used as follows.

To retrieve an indexed property:

```
CALL AXGETINDEXPROP USING WIN-RETURN AXCTIVEXCTL-H PROPNAME
    RET-VALUE INDEX-1 INDEX-2 ...
```

WIN-RETURN is a numeric data item that is set to 0 if the function succeeds or to an error code if the function fails.

AXCTIVEXCTL-H identifies the handle generated for the ActiveX control by Cobol-WOW.

PROPNAME is an alphanumeric literal or data item containing the property name.

RET-VALUE is an alphanumeric or numeric data item that will receive the property value.

INDEX-1 INDEX-2 are numeric literals or data items that are the index(es) for the property. If more than one index is specified, the most significant index should be placed first.

To set an indexed property:

```
CALL AXSETINDEXPROP USIGN WIN-RETURN AXCTL-H PROPNAME
    PROP-VALUE INDEX-1 INDEX-2 ...
```

WIN-RETURN is a numeric data item that is set to 0 if the function succeeds or to an error code if the function fails.

AXCTIVEXCTL-H identifies the handle generated for the ActiveX control by Cobol-WOW.

PROPNAME is an alphanumeric literal or data item containing the property name.

PROP-VALUE is an alphanumeric or numeric literal or data item containing the property value to be set

INDEX-1 INDEX-2 are numeric literals or data items which are the index(es) for the property. If more than one index is specified, the most significant index should be placed first.

ActiveX Control Events

ActiveX control events are listed in the Events/Code Sections list of the Event-Handling Code dialog box. Code added to the events in the dialog box will be automatically executed when the control triggers the event.

ActiveX Control Methods

Some ActiveX controls provide special capabilities that are invoked as methods. These methods can be thought of as functions or procedures built into the control. These methods can be executed using the AXDOMETHOD function as follows:

```
CALL AXDOMETHOD USING WIN-RETURN ACTIVEXCTL-H METHOD-NAME
    PARAM-1 PARAM-2 ... [GIVING PARAM-RESULT]
```

Note Information about methods and parameters can be found in the control's documentation. When supplying parameters to a method you do not need to worry about the data types being used. Cobol-WOW will convert the data to the proper format based on information contained in the control. Simply supply the parameters in the proper order. Parameters identified by the control's documentation as optional may be omitted.

WIN-RETURN is a numeric data item that is set to 0 if the method succeeds or to an error code if the method fails.

ACTIVEXCTL-H identifies the handle generated for the ActiveX control by Cobol-WOW.

METHOD-NAME is an alphanumeric literal or data item containing the control name.

PARAM-1 PARAM-2 are the parameters for the method. Multiple parameters may be specified. The reserved word, OMITTED, may be used to designate an unspecified optional parameter when parameter(s) to the right of the OMITTED parameter are to be specified. (Do not use OMITTED for unspecified trailing parameters.)

PARAM-RESULT is an optional parameter to receive the value returned by the method. Note that not all methods return values. This is not the same as the WIN-RETURN parameter; PARAM-RESULT is only valid if the WIN-RETURN value is 0.

Note Some methods may change the content of parameters as an undocumented side effect. Use the BY CONTENT reserved word to protect values in the calling program from such an outcome.

Example

Many controls provide an AboutBox method that will display an About Box identifying the control. This method generally requires no parameters. The AboutBox method for a control with the name MYACTIVEX would be invoked as follows:

```
CALL AXDOMETHOD USING WIN-RETURN MYACTIVEX-H "ABOUTBOX".
```

Limitations

ActiveX controls have the following limitations when used with Cobol-WOW:

- Some control properties may not be available for querying or modification at runtime. This is determined by the control. If this occurs, a message box will be displayed.

- An ActiveX control handle is NOT a window handle. You cannot pass an ActiveX control handle to a function that expects a window handle, such as GETWINDOWTEXT. An attempt to do this will result in a message box being displayed. Some ActiveX controls will expose a window handle as a property to allow you to use Windows API functions on the control.

- ActiveX controls that function as containers are not supported. You cannot place one ActiveX control inside another.

- ActiveX controls that require data binding are not supported.

Distribution Issues

If you use ActiveX controls to develop your application, these controls will have to be distributed with your application. Pay attention to the licensing issues associated with any controls you use. Some controls will require a license only for development use, others require a license for development and deployment.

Appendix C: Understanding the Application Architecture

This appendix defines the architecture for integrating the graphical user interface of an application for Windows with the Cobol-WOW application development framework.

Initial Creation of a Cobol-WOW Program

When you create a new Cobol-WOW program, many files are created and processed. The following figure illustrates the files and components involved in the initial creation of a Cobol-WOW program having two forms. In this example, the program is called MyApp, and the forms are named form1 and form2.
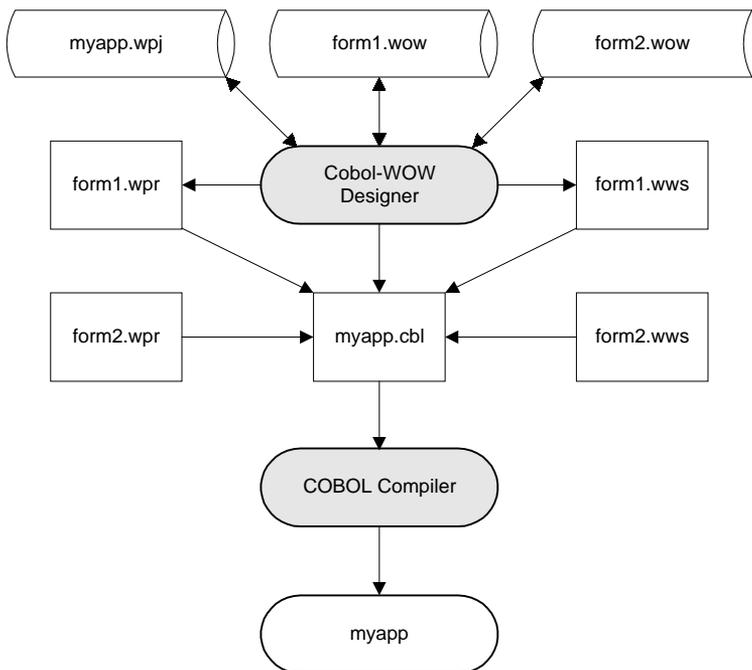
Initial Creation of a Cobol-WOW Program

Project File (.wpj)

The Cobol-WOW Designer creates a project file to store project information, in particular, a list of the forms included in the project. This file, which has the extension **.wpj**, is needed only at design time. As you add or rename forms from the project, this file is automatically updated. For more information, see "Cobol-WOW Projects" on page **57**.

Form File (.wow)

The Cobol-WOW Designer component manages the entire process of creating a form. When you first save a new form, the Designer creates a file that stores the definition of the form. This type of file is known as a Cobol-WOW form file and has a default extension of **.wow**. The .wow file is read and written to by the Designer, but it is not needed during runtime. As you edit a form, the modifications are stored in the .wow file. This file is similar to a word processing file, in that just as a word processing file contains a single document, a .wow file contains the definition of a single form. See page **46** for additional information.

Working Storage Copy File (.wws)

The Cobol-WOW Designer generates a copy file for each form that contains a binary definition of the form. This binary definition is declared as a COBOL data item. This type of copy file is known as a Cobol-WOW Working Storage file and has a default extension of **.wws**. Any program that uses the form must contain this copy file so that it has a definition of the form. Since the form definition is in the COBOL Working-Storage Section, it is loaded into memory when the program is loaded. This allows the form to be created quickly at runtime by a single call with no disk access.

Procedure Division Copy File (.wpr)

The Cobol-WOW Designer also generates a Procedure Division copy file for each form with a default extension of **.wpr**. This copy file contains the event-handling logic for the form and the message interpretation logic that will make the event-handling code execute. This copy file must be included in any program that uses the form.

COBOL Skeleton Program File (.cbl)

The Cobol-WOW Designer generates a skeleton program, based on the project file, with enough logic to display, use, and remove the forms. This type of file is known as an RM/COBOL source file and has a default extension of **.cbl**. This skeleton program provides enough COBOL code to begin your program. As you enhance the program, you will probably want to add additional functionality, such as file access, to the main program.

COBOL Executable Program File (.cob)

The files generated by the Cobol-WOW Designer combine to make a compilable and executable program. This file type is known as an RM/COBOL object file and has a default extension of **.cob**. The Designer executes the COBOL compiler on the skeleton program, which includes the two copy files. This file is created when you compile the source code with the Build command on the Project menu. Once the program is compiled, it can be run like any other COBOL program.

Ongoing Maintenance of a Cobol-WOW Program

As you continue to modify and maintain a Cobol-WOW program, the process is illustrated below.
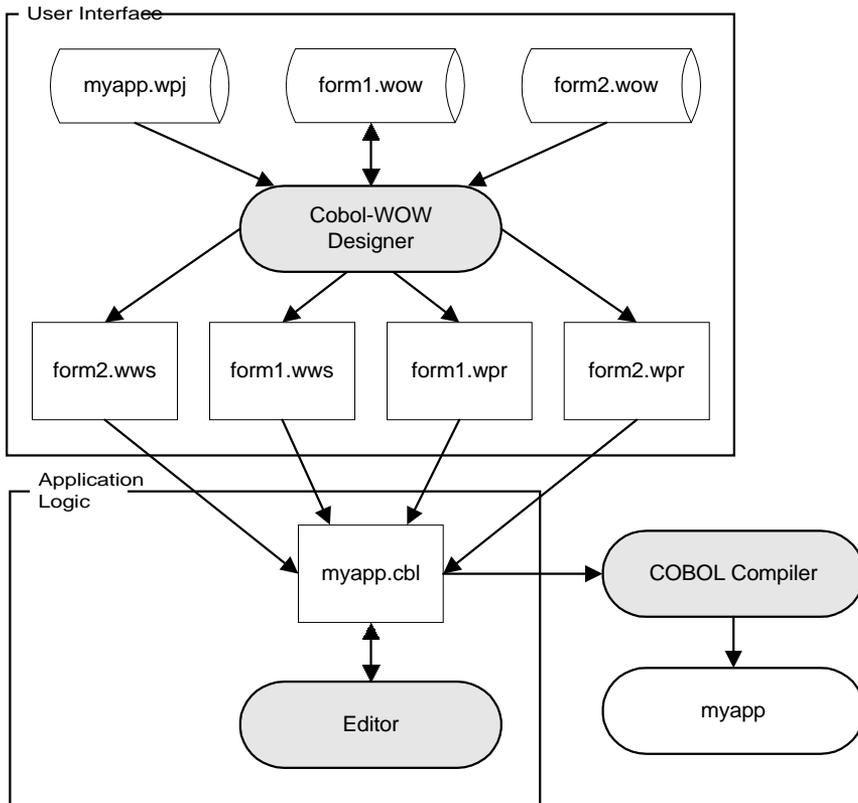
Enhancement and Modification of a Cobol-WOW Program

The Cobol-WOW design tool, the Cobol-WOW Designer, defines the user interface. The user interface logic is included in the form and the Designer generates only the **.wws** and **.wpr** copy files. The Designer no longer continues to regenerate the skeleton program.

In fact, the skeleton program has now become something much more. It has become the repository for the application logic. The program can be edited with any editor.

Whenever the form is modified, the copy files must be regenerated. The program then can be recompiled.

In some circumstances you may want to edit event-handling code outside the Cobol-WOW Designer. This will work satisfactorily. The Designer will detect the changes during the editing session and preserve the modifications.

How a Cobol-WOW Program Works

You can understand how a Cobol-WOW program works by looking at four files: **windows.cpy**, *formname*.**wws**, *formname*.**cbl**, and *formname*.**wpr**.

WINDOWS.CPY

The **windows.cpy** copy file, supplied with the Cobol-WOW DLL, declares the data items needed to interface to Windows. Windows was created to recognize many numerical constants. This file declares these values as COBOL data items with names that are meaningful and consistent with Windows programming constructs. This file should never be modified.

Let's examine a few of the data items declared in **windows.cpy**.

Many Windows API functions require a true or false value. In Windows, TRUE = 1 and FALSE = 0. Because TRUE and FALSE have an entirely different meaning in COBOL, the **windows.cpy** file includes the following declaration:

```
01 WIN-BOOLEAN-VALUES.
    03 WIN-TRUE        PIC 9(4) COMP-4 VALUE 1
    03 WIN-FALSE       PIC 9(4) COMP-4 VALUE 0.
```

In writing Cobol-WOW programs, you can use WIN-TRUE for TRUE and WIN-FALSE for FALSE. In the following examples, the first line of code enables a window; the second disables it.

```
CALL ENABLEWINDOW USING WIN-RETURN WND-H WIN-TRUE.
CALL ENABLEWINDOW USING WIN-RETURN WND-H WIN-FALSE.
```

The **windows.cpy** file also declares the data items needed to store Windows messages.

```
01 WIN-MSG-WS.

    03 WIN-MSG-HANDLE        PIC 9(10) COMP-4.
    03 WIN-MSG-HANDLE-A REDEFINES WIN-MSG-HANDLE PIC X(8).
    03 WIN-WPARAM-H          PIC S9(10) COMP-4.
    03 WIN-WPARAM-L          PIC S9(10) COMP-4.
    03 WOW-KEY-VALUE REDEFINES WIN-WPARAM-L PIC 9(10) COMP-4.
    03 WIN-LPARAM            PIC S9(10) COMP-4.
    03 WIN-LPARAM-A REDEFINES WIN-LPARAM PIC X(8).
    03 WIN-MSG-ID            PIC 9(10) COMP-4.
    03 WIN-MSG-ID-RED REDEFINES WIN-MSG-ID.
       05 FILLER                    PIC X(6).
       05 WIN-MSG-ID-A          PIC XX.
```

This file also contains the declarations for all the Windows API functions and messages that can be used with Cobol-WOW.

FORMNAME.WWS

As discussed in the section "Initial Creation of a Cobol-WOW Program" on page **199**, the *formname***.wws** copy file contains a binary definition of a form. It also contains special variables for use in event-handling code.

Create a sample form with the Name property CUSTINFO and two edit controls, CUST-NAME and CUST-ADDRESS. The first data item you will see in the **custinfo.wws** copy file is the data item used to store the handle of the form after it is created. This handle is needed to perform operations on the form, such as hiding or disabling it. For example:

```
01 CUSTINFO-H          PIC 9(5) COMP-4 VALUE 0.
```

The next data items in **custinfo.wws** define the ID numbers of the controls on the form. If the form contains a pulldown menu, ID numbers for the menu controls would also be defined. These ID numbers are required for some Windows API functions and for the event-handling code generated by the Cobol-WOW Designer. For example:

```
01 CUSTINFO-IDS.
    03 CUST-NAME-ID     PIC 9(5) COMP-4 VALUE 1.
    03 CUST-ADDRESS-ID  PIC 9(5) COMP-4 VALUE 2.
```

You will then see the data items that contain the handles of the individual controls on the form after the form is created. The handles are required for most Windows API functions and for the event-handling code generated by the Cobol-WOW Designer. For example:

```
01 CUSTINFO-HS.
    03 CUST-NAME-H      PIC 9(5) COMP-4 VALUE 0.
    03 CUST-ADDRESS-H   PIC 9(5) COMP-4 VALUE 0.
```

Note All handles are initialized with the value 0, while the IDs are initialized with the correct values. The handle data items will receive values when the form is created.

Finally, you will see the data item that is the binary definition of the form. It may or may not include comments describing the form contents, depending on how the form was generated. This definition begins with the following:

```
01 CUSTINFO-DEF.
```

FORMNAME.CBL

The *formname*.cbl file is the COBOL skeleton program file generated by the Cobol-WOW Designer. This program contains the logic necessary to create, use, and destroy the form.

The Working-Storage Section of the skeleton program contains only the two copy files, described on page **200**. The Procedure Division is more complex.

The high-level control of the skeleton program is MAIN-FUNCTION.

The first statement in MAIN-FUNCTION is as follows:

```
PERFORM PROGRAM-INITIALIZATION.
```

This procedure contains only an EXIT statement. It is a placeholder intended to indicate a place in the program where you could open files or perform other initialization associated with the application logic.

The second statement in MAIN-FUNCTION

```
PERFORM CREATE-WINDOWS.
```

executes the following procedure:

```
CREATE-WINDOWS.
    PERFORM WOW-CREATE-FORMNAME.
```

This procedure creates any form that should be created at the start of the program. In the currently generated skeleton program, you have only one form, even though more could be added. (The procedure, WOW-CREATE-FORMNAME, is declared in *formname*.wpr, discussed on page **207**.)

The third and fourth statements in MAIN-FUNCTION

```
SET WOW-QUIT TO FALSE.
PERFORM PROCESS-EVENTS UNTIL WOW-QUIT.
```

combine to execute PROCESS-EVENTS until the condition WOW-QUIT. The event handling for the form is performed in a loop, which is terminated only when this condition is set to TRUE by some part of the event-handling code, such as the Quit option on a File menu or a Cancel button.

The fifth statement in MAIN-FUNCTION

```
PERFORM DESTROY-WINDOWS.
```

executes the following procedure:

```
DESTROY-WINDOWS.
    PERFORM WOW-DESTROY-FORMNAME.
```

This procedure destroys any forms that were created at the start of the program. Although there is only one form, more may be added. (The procedure, WOW-DESTROY-FORMNAME, is declared in *formname*.**wpr**, described on page **207**.)

The sixth statement in MAIN-FUNCTION

```
PERFORM PROGRAM-SHUTDOWN.
```

executes a procedure that contains only an EXIT statement. It serves as a placeholder to indicate a place in the program where you can close files or do other cleanup associated with the application logic.

The last statement in MAIN-FUNCTION is GOBACK, which exits the program.

PROCESS-EVENTS is the other procedure in *formname*.**cbl** that you should examine.

```
PROCESS-EVENTS.
    CALL WOWGETMESSAGE USING WIN-RETURN WIN-MSG-WS
         ACTIVEXX-EVENT-WS.
    EVALUATE WIN-MSG-HANDLE
         WHEN FORMNAME-H PERFORM WOW-FORMNAME-EVENTS
    END-EVALUATE.
```

This procedure retrieves the Windows messages from the message queue and dispatches them to the appropriate form. The CALL statement retrieves the message information. The EVALUATE statement checks the message handle against the handle of each form used by the program and performs WOW-FORMNAME-EVENTS for the appropriate form. (Remember, there is only one form in the skeleton program.)

The procedure, WOW-FORMNAME-EVENTS, declared in *formname*.**wpr**, is discussed in the following section.

FORMNAME.WPR

The *formname*.wpr copy file contains the event-handling code defined for the form.

Let's examine the form you created with the name CUSTINFO, two edit controls, and an OK command button called OK-BTN, with code attached to the Click event for the command button.

This first items in the **custinfo.wpr** copy file are the declarations of all the event-handling code defined for the form. The following procedure is responsible for connecting the event-handling code to the correct Windows message:

```
WOW-CUSTINFO-EVENTS.
    EVALUATE WIN-MSG-ID
          WHEN WM-COMMAND
                EVALUATE WIN-LPARAM-L
                      WHEN OK-BTN-H OF CUSTINFO-CTL-HS
                            EVALUATE WIN-LPARAM-H
                                  WHEN BN-CLICKED PERFORM
OK-BTN-CLICK
                                  END-EVALUATE
                END-EVALUATE
    END-EVALUATE.
```

This procedure evaluates a Windows message and compares the parameters to those of the form and its controls. When the procedure finds a message that corresponds to an event with event-handling code, it performs that event-handling code, in this case, OK-BTN-CLICK. The size and complexity of this procedure will vary greatly depending upon the size and complexity of the form. This code is generated entirely by the Cobol-WOW Designer, and since it is built on the EVALUATE statement, the execution time does not degrade as additional controls and events are added.

The next item in the form procedure copy file, **custinfo.wpr**, is the procedure that creates the form with the controls defined in the Designer. This procedure also loads the handles for each of the form's controls into CUSTINFO-CTL-HS.

```
WOW-CREATE-CUSTINFO.
    INITIALIZE WIN-STYLE.
    CALL WOWCREATEWINDOW USING CUSTINFO-H
          CUSTINFO-DEF WIN-STYLE 0 CUSTINFO-CTL-HS.
```

Finally, you will see the procedure that destroys the form:

```
WOW-DESTROY-CUSTINFO.
    CALL DESTROYWINDOW USING WIN-RETURN CUSTINFO-H.
```

How a Cobol-WOW Program Works with Windows

The Cobol-WOW DLL adds a thin layer of logic between the COBOL runtime system and Windows, which makes Windows presume that it was designed to work with COBOL. This thin layer of logic processes function calls and messages to make them "feel" like COBOL (on the COBOL side) and feel like C (on the Windows side).

The following figure illustrates the flow of this process.

Execution of a Cobol-WOW Program

You can see that the COBOL program does not directly communicate with Windows. When you call Windows, the call goes to the Cobol-WOW DLL, then the Cobol-WOW DLL calls Windows. Notice that the arrow goes in both directions between Windows and the Cobol-WOW DLL.

The Windows operating system was designed to call application code directly in order to pass messages to a program. Although Windows cannot call the interpretive COBOL code, it can call the Cobol-WOW DLL and gives it the messages. The Cobol-WOW DLL stores the messages in a message queue and gives them to the COBOL program when the WOWGETMESSAGE function is executed.

In addition to receiving messages, this approach also provides Cobol-WOW with the ability to encapsulate the event-driven architecture of Windows within the traditional structure of COBOL programs. Instead of making the programs respond to events at all times, the program can choose when to go into event-driven operation and when to sequentially process operations like traditional COBOL programs. By preserving the type of set-up and shut-down logic typically used by COBOL programs, it is easier to create report and posting programs, and to migrate legacy programs.

Using Cobol-WOW Programs with Non-Cobol-WOW COBOL Programs

How do Cobol-WOW programs coexist with non-Cobol-WOW programs? Since Cobol-WOW programs are regular COBOL programs, there are several issues to consider.

Calling To and From Cobol-WOW Programs

Cobol-WOW programs can be called by, as well as call, legacy COBOL programs. Cobol-WOW programs can be passed Linkage Section parameters, and can pass Linkage Section parameters to legacy COBOL programs. The following figure illustrates this process.
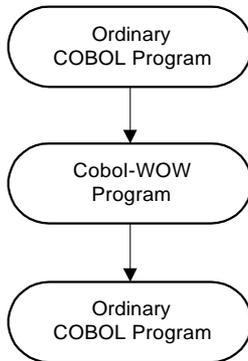
Cobol-WOW Program Calling and Called by a Non-Cobol-WOW COBOL Program

Because Cobol-WOW programs do not require any special Linkage Section parameters, they can be plugged into legacy applications and called by legacy programs as easily as any other COBOL program. Additionally, since Cobol-WOW programs can call legacy programs, existing utility programs and subroutine programs can be called in the same manner as they are called by legacy programs.

Visual Considerations of Cobol-WOW and Non-Cobol-WOW Programs

All non-Cobol-WOW programs use the standard COBOL main window to display and enter information. Cobol-WOW programs create their own windows. These windows will not interfere with each other; in fact, a Cobol-WOW program can also display information in the standard COBOL main window with a DISPLAY statement. The standard COBOL main window can also be hidden and displayed using the C$SHOW subprogram. For more information about the C$SHOW subprogram, see the *RM/COBOL User's Guide*.

Appendix D: Using Cobol-WOW with RM/Panels

This appendix describes how to use Cobol-WOW with RM/Panels to enhance existing panel libraries and also discusses how to migrate panels to Cobol-WOW forms.

Enhancing Existing Panel Libraries

For those RM/Panels users who would like to improve their Windows presentation without modifying application code, the Cobol-WOW Designer can be used to enhance existing panels to use the full spectrum of Windows fonts and colors. Developers are no longer restricted to using a single, fixed-width font nor to a limited color palette.

These enhanced panels can be used by existing programs without source code changes, simply by using the RM/Panels runtime (supplied with RM/COBOL for Windows, version 7.00.02 and higher) supplemented with a Cobol-WOW panel runtime-based DLL (**wowpanrt.dll**). Enhancing a panel in this manner for Windows does not limit portability, or prevent the panel or panel library from continued use in DOS or UNIX. Nor is it necessary to enhance every panel in the application.

Key features for enhancing existing panel libraries include the following:

- Ability to edit RM/Panels screens with the Cobol-WOW Designer.

- No need to change RM/Panels source code.

- Cobol-WOW editing of panels does not create additional files.

- Ability to test panels from the Cobol-WOW Designer.

For more information about opening and modifying an existing panel, see page **213**.

Character-Based GUI Portability and Cross Development

The move to a full graphical user interface (GUI) does not sacrifice the ability to continue to deploy an application in a character-based form. However, there are some issues that must be considered in order to continue application development with an interface optimized for both environments.

When a panel is enhanced, the data fields/controls added to the panel (see page **215**) will be present on both the character-based and graphical representations of the panel. There are some properties of the controls that are specific to either the graphical or character-based environment. For example, each field/control has a Line and Column property, and Top and Left properties. The Top and Left properties are used in Windows only. The Line and Column properties are used in DOS and UNIX only.

The character-based RM/Panels Library Manager does not allow access to the Windows-only properties. The Cobol-WOW Designer, however, does allow you to edit the character-only properties. The effects of the character-only properties are not visible from the Cobol-WOW Designer.

New fields/controls can be added to a panel using either the RM/Panels Library Manager or the Cobol-WOW Designer. Theoretically, if you added a control in the Cobol-WOW Designer, you could set its character properties and immediately use the panel in the character environment. Practically speaking, though, you will want to edit the panel using the RM/Panels Library Manager to accurately tailor the panel before deploying the panel in the character environment.

Note After adding controls to a panel using the RM/Panels Library Manager, you will have to edit the panel using the Cobol-WOW Designer before running a Cobol-WOW enhanced panel in RM/Panels. Failure to do so will result in the panel being displayed without the Cobol-Wow enhancements.

Communicating with RM/Panels

The Cobol-WOW Designer must interface to the RM/Panels COBOL programs. This is done via TCP/IP using Cobol-RPC, which paves the way for client/server implementations. Two files, which are included with Cobol-WOW, are specifically required to handle this communication. These files are **rmrpc32s.dll** and **cobolrpc.ini**.

The Cobol-RPC DLL, **rmrpc32s.dll**, allows the RM/Panels COBOL programs to be invoked by the Cobol-WOW Designer. When the Designer needs to invoke a COBOL program, it starts a new process. This process executes the RM/COBOL runtime and includes **rmrpc32s.dll** on the command line with the L= option.

The Windows initialization file, **cobolrpc.ini**, contains configuration information for **rmrpc32s.dll**. If problems occur, the following two entries can be manually changed:

```
[ServerConfig]
Port=5000
StartupCommand=runcobol rpcinit.cob l=rmrpc32S.DLL l=wowpan.obj
```

The Port value can be changed if port 5000 is already in use on the system. Any value can be used, but values over 1024 are best.

The StartupCommand entry must be changed if the **runcobol.exe** file is not in the path.

Modifying an Existing Panel Library

The first step in the process is modifying an existing panel to create a more typical Windows "look-and-feel." This process begins the same way as any Cobol-WOW session. The supplied sample library, **sample.lib** (located in cobolwow\pansmple), can be used to follow these exercises exactly.

Open the library

To open an existing panel library, take the following steps:

1. Start the Cobol-WOW Designer.

2. From the Panels menu, click **Open**.

   Any open project is closed automatically and the Open Panel dialog box appears.

Open Panel Dialog Box

3. In the Open Panel dialog box, find the desired panel library and open it. Panel libraries have the extension **.lib**. (For this exercise, you will use **sample.lib**, which, by default, is located in C:\CobolWOW\PANSMPLE\.) The Select Panel dialog box opens.

   Note Cobol-WOW must interface to the RM/Panels COBOL programs via TCP/IP using Cobol-RPC.
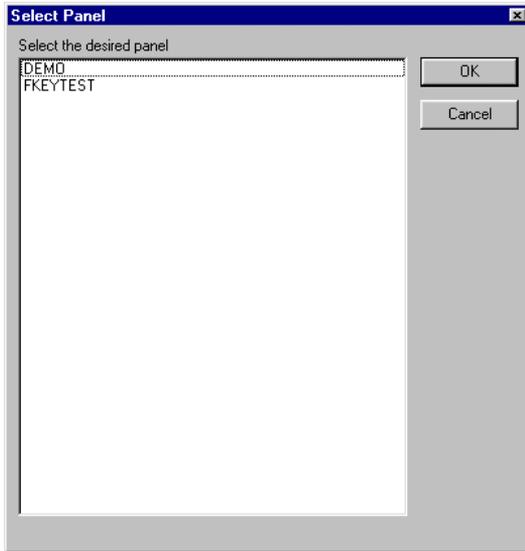
   Select Panel Dialog Box

4. Select the panel to be modified and click **OK**.

   The panel will be opened in the Cobol-WOW Designer. A default graphical representation will be displayed. The size, shape, location, color, fonts, and other properties of the controls and overall window can then be modified.

Change controls

Note RM/Panels refers to the objects called "controls" in Cobol-WOW as "data fields." All types of RM/Panels data fields can be added to the panel using the Cobol-WOW Toolbox.

The properties of each field/control are displayed in the Cobol-WOW Properties dialog box. Each of the properties listed can be modified. Some properties, such as Column and Line, affect only the character implementation of the panel. Others, such as ForeColor, affect only the Windows implementation.

Remember that it is possible to modify several fields/controls at once by selecting multiple fields/controls, and using the Background Color, Foreground Color, and Font options from the Control menu.

Add controls

Note RM/Panels refers to the objects called "controls" in Cobol-WOW as "data fields." All types of RM/Panels data fields can be added to the panel using the Cobol-WOW Toolbox.

When you open a panel in Cobol-WOW, the Cobol-WOW Toolbox automatically displays only the RM/Panels data fields (see page **217**) that can be added to the panel. All types RM/Panels fields/controls can be added to the panel using the Cobol-WOW Toolbox. If you add a field/control to a panel, you will want to use the Character Panel Editor to adjust the size, location, and color of the field/control before executing the panel in a character-based environment.

Note ActiveX controls (see page **191**) cannot be added to an enhanced panel. The following intrinsic controls (see page **87**) also cannot be added to an enhanced panel: animation, date time picker, IP address, month calendar, progress bar, status bar, tab, timer, toolbar, trackbar, and updown, as well as any shapes (ellipse, line, rectangle, and rounded rectangle). Date time picker, IP address and month calendar are not available.

Delete controls

Note RM/Panels refers to the objects called "controls" in Cobol-WOW as "data fields." All types of RM/Panels data fields can be added to the panel using the Cobol-WOW Toolbox.

Panel fields/controls may be deleted from within the Cobol-WOW Designer. The modified **.ws** file will be generated automatically when the panel is saved. Be sure to recompile any programs that use the panel.

Save a panel

An enhanced panel can be saved at any time during the editing session using the Save command on the Panels menu. When the enhanced panel is saved in this manner, the description of the panel is written back into the standard panel library. No Cobol-WOW-specific files are created. The panel library must be used to re-open the panel in the Cobol-WOW Designer and to operate the panel at runtime.

When the enhanced panel is saved, the panel copy files are automatically (re)generated. This is a slight variation in behavior from the DOS or UNIX versions of RM/Panels v2

where the copy files could be generated optionally. Always generating these files preserves the integrity of the relationship between the copy files and the actual panel definition in the RM/Panels library and helps prevent undesirable problems, such as 104 errors.

The copy file generation is done based on the definition in the panel library (maintained through the RM/Panels Library Manager's Code Generation dialog box). This includes the path used for placing the generated files.

If you do not wish to save your edits, use the **Recreate GUI** command on the Panels menu.

Test a panel

The Test command on the Panels menu enables testing of panels during editing. There is a small difference from the way in which previous versions of Cobol-WOW performed this task. Before testing the panel, any editing changes that have been made are permanently saved to the panel library.

Run an application with an enhanced panel

To use the enhanced panel, take the following steps:

1. Run the program using **runpan2.cob**, which is shipped with Cobol-WOW v2.26 and higher.

2. Load the Cobol-WOW panel runtime (**wowpanrt.dll**) by adding the following line to the command line:

        l=wowpanrt.dll

Any panel that has been edited with Cobol-WOW will be displayed using the full Windows appearance. Any panels that have not been edited with Cobol-WOW will continue to be displayed in the same manner as RM/Panels v2.*x*.

Setting Properties for RM/Panels Data Fields

RM/Panels refers to the objects called "controls" in Cobol-WOW as "data fields." All types RM/Panels data fields can be added to the panel using the Cobol-WOW Toolbox.

Data fields/controls have a number of configurable characteristics. These characteristics are called properties. Properties are the primary means by which fields/controls are manipulated. Setting properties defines how fields/controls are displayed and how they function in the running application.

When you open a panel in the Cobol-WOW Designer, you use the Properties dialog box (see page **8**), which lists each property and its value, to set the default (initial) properties of a selected field/control.

The following list summarizes the data fields found in the Toolbox when you open an RM/Panels panel in the Cobol-WOW Designer.

- Check Box. Displays a Yes/No, True/False, or On/Off option. You can check any number of check boxes on a form at one time.

- Combo Box. Combines a text box with a list box. Allows a user to type in a selection or select an item from a drop-down list.

- Command Button. Carries out a command or action when a user chooses it.

- Date Edit Box. Provides an area in which a date can be displayed or entered.

- Edit Box. Provides an area to enter or display text.

- Group Box. Provides a visual and functional container for other controls. It is generally used to enclose related controls (usually check boxes or option buttons).

- List Box. Displays a list of choices from which the user can select one or more items.

- Multi-Line Edit Box. Provides a small, fixed space into which a user can enter several lines of text, a portion of which is hidden until the user scrolls its contents using scroll bars.

- Numeric Edit Box. Provides an area to input or display numeric data.

- Option Button. Presents mutually exclusive options in an option control. Option buttons are usually used with the group box control to form groups where only one of the listed buttons can be selected at one time.

- Scroll Bar. Allows a user to add scroll bars to controls that do not automatically provide them. (These are not the same as the built-in scroll bars that are found with many controls.)

- Static Text. Displays text, such as titles or captions, in regular outlines or filled rectangles, that the user cannot interact with or modify.

- Time Edit Box. Provides an area in which the time can be displayed or entered.

Check Box Field/Control

To add a check box field/control to a panel/form, click **Check Box** from the Toolbox.

The check box data field/control displays an option that can be turned on or off. The check box is similar to the command button, in that the primary method of operation is clicking it. The check box, however, represents data, not a request for action.

All of the properties for this field/control are listed in the following table. For a description of these properties, see "Common Data Field Properties" on page **241**.

Properties

3D	EntryOrder	ForeColor	PromptText
Accelerator	ErrorMessage	Height	SelectedAttr
BackColor	FontBold	HelpMessage	StartOfGroup
Beep	FontItalic	Left	TimeOut
Column	FontName	Length	TimeOutValue
DefaultToPressed	FontSize	Line	Title
DisabledAttr	FontStrikethru	MnemonicAttr	Top
EnabledAttr	FontUnderline	Name	Width

Combo Box Field/Control

To add a combo box field/control to a panel/form, click **Combo Box** from the Toolbox.

The combo box field/control combines the list selection capability of a list box with the edit box's ability to type in a value. Alternatively, to save screen space, you may wish to show only a portion of the list box's selections. And, there may be instances when you would like to display the currently selected item in a static text box area when the entire list is not displayed. The combo box control can perform both tasks.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	CurChoice	FontSize	PromptText
BackColor	DisabledAttr	FontStrikethru	ScrollBar
Beep	DoubleClick	FontUnderline	SelectedAttr
Border	DropDown	ForeColor	StartOfGroup
BorderAttr	EnabledAttr	Height	StaticChoices
ChoiceHelp	EnabledForInput	HelpMessage	TimeOut
ChoicesToDisplay	EntryOrder	*InputField	TimeOutValue
ChoicesToStore	ErrorMessage	Left	Top
ChoiceValue	FontBold	Length	Width
ChoiceWidth	FontItalic	Line	
Column	FontName	Name	

InputField Property

The InputField property determines whether an input field is to be attached to a list box, making the field/control a combo box.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the InputField property:

Value	Description
False	An input field/control is not attached to the list box.
True	An input field/control is attached to the list box (the default).

Command Button Field/Control

To add a command button field/control to a panel/form, click **Command Button** from the Toolbox.

The command button (also known as push button) field/control causes an action to occur when the user either clicks the button or presses a key.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	EntryOrder	Height	SelectedAttr
Accelerator	ErrorMessage	HelpMessage	*SizeType
BackColor	FontBold	Left	*SizeValue
Beep	FontItalic	Length	StartOfGroup
Column	FontName	Line	TimeOut
DefaultValue	FontSize	MnemonicAttr	TimeOutValue
DisabledAttr	FontStrikethru	Name	Title
EnabledAttr	FontUnderline	PromptText	Top
EnabledForInput	ForeColor	*PushedAttr	Width

PushedAttr Property

The PushedAttr property indicates the attribute that should be used to display the command button while it is depressed. Valid values are blank (the library default) or A through P (as defined by the attribute code assigned to a block of text within a panel).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

SizeType Property

The SizeType property specifies the size of the button displayed for the command button.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the SizeType property:

Value	Description
Auto	The button is just wide enough to hold the title.
Small	Refers to the default set for the panel library.
Medium	Refers to the default set for the panel library.
Large	Refers to the default set for the panel library.
Explicit	You can enter a number that specifies the size of the button in characters.

SizeValue Property

The SizeValue property specifies the width of the button in character positions. This value is valid only if the SizeType property value is set to explicit (see above).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Date Edit Box Field/Control

To add a date edit box field/control to a panel/form, click **Date Edit Box** from the Toolbox.

The date edit box field/control provides an area on a panel/form in which a date can be displayed or entered.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following

sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	DoubleClick	FontUnderline	OccYOffset
AlwaysDisabled	EnabledAttr	ForeColor	PromptText
AutoExit	EnabledForDisplay	Height	Protected
BackColor	EnabledForInput	HelpMessage	SelectedAttr
Beep	EntryFormat	Left	StartOfGroup
BlankWhenZero	EntryOrder	Length	*StorageFormat
Border	ErrorMessage	Line	TimeOut
Column	FontBold	Name	TimeOutValue
DefaultToSystem	FontItalic	OccColOffset	Top
DefaultValue	FontName	OccLineOffset	Update
DisabledAttr	FontSize	Occurrences	Validation
DisplayFormat	FontStrikethru	OccXOffset	Width

StorageFormat Property (Date Edit Box)

The StorageFormat property specifies the format to be used when storing this field/control, based on years, months, and days, which are represented as follows:

- YYYY is a four-digit numeric representation of the year (for example, 2001)

- YY is a two-digit numeric representation of the year (for example, 01)

- MM is a two-digit numeric representation of the month (for example, 12)

- DD is a two-digit numeric representation of the day (for example, 30)

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the StorageFormat property for a date edit box field/control:

Value	Description
1	Date is displayed as YYYYMMDD.
2	Date is displayed as YYMMDD.
3	Date is displayed as MMDDYYYY.
4	Date is displayed as MMDDYY.

Edit Box Field/Control

`ab` To add an edit box field/control to a panel/form, click **Edit Box** from the Toolbox.

The edit box field/control provides an area to input or display text. This field/control replaces the COBOL ACCEPT statement. The user can enter any type of alphanumeric data in an edit box, including numeric data. Because no formatting is provided, numbers are entered in the same manner as text.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	EnabledAttr	Height	Prompt
AlwaysDisabled	EnabledForDisplay	HelpMessage	PromptText
AutoExit	EnabledForInput	*Justify	Protected
BackColor	EntryOrder	Left	SelectedAttr
Beep	ErrorMessage	Length	StartOfGroup
Border	FontBold	Line	TimeOut
Case	FontItalic	Name	TimeOutValue
*Class	FontName	OccColOffset	Top
Column	FontSize	OccLineOffset	Update
DefaultValue	FontStrikethru	Occurrences	Validation
DisabledAttr	FontUnderline	OccXOffset	Width
DoubleClick	ForeColor	OccYOffset	

Class Property

The Class property indicates the categories that RM/Panels allows for defining character sets. Valid values are blank or 1–5.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Justify Property

The Justify property indicates whether left, right, or center justification is required. This affects user input and values placed into the field/control with a MOVE statement.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Prompt Property

The Prompt property indicates whether prompt characters are to be provided for this field/control during input.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Group Box Field/Control

To add a group box field/control to a panel/form, click **Group Box** from the Toolbox.

The group box is a specialized box that is used to group other fields/controls, such as check boxes and option buttons.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following

sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	FontItalic	ForeColor	Name
BackColor	FontName	*Group	*TabStop
Caption	FontSize	Height	Top
*Enabled	FontStrikethru	Left	Width
FontBold	FontUnderline	*Locked	

Enabled Property

The Enabled property determines whether the field/control can respond to user-generated input (or events).

The following table lists the possible values of the Enabled property:

Value	Description
False	The field/control is disabled for user input.
True	The field/control is enabled for user input (the default).

Group Property

The Group property determines whether a field/control is the start of a group.

The following table lists the possible values of the Group property:

Value	Description
False	The field/control is not the start of a group (the default).
True	The field/control is the start of a group.

Locked Property

The Locked property determines whether or not a lock is placed on the field/control in order to prevent the field/control from being moved accidentally on the form.

The following table lists the possible values of the Locked property:

Value	Description
False	The field/control is not locked (the default).
True	The field/control is locked.

TabStop Property

The TabStop property is described on page **174**.

List Box Field/Control

To add a list box field/control to a panel/form, click **List Box** from the Toolbox.

The list box field/control allows the selection of one or several items from a list of items.

All of the properties for this field/control are listed in the following table. For a description of these properties, see "Common Data Field Properties" on page **241**.

Properties

3D	CurChoice	FontSize	ScrollBar
BackColor	DisabledAttr	FontStrikethru	SelectedAttr
Beep	DoubleClick	FontUnderline	StartOfGroup
Border	DropDown	ForeColor	StaticChoices
BorderAttr	EnabledAttr	Height	TimeOut
ChoiceHelp	EnabledForInput	HelpMessage	TimeOutValue
ChoicesToDisplay	EntryOrder	Left	Top
ChoicesToStore	ErrorMessage	Length	Width
ChoiceValue	FontBold	Line	
ChoiceWidth	FontItalic	Name	
Column	FontName	PromptText	

Multi-Line Edit Box Field/Control

To add a multi-line edit box field/control to a panel/form, click **Multi-Line Edit Box** from the Toolbox.

Sometimes you need to store a lot of text, but you do not want to use up a lot of screen space in your display. You can create a multi-line edit field/control that lets users enter several lines of text into a small, fixed space. Because the text box is smaller than the amount of information stored, part of the information is hidden. The user uses scroll bars to display the hidden information. You can design the text box to scroll its contents vertically or horizontally.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	EnabledAttr	Height	SelectedAttr
BackColor	EnabledForInput	HelpMessage	StartOfGroup
Beep	EntryOrder	Left	*Stream
Border	ErrorMessage	Length	TimeOut
Case	FontBold	Line	TimeOutValue
*ColsToDisplay	FontItalic	*LinesToDisplay	Top
*ColsToStore	FontName	*LinesToStore	Width
Column	FontSize	Name	*Wrap
DefaultValue	FontStrikethru	PromptText	
DisabledAttr	FontUnderline	Protected	
DoubleClick	ForeColor	*Required	

ColsToDisplay Property

The ColsToDisplay property specifies the number of columns to display for the field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ColsToStore Property

The ColsToStore property specifies the number of columns to store for the field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

LinesToDisplay Property

The LinesToDisplay property specifies the number of lines to display for the field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

LinesToStore Property

The LinesToStore property specifies the number of lines to store for the field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Required Property

The Required property determines whether the user must enter data into field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the Required property:

Value	Description
False	The field/control is not required (the default).
True	The field/control is required.

Stream Property

The Stream property indicates that insert and delete operations should affect the entire field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the

appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the Stream property:

Value	Description
False	Insert and delete operations should not affect the entire field/control (the default).
True	Insert and delete operations should affect the entire field/control.

Wrap Property

The Wrap property indicates that words automatically wrap to the succeeding line when they are typed, inserted, or deleted.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the Wrap property:

Value	Description
False	Words do not automatically wrap to the succeeding line (the default).
True	Words automatically wrap to the succeeding line.

Numeric Edit Box Field/Control

To add a numeric edit box field/control to a panel/form, click **Numeric Edit Box** from the Toolbox.

The numeric edit box field/control provides an area to input or display numeric data.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	DisplayFormat	FontUnderline	OccYOffset
AlwaysDisabled	DoubleClick	ForeColor	PromptText
*AssumeDecimal	EnabledAttr	Height	Protected
AutoExit	EnabledForDisplay	HelpMessage	SelectedAttr
BackColor	EnabledForInput	IntegerDigits	*Signed
Beep	EntryFormat	Left	StartOfGroup
BlankWhenZero	EntryOrder	Length	TimeOut
Border	ErrorMessage	Line	TimeOutValue
*CalculatorEntry	FontBold	Name	Top
Column	FontItalic	OccColOffset	Update
DecimalDigits	FontName	OccLineOffset	Validation
DefaultValue	FontSize	Occurrences	Width
DisabledAttr	FontStrikethru	OccXOffset	

AssumeDecimal Property

The AssumeDecimal property specifies that input to this field/control should be assumed to contain decimal digits even if no decimal is present.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the AssumeDecimal property:

Value	Description
False	A decimal is not assumed.
True	A decimal is assumed (the default).

CalculatorEntry Property

The CalculatorEntry property determines whether input to this field/control should be fully formatted while being input, with digits inserting to the left of the decimal point as with a calculator.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the CalculatorEntry property:

Value	Description
Default	The default set for the panel library applies to the field/control.
Yes	Input to the field/control is fully formatted while being input, overriding any default set for the panel library.
NO	Input to the field/control is not fully formatted while being input, overriding any default set for the panel library.

Signed Property

The Signed property specifies whether the field/control includes a plus (+) or minus (-) sign.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the Signed property:

Value	Description
False	The field/control is not signed.
True	The field/control is signed (the default).

Option Button Field/Control

To add an option button field/control to a panel/form, click **Option Button** from the Toolbox.

The option button (also known as radio button) field/control displays an option that can be turned on or off. Option buttons are usually used in groups where turning one button on turns the others off. For more information on grouping option buttons, see page **131**.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	DisabledAttr	FontUnderline	*NumericData
Accelerator	EnabledAttr	ForeColor	PromptText
BackColor	EnabledForInput	Height	SelectedAttr
Column	EntryOrder	HelpMessage	StartOfGroup
*DataItemName	ErrorMessage	IntegerDigits	TimeOut
*DataSigned	FontBold	Left	TimeOutValue
*DataSize	FontItalic	Length	Top
*DataValue	FontName	Line	Width
DecimalDigits	FontSize	MnemonicAttr	
DefaultToPressed	FontStrikethru	Name	

DataItemName Property

The DataItemName property specifies the data item name to be associated with the COBOL representation of this option button group.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DataSigned Property

The DataSigned property specifies whether this field, if numeric, stores signed numbers.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the DataSigned property:

Value	Description
False	The field/control does not store signed numbers.
True	The field/control stores signed numbers (the default).

DataSize Property

The DataSize property specifies the size of the data item.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DataValue Property

The DataValue property specifies the value to be given to the data item representing this group of option buttons when this button is pressed.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

NumericData Property

The NumericData property specifies whether this field/control is a represented by a numeric data item.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible,

however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the NumericData property:

Value	Description
False	The field/control is not represented by a numeric data item.
True	The field/control is represented by a numeric data item (the default).

Scroll Bar Field/Control

To add a scroll bar field/control to a panel/form, click either **Horizontal Scroll** or **Vertical Scroll** from the Toolbox. A horizontal scroll bar displays a horizontal bar that can be used to scroll information. A vertical scroll bar displays a vertical bar that can be used to scroll information.

The scroll bar field/control is used to allow a numeric value to be manipulated as a thumb position on a bar. By specifying the minimum and maximum, the value can be viewed relative to a range of possible values. This value and the scroll bar are often used to scroll the display of other information on a panel. For more information on using scroll bars, see page **139**.

All of the properties for both the horizontal and vertical scroll bar are listed in the following table. Properties that apply only to these fields/controls, or that require special consideration when used with them, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

Border	EnabledForInput	*MaximumValue	*StepSize
Column	EntryOrder	*MinimumValue	*ThumbAttr
DefaultValue	Height	Name	Top
DisabledAttr	Left	*PageSize	Width
EnabledAttr	Line	*Size	

MaximumValue Property

The MaximumValue property specifies the maximum value associated with the scroll bar. Valid values are 0–999.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

MinimumValue Property

The MinimumValue property specifies the minimum value associated with the scroll bar. Valid values are 0–999.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

PageSize Property

The PageSize property specifies the change in value to be associated with clicking on the scroll bar, above or below the thumb object, but not on the end-arrows.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Size Property

The Size property specifies the size of the scroll bar in characters.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

StepSize Property

The StepSize property specifies the change in value to be associated with clicking on the scroll bar end-arrows.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ThumbAttr Property

The ThumbAttr property specifies the attribute code associated with the thumb object of the scroll bar. Valid values are blank (the library default) or A through P (as defined by the attribute code assigned to a block of text within a panel).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Static Text Field/Control

To add a static text field/control to a panel/form, click **Static Text** from the Toolbox.

The static text field/control is used to display text, rectangular outlines, or filled rectangles. The static text control is also used to draw rectangles or outlines to highlight parts of a panel, group controls, or even create a design.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

3D	FontBold	ForeColor	Top
*Alignment	FontItalic	Height	Width
BackColor	FontName	Left	*WordWrap
Caption	FontSize	Length	
Column	FontStrikethru	Line	
*Effect	FontUnderline	*NoPrefix	

Alignment Property

The Alignment property determines how text is positioned in a static text field/control. The Alignment property allows the text of any static text control, not just multiline controls, to be aligned to the right, left, or center of the control.

The following table lists the possible values of the Alignment property:

Value	Description
0	Normal – Performs no justification (the default).
1	Left justifies text.
2	Centers text.
3	Right justifies text.

Effect Property

The Effect property changes a static text field/control into an empty rectangle or a colored group box without text. The color names actually designate one of the Windows configuration options and may not match the color name used.

The Effect property is used to determine the type of static text field/control that is displayed: text, outline, or rectangle. It is important to note that the text of a static text field/control is not displayed when the outline or rectangle effect is selected. When the 3D property (see page **241**) is set to True, the Effect property also has different appearances.

The following table lists the possible values of the Effect property:

Value	Description
0	None – Text is displayed (the default).
1	Draws a rectangle with the window group box color, usually black.
2	Draws a rectangle with the desktop background color, usually gray.
3	Draws a rectangle with the parent window's background, usually white.
4	Draws a black group box.
5	Draws a gray group box.
6	Draws a white group box.

NoPrefix Property

The NoPrefix property determines whether the ampersand (&) character causes the subsequent character to be underlined in a static text control.

The following table lists the possible values of the NoPrefix property:

Value	Description
False	The ampersand character (&) causes next character to be underlined (the default).
True	The ampersand character (&) character is displayed.

WordWrap Property

The WordWrap property determines whether text is wrapped to multiple lines on a static text field/control.

The following table lists the possible values of the WordWrap property:

Value	Description
False	Text is wrapped (the default).
True	Text is not wrapped.

Time Edit Box Field/Control

To add a time edit box field/control to a panel/form, click **Time Edit Box** from the Toolbox.

The time edit box field/control provides an area on a panel in which the time can be displayed or entered.

All of the properties for this field/control are listed in the following table. Properties that apply only to this field/control, or that require special consideration when used with it, are marked with an asterisk (*). These particular items are documented in the following sections. For information on the remaining properties, see "Common Data Field Properties" on page **241**.

Properties

*24HourFormat	DoubleClick	ForeColor	Protected
3D	EnabledAttr	Height	SelectedAttr
AlwaysDisabled	EnabledForDisplay	HelpMessage	StartOfGroup
AutoExit	EnabledForInput	Left	*StorageFormat
BackColor	EntryFormat	Length	TimeOut
Beep	EntryOrder	Line	TimeOutValue
BlankWhenZero	ErrorMessage	Name	Top
Border	FontBold	OccColOffset	Update
Column	FontItalic	OccLineOffset	Validation
DefaultToSystem	FontName	Occurrences	Width
DefaultValue	FontSize	OccXOffset	
DisabledAttr	FontStrikethru	OccYOffset	
DisplayFormat	FontUnderline	PromptText	

24HourFormat Property

The 24HourFormat property specifies whether the field/control displays information in 12-hour or 24-hour format.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the 24HourFormat property:

Value	Description
False	Time is not displayed in 24-hour format, but rather in 12-hour format (the default).
True	Time is displayed in 24-hour format.

StorageFormat Property (Time Edit Box)

The StorageFormat property specifies the format to be used for the storage of this field/control, based on hours, minutes, and seconds, which are represented by HH, MM, and SS, respectively.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the StorageFormat property for a time edit box field/control:

Value	Description
1	Time is displayed as HHMMSS.
2	Time is displayed as HHMM.
3	Time is displayed as HH.

Common Data Field Properties

This section summarizes the common properties that may be implemented in a field/control on a panel/form. Refer to the specific field/control in the preceding sections to determine the unique properties available for the field/control.

The following properties are used by several types of fields/controls.

Properties

3D	CurChoice	FontName	OccYOffset
Accelerator	DecimalDigits	FontSize	PromptText
AlwaysDisabled	DefaultToPressed	FontStrikethru	Protected
AutoExit	DefaultToSystem	FontUnderline	ScrollBar
BackColor	DefaultValue	ForeColor	SelectedAttr
Beep	DisabledAttr	Height	StartOfGroup
Border	DisplayFormat	HelpMessage	StaticChoices
BorderAttr	DoubleClick	IntegerDigits	TimeOut
BlankWhenZero	DropDown	Left	TimeOutValue
Caption	EnabledAttr	Length	Title
Case	EnabledForDisplay	Line	Top
ChoiceHelp	EnabledForInput	MnemonicAttr	Update
ChoicesToDisplay	EntryFormat	Name	Validation
ChoicesToStore	EntryOrder	OccColOffset	Width
ChoiceValue	ErrorMessage	OccLineOffset	
ChoiceWidth	FontBold	Occurrences	
Column	FontItalic	OccXOffset	

3D Property

The 3D property controls the appearance of a field/control. If this property is set to True, the field/control will have a three-dimensional effect.

The following table lists the possible values of the 3D property:

Value	Description
False	A three-dimensional control is not displayed (the default).
True	A three-dimensional control is displayed.

Note The panel/form 3D property settings of 1 (All 3D) and 2 (No 3D) will override the 3D property settings of individual controls. See page .

Accelerator Property

The Accelerator property specifies the accelerator key to be associated with this field/control. The value is an RM/COBOL termination code in the range 1–98. Pressing a key that generates this value while operating the panel is equivalent to pressing the field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

AlwaysDisabled Property

The Always Disabled property indicates that the field/control will never be enabled for input. In a GUI environment, this enables the field/control to be created as a static text control, rather than an edit box, which allows you to control the foreground color, rather than having Windows force a gray text color.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

AutoExit Property

The AutoExit property indicates whether the input cursor should move to the next field/control if the current field/control has had input that is of maximum length, as specified by its Length property (see page **252**).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

BackColor Property

The BackColor property is described on page **167**.

Beep Property

The Beep property determines whether a beep should be sounded when this field/control has input focus.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the Beep property:

Value	Description
Default	The default set for the panel library applies to the field/control.
Yes	A beep sounds when the field/control has input focus, overriding any default set for the panel library.
No	No beep sounds when the field/control has input focus, overriding any default set for the panel library.

BlankWhenZero Property

The BlankWhenZero property causes the field/control to display as blank when the value of the field/control is 0.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Border Property

The Border property determines whether this field/control is to have a border when displayed.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the Border property:

Value	Description
False	A border is not displayed (the default).
True	A border is displayed.

BorderAttr Property

The BorderAttr property determines whether a border will be displayed around a list box or the list box portion of a combo box field/control. Valid values are blank (the library default) or A through P (as defined by the attribute code assigned to a block of text within a panel).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Caption Property

The Caption property is described on page **168**.

Case Property

The Case property determines the case conversion of alphabetic characters entered into an edit box or multi-line edit box field/control.

The following table lists the possible values of the Case property:

Value	Description
0	Mixed – text case is not altered; accepted as typed (the default).
1	Converts all text to lowercase.
2	Converts all text to uppercase.

ChoiceHelp Property

The ChoiceHelp property determines whether a help message is specified for a choice in a list box or the list box portion of a combo box field/control. For example, in a list box displaying country names, "America" might be the list box choice and USAHELP might be the name of the help message.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the ChoiceHelp property:

Value	Description
False	A help message is not displayed (the default).
True	A help message is displayed.

ChoicesToDisplay Property

The ChoicesToDisplay property specifies the number of choices to display in a list box or combo box field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ChoicesToStore Property

The ChoicesToStore property specifies the number of choices to be stored for a list box or the list box portion of a combo box field/control. If the value in the ChoicesToStore property is greater than the value in the ChoicesToDisplay property (see page **245**), a scroll bar is created automatically. If at runtime execution, the list box or combo box does not contain more choices than can be displayed at one time, the scroll bar is disabled. The scroll bar does not change attributes.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ChoiceValue Property

The ChoiceValue property specifies the initial value of a list box or the list box portion of a combo box field/control when it is displayed.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ChoiceWidth Property

The ChoiceWidth property specifies the width of the entry in characters and also the size of the data item in a list box or the list box portion of a combo box field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Column Property

The Column property determines the number of columns that each occurrence of a field/control is offset from the previous occurrence. Valid values are 0 to the maximum width of the panel.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

CurChoice Property

The CurChoice property specifies the subscript of the value in the ChoiceValue property (see page **246**) of a list box or the list box portion of a combo box field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DecimalDigits Property

The DecimalDigits property indicates the number of digits that can be entered to the right of the decimal point in a numeric field.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DefaultToPressed Property

The DefaultToPressed property determines whether this field/control is to default to having the appearance of being pressed.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the DefaultToPressed property:

Value	Description
False	The field/control is not pressed (the default).
True	The field/control is pressed.

DefaultToSystem Property

The DefaultToSystem property causes the default value of the field/control to be set to the system date of the computer.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DefaultValue Property

The DefaultValue property specifies the default value for the field/control that is set if the RM/Panels standard runtime function, INITIALIZE FIELD, is executed.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DisabledAttr Property

The DisabledAttr property determines whether the field/control is disabled for data entry. Valid values are blank (the library default) or A through P (as defined by the attribute code assigned to a block of text within a panel).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DisplayFormat Property

The DisplayFormat property specifies the COBOL picture format to be used when displaying this field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DoubleClick Property

The DoubleClick property indicates whether the double click of a mouse should be reported on the field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the DoubleClick property:

Value	Description
False	A double click is not reported on the field/control (the default).
True	A double click is reported on the field/control.

DropDown Property

The DropDown property specifies that a drop-down list box is supported in a list box or the list box portion of a combo box field/control. A drop-down list box displays only one item until the user takes an action to display the other choices. A drop-down list box appears initially as a rectangular box showing the current choice with a down arrow. When you choose the down arrow, a list of available choices appears.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the DropDown property:

Value	Description
False	A drop-down list box is not supported (the default).
True	A drop-down list box is supported.

EnabledAttr Property

The EnabledAttr property determines whether the field/control is enabled for data entry. Valid values are blank (the library default) or A through P (as defined by the attribute code assigned to a block of text within a panel).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

EnabledForDisplay Property

The EnabledForDisplay property, when marked with an X, indicates that this field/control is enabled to display values.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

EnabledForInput Property

The EnabledForInput property, when marked with an X, indicates that this field/control is enabled to accept data entry.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

EntryFormat Property

The EntryFormat property specifies the COBOL picture format to be used during data entry.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

EntryOrder Property

The EntryOrder property determines the order in which the fields/controls are operated, with 1 being first, 2 being next, and so on. Any number between 1 and 150 is valid. The value cannot be greater than the number of fields/controls on the panel. Be default, the value is calculated and set by RM/Panels, but you may change it. This property is required.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ErrorMessage Property

The ErrorMessage property specifies the error message associated with this field/control. The RM/Panels Message Editor appears when the cursor is on this field/control and you press F3 or double-click the mouse.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Font Bold, FontItalic, FontName, FontSize, FontStrikethru, and FontUnderline Properties

These properties are described beginning on page **169**.

ForeColor Property

The ForeColor property is described on page **170**.

Height Property

The Height property is described on page **171**.

HelpMessage Property

The HelpMessage property specifies the help message associated with this field/control. The RM/Panels Message Editor appears when the cursor is on this field/control and you press F3 or double-click the mouse.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

IntegerDigits Property

The IntegerDigits property indicates the number of digits that can be entered to the left of the decimal point in a numeric field.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Left Property

The Left property is described on page **171**.

Length Property

The Length property specifies the number of characters in the field/control. The values must be in the range of 1 to the maximum width of the panel, as specified by its Width property (see page **259**).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible,

however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Line Property

The Line property indicates the number of lines that each occurrence of the field/control is offset from the previous occurrence. Valid values are 1 to the maximum length of a panel, as specified by its Length property (see above).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

MnemonicAttr Property

The MnemonicAttr property identifies the mnemonic character associated with the field/control.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Name Property

The Name property is described on page **172**.

OccColOffset Property

The OccColOffset property indicates the number of columns that each occurrence of a field/control is offset from the previous occurrence. Valid values are 0 to the maximum width of a panel, as specified by its Width property (see page **259**).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

OccLineOffset Property

The OccLineOffset property indicates the number of lines that each occurrence of a field/control is offset from the previous occurrence. Valid values are 1 to the maximum length of a panel, as specified by its Length property (see page **252**).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Occurrences Property

The Occurrences property indicates the number of times this field/control occurs on the panel/form.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

OccXOffset Property

The OccXOffset property specifies the number of pixels that multiple occurrences of the field/control should be offset from each other horizontally. This property affects the display of the panel only when using Cobol-WOW.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

OccYOffset Property

The OccYOffset property specifies the number of pixels that multiple occurrences of the field/control should be offset from each other vertically. This property affects the display of the panel only when using Cobol-WOW.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the

appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

PromptText Property

The PromptText property specifies the text that is displayed on the panel/form to prompt the user to enter a correct value.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Protected Property

The Protected property, when marked with an X, indicates that while the input cursor moves into this field/control, the value may not be changed by the user.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ScrollBar Property

The ScrollBar property is described on page **173**.

SelectedAttr Property

The SelectedAttr property determines whether the field/control has input focus. Valid values are blank (the library default) or A through P (as defined by the attribute code assigned to a block of text within a panel).

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

StartOfGroup Property

The StartOfGroup property, when marked with an X, indicates that this field/control is the start of a number of fields/controls (for example, a group of option buttons) that is to be treated as a group. A group includes all fields/controls having contiguous entry order numbers until the next StartOfGroup property is encountered.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

StaticChoices Property

The StaticChoices property determines whether the choices in a list box or the list box portion of a combo box field/control are specified on the panel or are supplied by the application program.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the StaticChoices property:

Value	Description
False	Choices in a list box are supplied by the application program (the default).
True	Choices in a list box are specified on the panel.

TimeOut Property

The TimeOut property determines whether this field/control should wait a maximum time for input, when input is needed.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the TimeOut property:

Value	Description
Default	The default set for the panel library applies to the field/control.
Yes	The field/control should wait for input, overriding any default set for the panel library.
No	The field/control should not wait for input, overriding any default set for the panel library.

TimeOutValue Property

The TimeOutValue property specifies the amount of time to wait for input when the TimeOut property is set to Yes or if the default for the panel library is set to Yes.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Title Property

The Title property specifies the text that appears with the field/control. Note that for the check box field/control (see page **218**), the Title property specifies the text that appears to the right of the check box.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Top Property

The Top property is described on page **174**.

Update Property

The Update property indicates whether the current value of this field/control should be updated or completely replaced by new input.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible,

however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the Update property:

Value	Description
Default	The default set for the panel library applies to the field/control.
Yes	The field/control should wait for input, overriding any default set for the panel library.
No	The field/control should not wait for input, overriding any default set for the panel library.

Validation Property

The Validation property specifies the type of validation to be applied upon input to a field/control. The following types of validation are possible:

- List of values. A list of values, separated with commas, for example: 1,4,47. If a space is included as a valid value, it cannot be the last entry on the list.

- Range of values. A range of values, specified by separating the lowest value and the highest value with two periods, for example: 5..30. Ranges are inclusive by default. Ranges can be made exclusive by inserting greater than and less than symbols before the beginning and ending values, for example: >A..<Z.

- Conditions. The following operators can be used to specify conditions:

Equal =	Greater than >
Not equal !=	Not greater than !>
Less than <	Greater than or equal to >=
Not less than !<	Less than or equal to <=

You can combine a list of values, a range of values, and a condition in a single validation by separating them with commas.

When validating date edit box fields/controls (see page), the following special names can be used to validate the field/control against the system date:

- DATE (the system date)

- YEAR (the system year)

- MONTH (the system month)

- DAY (the system day)

DATE is the only name that contains all components of the system date. The following validation allows only the entry of a date greater than the system date: >DATE. The other three names can be used to validate a numeric edit box field/control (see page **230**) against a single component of the system date. The following validation forces entry of a year that is smaller than the system year: <YEAR.

When validating time edit box fields/controls (see page **237**), a special name, TIME, can be used to validate the field/control against the system time.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Width Property

The Width property determines, in pixels, the width of the field/control.

Set the Width property with any value from 0 to the value specified in the Width property of the form less the value specified in the Left property (see page **252**) of the field/control.

Setting Properties for RM/Panels Panels

RM/Panels refers to the objects called "forms" in Cobol-WOW as "panels." Panels are the containers within which you group fields/controls.

Like fields/controls, panels/forms have a number of configurable characteristics called properties. When you open a panel/form in the Cobol-WOW Designer, you use the Properties dialog box (see page **8**), which lists each property and its value, to set the default (initial) properties of a selected panel/form.

The following properties are used by panels/forms:

Properties

3D	BorderType	GeographicMotion	Prefix
BackColor	Description	Height	StoreByName
BackgroundAttr	DropShadow	HelpAttr	Title
Bitmap	EndUserEditing	HelpMessage	Top
BitmapMode	ErrorAttr	Icon	Width
BorderAttr	ErrorMessage	Left	Windowed

3D Property

The 3D property controls the three-dimensional appearance of fields/controls in a panel/form.

Note The form 3D property settings of 1 or 2 will override the 3D property settings of individual fields/controls.

The following table lists the possible values of the 3D property:

Value	Description
0	Mixed — Allows two-dimensional and three-dimensional settings of individual fields/controls in a form (the default).
1	All 3D — Forces all fields/controls to a three-dimensional appearance.
2	No 3D — Forces all fields/controls to a two-dimensional appearance.

BackColor Property

The BackColor property is described on page **178**.

BackgroundAttr Property

The BackgroundAttr property indicates the default attribute code for the background of the panel/form. Valid values are blank (the library default) or A through P.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Bitmap Property

The Bitmap property is described on page **179**.

BitmapMode Property

The BitmapMode property determines how the bitmap is displayed in a panel/form.

Note Single and double borders are identical when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**).

The following table list the possible values of the BitmapMode property:

Value	Description
0	Displays the bitmap in its original size (the default). If the bitmap is smaller than the panel, the remaining space is filled with the background color. If the bitmap is larger than the panel, only the portion of the bitmap that fits inside the panel is displayed.
1	Stretches the bitmap to fit exactly within the panel. This may result in some distortion of the bitmap image, especially if the size difference between the bitmap and the panel is substantial.
2	Tiles bitmap to fit the panel. If BitmapMode is set to Tile, the bitmap, if smaller than the panel, is displayed in a tiled pattern multiple times within the panel.

BorderAttr Property

The BorderAttr property determines whether a border will be displayed around a panel/form if the Windowed property (see page **266**) is set to True. Valid values are blank (the library default) or A through P.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

BorderType Property

The BorderType property specifies the kind of border that will be displayed around the panel/form if the Windowed property (see page **266**) is set to True.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table list the possible values of the BorderType property:

Value	Description
S	Panel is bordered by a single line (the default).
D	Panel is bordered by a double line.
N	Panel has no border.

Description Property

The Description property describes the panel/form and is displayed on the RM/Panels Library Manager screen.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

DropShadow Property

The DropShadow property determines whether a shaded edge should be displayed around the lower and right borders of the panel/form if the Windowed property (see page) is set to True.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the DropShadow property:

Value	Description
False	A shaded edge is not displayed around the windowed panel (the default).
True	A shaded edge is not displayed around the windowed panel.

EndUserEditing Property

The EndUserEditing property determines whether the end-user can edit the panel/form.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible,

however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the EndUserEditing property:

Value	Description
False	The end-user cannot edit the panel.
True	The end-user can edit the panel (the default).

ErrorAttr Property

The ErrorAttr property indicates the default attribute code for error messages. Valid values are blank (the library default) or A through P.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

ErrorMessage Property

The ErrorMessage property specifies the error message associated with this panel/form when the end-user enters an invalid value. The RM/Panels Message Editor appears when the cursor is on this panel and you press F3 or double-click the mouse.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

GeographicMotion Property

The GeographicMotion property determines whether the movement of the cursor between fields/controls on the panel/form during input is based on the field/control sequence number of their physical location on the monitor.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the

appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the GeographicMotion property:

Value	Description
False	Cursor motion is not based on the physical location of fields/controls on the monitor (the default).
True	Cursor motion is based on the physical location of fields/controls on the monitor.

Height Property

The Height property is described on page **182**.

HelpAttr Property

The HelpAttr property indicates the default attribute code for error messages. Valid values are blank (the library default) or A through P.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

HelpMessage Property

The HelpMessage property specifies the help message associated with this panel/form when the end-user requests help. The RM/Panels Message Editor appears when the cursor is on this panel/form and you press F3 or double-click the mouse.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Icon Property

The Icon property is described on page **182**.

Left Property

The Left property is described on page **183**.

Prefix Property

The Prefix property specifies the prefix to be used when generating **.ws** and **.prc** files.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

StoreByName Property

The StoreByName property determines whether fields/controls on the panel/form are stored by name or by sequence number in the generated **.ws** file.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

The following table lists the possible values of the StoreByName property:

Value	Description
False	Fields/controls are stored by sequence number in the generated .ws file.
True	Fields/controls are stored by name in the generated .ws file (the default).

Title Property

The Title property specifies the title to be associated with the panel/form if the Windowed property (see page **266**) is set to True.

Note This property has no effect when running a program with a Cobol-WOW panel runtime that uses a Cobol-WOW DLL (**wowpanrt.dll** or **wowrt.dll**). It is possible, however, to edit this property in the Cobol-WOW Designer in order to tailor the appearance of the application for situations when the panel is not used with a Cobol-WOW-enabled panel runtime.

Top Property

The Top property is described on page **187**.

Width Property

The Width property is described on page **187**.

Windowed Property

The Windowed property determines whether the panel/form should be displayed and removed as a window. The following properties pertain to windowed panels only: BorderAttr (see page **261**), BorderType (see page **261**), DropShadow (see page **262**), and Title (see page **265**).

The following table lists the possible values of the Windowed property:

Value	Description
False	The panel is not displayed and removed as a window.
True	The panel is displayed and removed as a window (the default).

Configuring Function Keys

The following sections compare how to configure function keys with RM/Panels and Cobol-WOW.

How to Configure Function Keys with RM/Panels

With RM/Panels 2.*x*, keyboard input was done through a COBOL ACCEPT statement in the RM/Panels runtime module (**runpan2.cob**). Because of this, the keys that terminated input and the exception numbers generated by those keys were configured through the RM/COBOL configuration file. In this configuration file, a key was specified as terminating input and returning a specific exception number. The following is a sample entry from an RM/COBOL configuration file that assigns an exception value of 27 to the Escape key:

```
TERM-INPUT Action=Screen-Escape      Code=27          ESC
```

When a user pressed the Escape key from an RM/Panels 2.*x* application, the ACCEPT statement terminated and returned the value of 27. **runpan2.cob** stored this exception value in RMP—EXCEPTION-NUMBER. Typically, condition names assigned to RMP—EXCEPTION-NUMBER were then used to determine which function key was pressed. Here is an excerpt from **rmpanels.ws** that shows how this is declared:

```
05 RMP--EXCEPTION-NUMBER    PIC 9(3) VALUE 0.
    88 F10-KEY                 VALUE 10.
    88 ESCAPE-KEY              VALUE 27.
```

In application code, the developer can then do something like the following to take action when the Escape key is pressed:

```
IF ESCAPE-KEY
    PERFORM CANCEL-INPUT.
```

How to Configure Function Keys with Cobol-WOW

Cobol-WOW does not use COBOL ACCEPT statements for keyboard input. The **runpan2.cob** shipped with Cobol-WOW 2.26 and higher uses the Cobol-WOW runtime to receive and monitor the messages generated by Windows, including the keystroke messages. This means that the COBOL runtime system does not do any function key interpretation. This also means that the entries in the RM/COBOL configuration file will have no effect.

Because of this, a different mechanism is required in order to associate exception numbers with keyboard keys. The same mechanism of returning numeric values in RMP—EXCEPTION-NUMBER should be preserved, however, so that application code does not have to be altered. The Cobol-WOW runtime must be notified, for example, that an exception number of 27 is expected when the Escape key is pressed.

This is accomplished through a section added to the Cobol-WOW initialization file (**cblwow.ini**). The [RMPanelsFunctionKeys] section contains entries that specify exception numbers for each key that needs to be detected. The following sample shows how to associate an exception value of 27 with the Escape key:

```
[RMPanelsFunctionKeys]
ESC=27
```

The left half of the entry is the name of the key as labeled on the keyboard. The right half of the entry is the exception number the key should return.

RM/Panels function keys can be configured using the Windows key names or the RM/COBOL key names.

The following sample entries illustrate these approaches. These approaches can be mixed in the same configuration file. The F1 and F2 entries rely on the names used internally by Windows for the keys. The entries WF4 and WF5 are names used by RM/COBOL.

Sample Cobol-WOW Configuration File Entry

```
[RMPanelsFunctionKeys]

; Windows key names
F1=1
F2=2
F3=3
F4=4
F5=5
F6=6
F7=7
F8=8
F9=9
F10=10
ESC=27
LEFT=65
RIGHT=66
UP=52
DOWN=53
ENTER=13
Shift+F1=11
Shift+F2=12
Control+F1=21
Control+F2=22
; RM/COBOL key names
WF4=4
WF5=5
WSFT+WF4=14
WSFT+WF5=15
WCNT+WF4=24
WCNT+WF5=25
```

The **cblwow.ini** file must be present on a system to run a Cobol-WOW-enhanced RM/Panels application so that the function key information can be loaded by the Cobol-WOW runtime. The following examples show corresponding entries between a typical RM/COBOL runtime configuration file and the new [RMPanelsFunctionKeys] section in the **cblwow.ini** file.

Sample RM/COBOL Configuration File Entry

```
TERM-INPUT Action=Screen-Terminate            CODE=13   CR

TERM-INPUT Action=Screen-Terminate            CODE=1    NUL 59

TERM-INPUT Action=Screen-Terminate            CODE=2    NUL 60

TERM-INPUT Action=Screen-Terminate            CODE=3    NUL 61

TERM-INPUT Action=Screen-Terminate            CODE=4    NUL 62

TERM-INPUT Action=Screen-Terminate            CODE=5    NUL 63

TERM-INPUT Action=Screen-Terminate            CODE=6    NUL 64

TERM-INPUT Action=Screen-Terminate            CODE=7    NUL 65

TERM-INPUT Action=Screen-Terminate            CODE=8    NUL 66

TERM-INPUT Action=Screen-Terminate            CODE=9    NUL 67

TERM-INPUT Action=Screen-Terminate            CODE=10   NUL 68

TERM-INPUT ACTION=SCREEN-PREVIOUS-FIELD       CODE=52   NUL 72

TERM-INPUT ACTION=LEFT-ARROW                  CODE=65   NUL 75

TERM-INPUT ACTION=RIGHT-ARROW                 CODE=66   NUL 77

TERM-INPUT                                    CODE=53   NUL 80
```

Using Global Default Property Settings

Many characteristics of the standard (intrinsic) Windows controls that are created are
controlled by the Cobol-WOW defaults, established as global defaults using the Save
Properties command options on the Control, Form, or Options menus, and stored in the
cblwow.ini file. For example, if you have saved global defaults for a static text field,
including font information, all static fields that are created will have that font. The same
applies to background and foreground colors. This applies to all control types, not just
static text fields. Therefore, by manipulating the global default property settings using
commands on the Control, Form, or Options menus in a Cobol-WOW session <u>before</u>
editing the panel with Cobol-WOW, you can eliminate much of the work you would
otherwise have to do manually in the Cobol-WOW Designer to alter font and color
settings. As Cobol-WOW creates the GUI versions of the controls for the first time, it
will follow these defaults.

You may want to establish the FixedSys font as the global default for static text fields.
This will create the panels in the Cobol-WOW Designer with the closest representation of

the existing character layout. However, you will almost certainly want to change this font to something that is more typical of Windows, such as MS Sans Serif.

Restrictions

The following restrictions apply to using panels with Cobol-WOW:

- Since panels that are displayed by Cobol-WOW are displayed in their own windows, COBOL ACCEPT and DISPLAY statements cannot be used to affect these windows. Programs that use ACCEPT and DISPLAY statements should be modified to replace the statements with RM/Panels functions, such as RMP–DF–*fieldname*.

- The RM/COBOL C$ routines for reading and writing to the screen function in the COBOL main window. Cobol-WOW-enhanced panels do not use the COBOL window, so these C$ routines cannot be used with these panels.

- RM/Panels version 2.1 allowed RM/Panels applications to generate panels dynamically. Dynamically-generated panels are not stored in a panel library, however, which means they cannot be opened in the Cobol-WOW Designer. You cannot make dynamic changes to a Cobol-WOW-enhanced panel. Dynamic modifications of panels are not compatible with the Cobol-WOW method of displaying.

Migrating Panel Libraries to Cobol-WOW Forms

For those users who want to take advantage of all the capabilities of Cobol-WOW, it is not necessary to manually recreate your panels as Cobol-WOW forms. You can immediately begin programming with the form using Cobol-WOW. Moreover, RM/Panels panels and Cobol-WOW forms can coexist in the same application, which provides a gradual migration path for those who want it.

Note Panels that are generated dynamically by an RM/Panels application cannot be migrated to Cobol-WOW forms. Dynamically-generated panels are not stored in a panel library, which means they cannot be opened in the Cobol-WOW Designer.

Migrate a Panel Library

To migrate a panel library to Cobol-WOW forms, take the following steps:

1. Start the Cobol-WOW Designer.

2. On the **Panels** menu, click **Open**.

   Any open project will be closed automatically. The Open Panel dialog box appears.

3. From the Open Panel dialog box, find the desired panel library and open it. Panel libraries have the extension **.lib**. Cobol-WOW must interface to the RM/Panels COBOL programs via TCP/IP using Cobol-RPC.

   The Select Panel dialog box opens.

4. Select the panel to be modified and click **OK**.

   The panel will be opened in the Cobol-WOW Designer. A default graphical representation will be displayed. The size, shape, location, color, fonts, and other properties of the controls and overall window can then be modified.

5. Edit the panel as desired.

6. On the **Panels** menu, click **Export**. The panel will be saved as a Cobol-WOW form with the extension **.wow**.

7. On the **Panels** menu, click **Close**.

8. On the **Project** menu, click either **Open** or **New** and open or create the project to which you want to add the form.

   The former panel can be edited in the same manner as any other Cobol-WOW form.

Appendix E: Using Cobol-WOW Thin Client

This appendix describes how to install and use Cobol-WOW Thin Client, which allows the user interface to exist on the Windows client machine and the COBOL program (data processing) to occur on the server.

Understanding Cobol-WOW Thin Client

Cobol-WOW Thin Client provides the ability to execute Cobol-WOW programs in a client/server architecture over a LAN or the Internet. All programs and data reside and execute on the server, but the Windows user interface is presented on a Windows workstation. This client/server implementation is carried out by integrating RPC+ (formerly Cobol-RPC) technology with Cobol-WOW.

From a conceptual standpoint, you can consider a Thin Client application in the following manner. The Windows client workstation executes a small **.exe** program (**tclient.exe**) on Windows that connects to the server. The server, upon receiving this connection request, begins execution of the application on the server. The application runs as a normal RM/COBOL program on the server until a Cobol-WOW function is invoked. All Cobol-WOW functions are intercepted by special logic in the runtime, which routes the requests back to the client, where they are executed. This causes the user-interface to be presented on the client. When the Cobol-WOW function completes execution, control is returned back to the server.

Only a few files are installed on the client workstation. These files allow the client to initiate the connection to the server and to carry out the Windows user interface functionality. The bulk of the installation is on the server. The server must host the facilities for receiving the connection request, executing the application, and forwarding the Windows user interface requests to the client. For more information on installing and configuring Cobol-WOW Thin Client, see page .

Benefits of Cobol-WOW Thin Client

Cobol-WOW Thin Client provides benefit in a variety of ways, including:

- **Simplified management.** Simplified computing means lower ownership costs and increased resource efficiency of each end-user.

- **Access to legacy systems.** Extends the life of a COBOL application. Customers can retain the access to existing legacy systems, databases, and applications, while benefiting from popular, Windows-based applications.

- **Reduced cost of ownership.** Thin clients do not require many of the features of a PC because network servers do most of the work running programs and storing data.

Installing and Configuring Cobol-WOW Thin Client

To use Cobol-WOW Thin Client, you must install both the Cobol-WOW Thin Client and the RPC+ server software. These are supplied on different distribution disks and must be installed individually. Both may be installed on the same computer, allowing that computer to function as both client and server. This is useful for testing and debugging purposes and an application can be deployed in this manner as well.

Please refer to the installation instructions included on the distribution media for specific instructions on installing the client and server software.

Once Cobol-WOW Thin Client is installed, some configuration must be done before it can be used. Since using the Thin Client portion of Cobol-WOW involves little or no additional coding, configuration of the client and server are the primary issues in its use. Configuration information for both the client and server are stored in the **cobolrpc.ini** file. Configuration information can be changed by editing this file.

For more information on configuring Cobol-WOW Thin Client, see one of the following topics:

- Files Installed on the Windows Client Workstation

- Files Installed on a Windows Server

- Files Installed on a UNIX Server

Files Installed on the Windows Client Workstation

The following list describes each file that must be installed on the Windows client workstation in order to use the Thin Client portion of Cobol-WOW.

Files	Description
tclient.exe	The Cobol-WOW Thin Client executable program. It is the module that must be executed to begin the Thin Client session. It will load the required DLLs and read the configuration file, RpcPlus.ini. The tclient.exe file may be placed in any location.
wowrt.dll	The same Cobol-WOW runtime DLL that is used with standalone Cobol-WOW programs. Instead of being invoked by the RM/COBOL runtime, it is invoked by tclient.exe. This DLL must be placed in the same (or working) directory as tclient.exe or in a directory specified in the PATH environment variable.
RpcPlusRM.dll	The RM/COBOL interface to the RPC+ DLL (RpcPlus.dll). Since tclient.exe is built using the same parameter-passing mechanisms as an RM/COBOL program, this DLL must be used to interface to the RPC+ routines. This DLL must be placed in the same (or working) directory as tclient.exe or in a directory specified in the PATH environment variable.
RpcPlus.dll	The RPC+ DLL. It handles communications with the server. This DLL must be placed in the same (or working) directory as tclient.exe or in a directory specified in the PATH environment variable.
RpcPlus.ini	A configuration file that tells RPC+ what server to connect to and what port to use. The contents of the file look like this:

```
[ClientConfig]
DefaultServer=xxx.xxx.xxx.xxx

[ServerConfig]
Port=portnumber
```

The DefaultServer entry specifies the IP address or the name of the Cobol-WOW Thin Client Server. The Port entry specifies the port number on the server associated with this service. These entries can be changed as needed for your installation.

The RpcPlus.ini file must be in the same (or working) directory as tclient.exe or in the Windows directory.

Files Installed on a Windows Server

The following list describes each file that must be installed on a Windows server in order to use the Thin Client portion of Cobol-WOW.

Files	Description
RpcPlusServer.exe	This program performs the important function of listening for a connection request, then starting the RM/COBOL runtime and application. This file can be installed in any location.
RpcPlusWOW.dll	The Cobol-WOW interface to RPC+ (RpcPlus.dll). The application programs make calls to functions such as WOWSETPROP and WOWGETPROP. The RpcPlusWOW.dll intercepts those calls and uses RPC+ to route them to the Windows client. This DLL may be placed in any location, because it must be specifically loaded in the runcobol command line with the L= option. A path can be included if the RpcPlusWOW.dll is located in a directory other than the working directory.
RpcPlus.dll	The RPC+ DLL. It handles communications with the Windows client. This DLL must be placed in the working directory for the application or in a directory specified in the PATH environment variable.
RpcPlus.ini	A configuration file with important information for RPC+. See the following example of the contents of this file.

Sample Contents of RcpPlus.ini for a Windows Server

```
[ClientConfig]
DefaultServer=CLIENT

[ServerConfig]
CobolType=rmcobol
StartupCommand=runcobol myapp.cob
L=rpcpluswow.dll
Port=5010
LogActivity=TRUE
LogFileName=rpcplus.log
WorkingDir=\myapp
```

In the configuration file installed on a Windows server, the only entry required in the [ClientConfig] section is DefaultServer. The DefaultServer entry <u>must</u> specify CLIENT.

In the Thin Client architecture, CALLs made by code executing on the server must be routed back to the CLIENT. This entry causes that to happen.

The first three entries in the [ServerConfig] section are required. The CobolType entry must specify rmcobol. This causes the RpcPlusServer program to use the correct command line format when starting the RM/COBOL runtime.

The StartupCommand entry can be edited to suit your installation. You may need to add a path to the runcobol command, but the runcobol command must be invoked here. You can specify whichever application program should be started for the application, presumably your main program. This program can call any number of subprograms in the normal COBOL manner. You can specify a path to this program. The RUNPATH environment variable or Windows Registry setting will be used to locate any called subprograms, but not the initial program. Finally, you may need to add a path to the L= option which loads RpcPlusWOW.dll, depending on where you installed that file.

The Port entry is required, and must specify the same port number that is contained in the RpcPlus.ini file on the client.

The last three entries are optional. The LogActivity=TRUE option tells RPC+ to record all connections in a log file. The LogFileName entry specifies the name of the log file. A path may be added to this file name.

It is highly recommended that you specify a LogFileName. Any communication errors or other problems detected by RPC+ will be written to this file. If no file is specified, these errors will be displayed in a message box. This will require a user to dismiss the message box before the RM/COBOL runtime can terminate.

The WorkingDir entry specifies a directory that will be established as the working directory for the RM/COBOL runtime, and therefore your application. If this entry is not specified, the working directory will be the working directory associated with the execution of RpcPlusServer.exe

Obviously, the most important item to install on your Windows server is your application, which can be placed in any location. There is no need to install it in the same location as any of the Cobol-WOW Thin Client files, although you certainly can. If you want to load your application in a separate area, the WorkingDir entry is a handy way to "move" to your application's directory.

Files Installed on a UNIX Server

The following list describes each file that must be installed on a UNIX server in order to use the Thin Client portion of Cobol-WOW. The first two files in the table, **/etc/services** and **/etc/inetd.conf**, are UNIX system files that must be edited to enable the built-in service, inetd, to handle accepting the connection requests from **tclient.exe** and launching the COBOL application.

Files	Description
/etc/services	The file that contains a list of service names and TCP/IP configuration information. An entry must be added to the /etc/services file as follows: `rpcplus 5000/tcp` This entry defines `rpcplus` as a service using `tcp` protocol on port `5000`. Do not change the service name or protocol. You can, however, select a different port number. Just be certain that the port number you select is not used by any other service on the server and matches the port number used by the Cobol-WOW Thin Client.
/etc/inetd.conf	The file that contains a list of services for which inetd should handle connection requests. An entry must be added to the /etc/inetd.conf file as follows: `rpcplus stream tcp nowait root /bin/sh /bin/sh/rpcplus/rpcstart` where `rpcplus` is the name of the service inetd is supposed to listen for. This service must be described in /etc/services, as discussed above. You should not need to change this. `stream tcp nowait` describe the type of network communication needed. Do not change these options. `root` indicates the user for which the server process will be initiated. You may want to have your application executed under a different user name. Be certain that the user name used here has adequate permissions to find and execute the application. `/bin/sh` is the name of the program that inetd should initiate for the service. `/bin/sh` is specified because a shell script is used to start the application. This entry is repeated and should not be changed. `/rpcplus/rpcstart` is a shell script that starts the RM/COBOL runtime system. The rpcstart script can be edited to set additional environment variables required by the application, or to set a working directory.

Files	Description
libetsrpc.so	The Cobol-WOW Thin Client shared object module. This file should be placed in the rmcobolso subdirectory of the RM/COBOL runtime installation. This shared object is the Cobol-WOW interface to the RpcPlus.dll. The application programs make calls to functions such as WOWSETPROP and WOWGETPROP. The RpcPlusWOW.dll intercepts those calls and uses RPC+ to route them to the Windows client.
rpcstart	A file that is a shell script, which starts the application. It must contain at least the following:

```
    TERM=ansi
  export TERM
  runcobol myapp.cob K
```

	This script can be expanded to set environment variables or to establish the current working directory. It is advisable to add a full path to runcobol and also to myapp. Myapp is the first RM/COBOL program in your application. It can, however, have any name you wish. The K option is required on the runcobol command line to suppress the banner. Displaying the banner would result in corruption of the client/server communications stream.
RpcPlus.ini	A configuration file with important information for RPC+. See the following example of the contents of this file.

Sample Contents of RcpPlus.ini for a UNIX Server

```
[ClientConfig]
DefaultServer=CLIENT

[ServerConfig]
Port=5010
LogActivity=TRUE
LogFileName=rpcplus.log
```

In the configuration file installed on a UNIX server, the only entry required in the [ClientConfig] section is DefaultServer. The DefaultServer entry must specify CLIENT. In the Thin Client architecture, CALLs made by code executing on the server must be routed back to the CLIENT. This entry causes that to happen.

The Port entry is required, but it is not used during normal operation. It can be used in some debugging situations.

The last two entries are optional. The LogActivity=TRUE option tells RPC+ to record all connections in a log file. The LogFileName entry specifies the name of the log file. A path may be added to this file name. It is highly recommended that you specify a LogFileName. Any communication errors or other problems detected by RPC+ will be written to this file. If no file is specified, these errors will be displayed in a message box. This will require a user to dismiss the message box before the RM/COBOL runtime can terminate.

Obviously, the most important item to install on your UNIX server is your application, which can be placed in any location. There is no need to install the application in the same location as any of the Cobol-WOW Thin Client files, although you certainly can. If you want to load your application in a separate area, adding a **cd** command to the rpcstart script is a handy way to "move" to your application's directory.

Running the Application with Cobol-WOW Thin Client

The following list describes each of the actions necessary to run your application with Cobol-WOW Thin Client on the server and Windows client workstation.

On this type of hardware	Do this

UNIX server	After editing the configuration files for inetd, the inetd daemon must be refreshed. This can be accomplished on all servers by rebooting the machine, but also can be accomplished on most systems with the following command:
	
	`kill -HUP pid`
	
	Where `pid` is the process id of the inetd process. This command causes inetd to reread its configuration files.
	
	You can use the following command to determine whether inetd is listening for a connection on the rpcplus service port:
	
	`etstat -a | grep "rpcplus"`
	
	This command should show an rpcplus port in a LISTEN state. Once that is shown, you are ready to start the client with the following command:
	
	`netstat -a | grep "rpcplus"`
	
	This command should show an rpcplus port in a LISTEN state. Once that is shown, you are ready to start the client.

Windows server	After you have installed your files, you need to execute RPCPlusServer.exe on the server. This will cause the server to start listening for connections from the client workstation. When a connection is received, the server will automatically start the application.
Windows client workstation	Once a server is listening for connections, you can execute tclient.exe on the client workstation. The application's interface will appear on the client.

Index

H

I

M