

# FUJITSU Software

## PowerCOBOL V11.0

A decorative horizontal band with a red-to-dark-red gradient, featuring abstract, glowing white and red lines that swirl and intersect, creating a sense of motion and energy.

# Getting Started

Windows

B1WD-3366-01ENZ0(00)  
August 2015

# Preface

---

Fujitsu PowerCOBOL is a Graphical User Interface (GUI) design and implementation tool for COBOL programmers. It provides all the graphical controls commonly used in applications today and gives the COBOL programmer the ability to respond to events and manipulate the controls using COBOL.

In addition to PowerCOBOL's GUI capabilities all the power of Fujitsu's standard COBOL compiler are available to the PowerCOBOL user.

This Getting Started manual aims to take you through the most significant parts of the product's functionality as quickly as possible, thus making the most effective use of your time.

## Audience

---

COBOL programmers ready to start creating GUI applications.

Product reviewers who are assessing Power COBOL's suitability for an organization or who are writing an article on PowerCOBOL.

You do not need to know COBOL to work through this manual, but you will need to know COBOL to make serious use of PowerCOBOL.

## How This Manual is Organized

---

This manual contains the following information:

Chapter 1	Introduction
Chapter 2	A Quick Tour
Appendix A	Tips
Appendix B	Support

## Related Manuals

---

NetCOBOL Getting Started  
NetCOBOL User's Guide  
NetCOBOL Language Reference  
NetCOBOL Debugging Guide  
PowerCOBOL User's Guide  
PowerCOBOL Reference  
PowerBSORT User's Guide  
PowerFORM Getting Started

## Trademarks

---

Microsoft, Windows, Windows Server and ActiveX are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Fujitsu, NetCOBOL and Fujitsu PowerCOBOL are trademarks or registered trademarks of Fujitsu Limited.

## Product Names

---

The following product names are abbreviated in this manual as follows:

Product Name	Abbreviation
Microsoft(R) Windows Server(R) 2012 R2 Datacenter	Windows Server 2012 R2
Microsoft(R) Windows Server(R) 2012 R2 Standard	
Microsoft(R) Windows Server(R) 2012 R2 Essentials	

Product Name	Abbreviation
Microsoft(R) Windows Server(R) 2012 R2 Foundation	
Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Essentials Microsoft(R) Windows Server(R) 2012 Foundation	Windows Server 2012
Microsoft(R) Windows Server(R) 2008 R2 Foundation Microsoft(R) Windows Server(R) 2008 R2 Standard Microsoft(R) Windows Server(R) 2008 R2 Enterprise Microsoft(R) Windows Server(R) 2008 R2 Datacenter	Windows Server 2008 R2
Windows(R) 8.1 Windows(R) 8.1 Pro Windows(R) 8.1 Enterprise	Windows 8.1
Windows(R) 8 Windows(R) 8 Pro Windows(R) 8 Enterprise	Windows 8
Windows(R) 7 Home Premium Windows(R) 7 Professional Windows(R) 7 Enterprise Windows(R) 7 Ultimate	Windows 7
Windows Server 2012 R2 Windows Server 2012 Windows Server 2008 R2 Windows 8.1 Windows 8 Windows 7	Windows

## Conventions Used in This Manual

---

Example of convention	Description
<b>setup, setup</b>	Characters you enter appear in bold.
<u>Program-name</u>	Underlined text indicates a placeholder for information you supply.
ENTER	Small capital letters are used for the name of keys and key sequences such as ENTER and CTRL+R. A plus sign (+) indicates a combination of keys.
...	Ellipses indicate the item immediately preceding can be specified repeatedly.
Edit, Literal	Names of menus and options appear with the initial letter capitalized.
[def]	Indicates that the enclosed item may be omitted.

Example of convention	Description
{ABC DEF}	Indicates that one of the enclosed items delimited by   is to be selected.
CHECK WITH PASCAL LINKAGE ALL PARAGRAPH-ID COBOL <u>ALL</u>	Commands, statements, clauses, and options you enter or select appear in uppercase. Program section names, and some proper names also appear in uppercase. Underlined text indicates the default.
DATA DIVISION. WORKING-STORAGE SECTION. * Get the #INCLUDE file to the data control #INCLUDE "NUMDATA.COB". PROCEDURE DIVISION.	This font is used for examples of program code.
The <i>form</i> acts as an application creation window.	Italics are occasionally used for emphasis.
"PowerCOBOL User's Guide" See Chapter 6, "Creating an Executable Program."	References to other publications or chapters within publications are in quotation marks.

## Export Regulation

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

August 2015

Copyright 1996-2015 FUJITSU LIMITED

# Contents

---

Chapter 1 Introduction.....	1
1.1 Overview of PowerCOBOL.....	1
1.2 Purpose of the PowerCOBOL Product.....	2
1.3 Key Features of PowerCOBOL.....	2
Chapter 2 A Quick Tour.....	4
2.1 Overview of the Tour through the Product.....	4
2.2 Tour through the Product.....	4
2.2.1 Introduction to PowerCOBOL using a Sample Project.....	4
2.2.2 Create Your Own Interface.....	14
2.2.3 Exploring the On-Line Reference.....	24
2.2.4 Using ActiveX Controls.....	24
2.2.4.1 Using ActiveX Controls.....	25
2.2.5 Constructing Sample Database.....	25
2.2.5.1 Creating Sample Database.....	25
2.2.5.2 Defining ODBC Source.....	26
Appendix A Tips.....	27
Index.....	28

# Chapter 1 Introduction

The introduction to PowerCOBOL explains the key features and benefits of the product and lists the major enhancements provided in the V4, V5, and V6 releases.

## 1.1 Overview of PowerCOBOL

PowerCOBOL is a development environment that has two major components:

- A project manager
- A form design tool

The project manager helps you keep track of all the pieces in your PowerCOBOL project by displaying a project tree alongside a list of the properties of the currently selected node of the tree (see Figure 1.1). It also provides all the functions you need to build, debug, and package your PowerCOBOL applications.

The form design tool (the Form Editor) provides all the functions required to create professional GUI windows and dialogs (see Figure 1.2). It contains a toolbox of controls, to which other controls can be added, and has all the functions required for matching object sizes and aligning controls.

Both the project manager and form design tool give access to the properties of all the controls and objects, which configure appearances and behaviors, and the event code, which is written in COBOL, that implements the application functions of the forms.

The purpose and features contained in these PowerCOBOL components are explained in the following sections.

Figure 1.1 The PowerCOBOL Project Manager window.

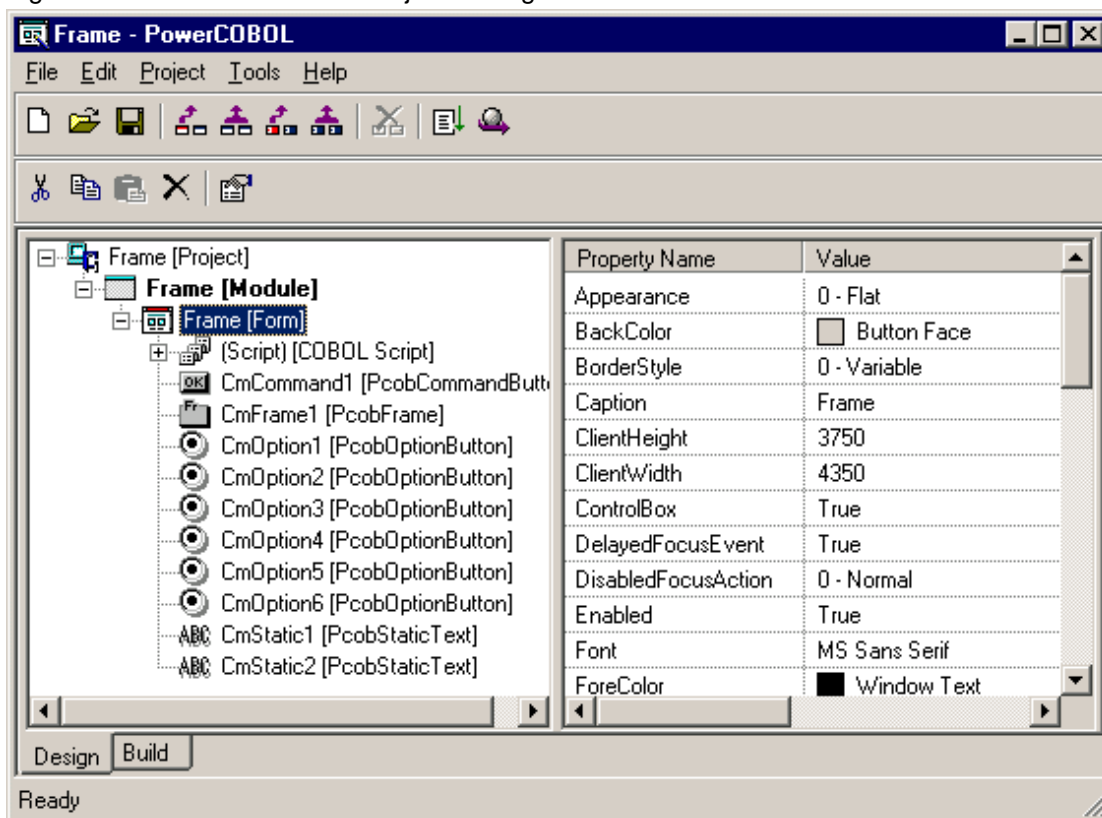
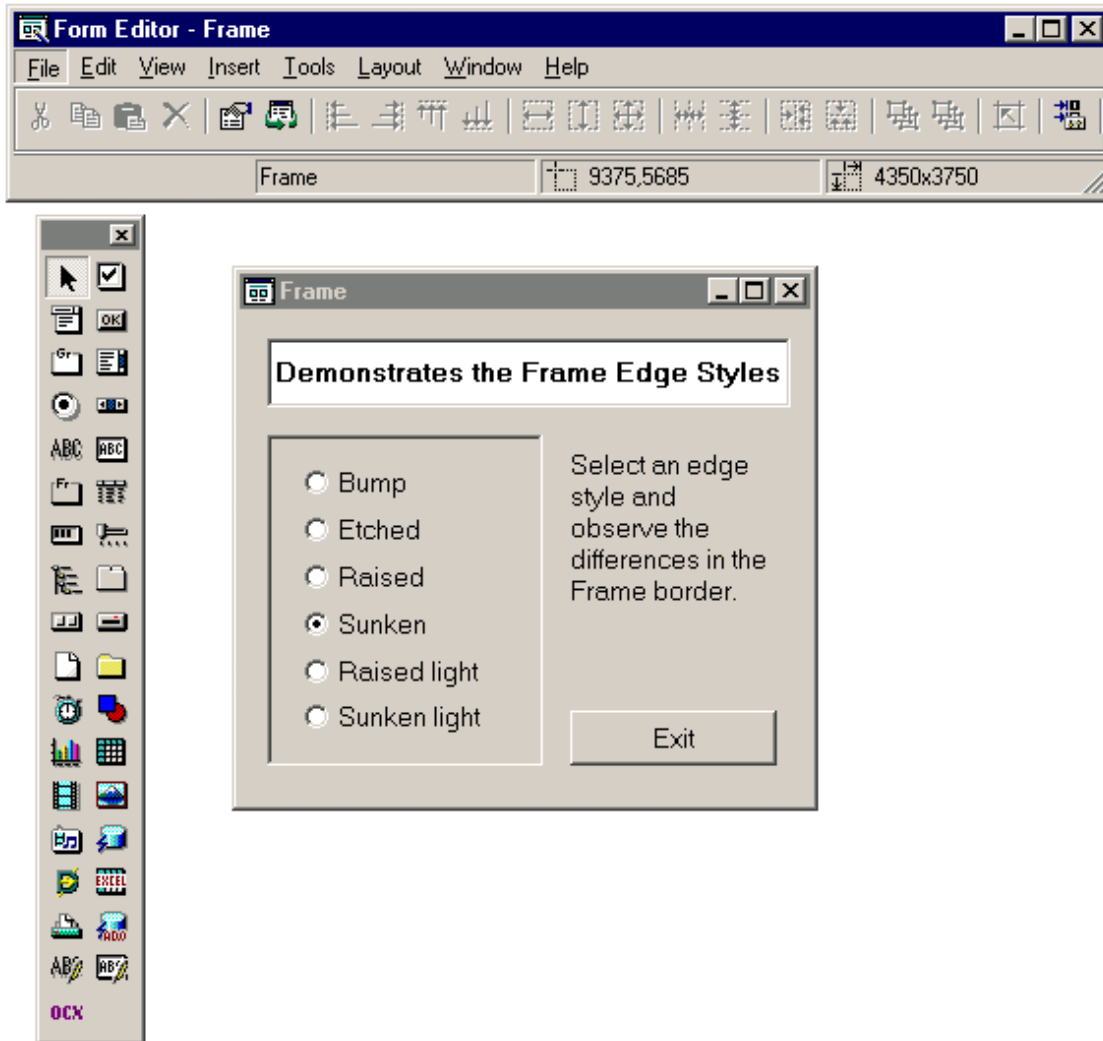


Figure 1.2 PowerCOBOL form design tool - the Form Editor.



## 1.2 Purpose of the PowerCOBOL Product

---

PowerCOBOL is designed to:

- Bring Windows **GUI programming** to COBOL programmers and applications.
- Provide **rapid prototyping and development** of the application interfaces.
- Provide straightforward **access to enterprise data**.
- Support **client/server** models.
- Give access to the latest controls by providing **ActiveX** support.
- Provide easy **integration with other application code** whether written in COBOL or other PC languages.

## 1.3 Key Features of PowerCOBOL

---

The key features of PowerCOBOL are:

- **Abstracts** the low-level Windows API to a higher level.
- Uses **COBOL** as the language **for** responding to **events** and invoking other **methods**.
- Follows the **OO COBOL** syntax **model** for interacting with GUI objects and their properties.

- Provides **COBOL edited data definitions** using standard PICTURE clause syntax.
- Supports **calls** to **standard COBOL** programs.
- Gives **guided development** - pop-up menus let you know exactly what events, methods and properties apply to the selected object. You do not have to tackle hundreds of pages of documentation to retrieve this information.
- Provides a **WYSIWYG** development environment in which you drag and drop objects onto forms and see the objects as they appear at execution time.
- Includes **all basic GUI controls**.
- Includes controls for **multimedia** features, **DDE**, and working with Microsoft **Excel**.
- Supports the import of **ActiveX** controls, providing **COBOL interfaces** to these controls.
- Is **COM** (Component Object Model) **compliant** - so you can mix and match PowerCOBOL code with other COM compliant code.
- Includes an **ADODataSource** control to permit access to remote database servers using **ADO** (Microsoft® ActiveX® Data Object) functions and controls.
- Supports for **third party ADO controls**.

The benefits of the PowerCOBOL approach will be quickly apparent to experienced COBOL programmers:

- You **don't** have to **learn** a new **scripting language** to work in a GUI environment.
- You **don't** have to **search** through **hundreds of pages** of low-level documentation to figure out what events might happen, which calls you can make or what properties apply to a given control.
- You **don't** have to **compile and execute** your code **to see** if your GUI looks just as you intended it to look.
- You can immediately **put** your **COBOL knowledge to work**.
- You can **integrate** your **existing application code** with your new GUI code.
- You can continue to write your code in the language designed for readability and maintainability.



# Chapter 2 A Quick Tour

Chapter 2 takes you on a quick tour of PowerCOBOL. After taking the tour you will have a good understanding of the capabilities of PowerCOBOL and, for those already familiar with COBOL, be ready to start creating your first PowerCOBOL applications.

## 2.1 Overview of the Tour through the Product

---

The guided tours in the following sections show you the following functions:

Introduction to PowerCOBOL using a Sample Project:

- Opening a project.
- Understanding the project tree view and properties panes.
- Editing a form.
- Seeing how code is associated with controls.
- Two ways of editing properties.
- Building and executing a project.
- Sizing, alignment and spacing aids.

Create your own interface

- Drag and drop design.
- Creating code to handle an event.
- Generating code for invoking methods.
- Generating code for referencing properties.
- Previewing interface behavior without building and executing.
- Building and executing an interface.
- Debugging an interface.

Exploring the On-Line Reference

- Quick route to details of controls, methods, events and properties.

Using ActiveX

- Adding ActiveX controls.
- Using ActiveX controls.

## 2.2 Tour through the Product

---


Follow the steps outlined in the sections below to acquaint yourself with the key features of the PowerCOBOL product.

### 2.2.1 Introduction to PowerCOBOL using a Sample Project

---

We will open the MouseEvent sample project and use it to explore some of PowerCOBOL's features.

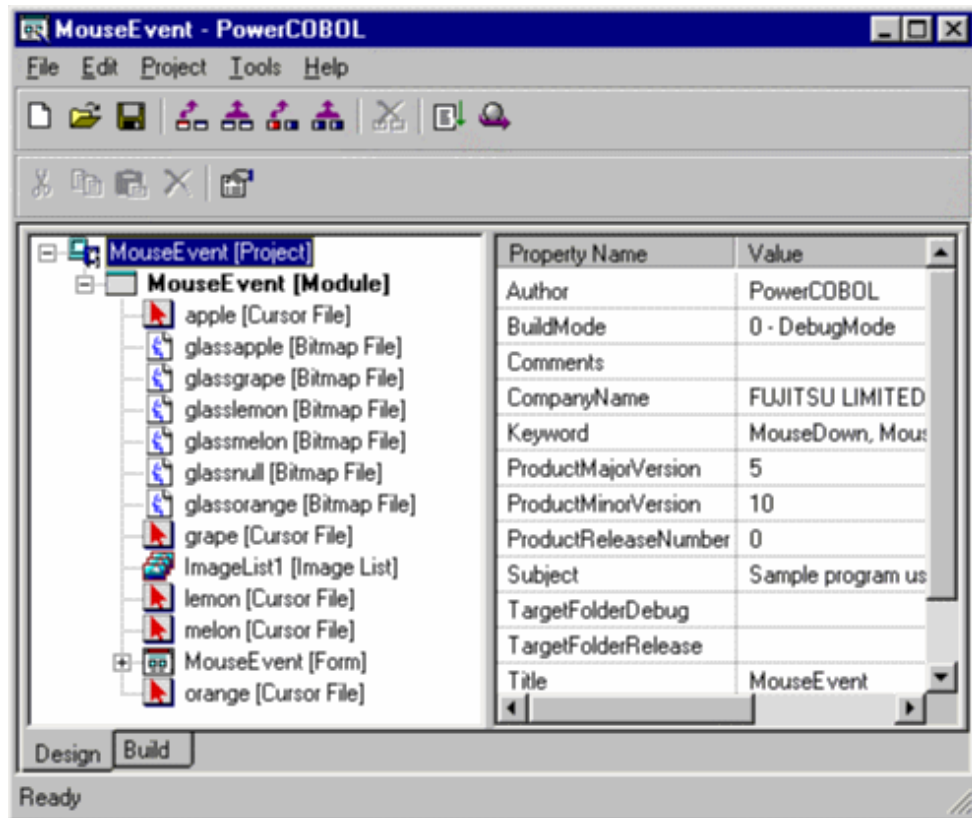
#### A. Invoke PowerCOBOL

1. From the Windows Start menu select Programs, NetCOBOL for Windows, PowerCOBOL.
2. We suggest you maximize the PowerCOBOL window so that you can focus totally on the product. To do this click on the maximize button, , in the top right corner. (The snapshots in this guide, however, will not use maximized windows to save space and give you clearer pictures.)

## B. Open the MouseEvent Sample Project

1. From the File menu, select Open.
2. Navigate to the:  
C:\Program Files\Fujitsu NetCOBOL for Windows\samples\PowerCOBOL\MouseEvent  
folder (assuming that you used the default folder names when you installed the products - substitute your folder names if you overrode the defaults).
3. Select the file "MouseEvent.ppj" and click Open. This file is the MouseEvent sample project - it lets you fill an imaginary glass with different fruit drinks.
4. Your window should be a bigger version of Figure 2.1:

Figure 2.1 PowerCOBOL Project Manager



This is PowerCOBOL's project manager. The parts of the window are:

**Menu bar and tool bars** - giving access to the project manager's functions.

**Design tab** - giving design related information. It is split into two panes. The left pane contains a project tree. The right pane displays the properties for the object selected in the project tree. You can edit properties in this pane by clicking on the property value.

**Build tab** - displaying the results of the last build and giving access to more build-related functions.

5. Expand the project tree by right-clicking on "MouseEvent [Project]" and selecting "Expand all" from the pop-up menu. This expands all the nodes in the project so you can see everything that makes up this application.
  - Near the top of the tree you can see the MouseEvent module. This is the executable file that is created when the project is built.
  - Under the MouseEvent module, after a number of bitmaps and cursors, is the MouseEvent form. This is the single window for the MouseEvent application and contains the key parts.
  - The module also includes a number of bitmap and cursor images which are shown as child nodes of the module.
  - Child nodes are in alphabetical order under the parent node.
  - Scroll up and down the tree to observe the different parts of the project: the COBOL "scriptlets" (the small pieces of COBOL code that control the application), command button, group box, static text, image, and list view.

- Notice how the different icons help you recognize the types of control.
6. Select different items in the project tree and observe how the properties change in the right-hand pane.
  7. Try editing a property in the right-hand pane. For example, select MouseEvent [Project] in the project tree and click on the Comments value in the properties list. Enter the description: "Sample program demonstrating the use of mouse events".

Figure 2.2 Editing a Property in Project Manager

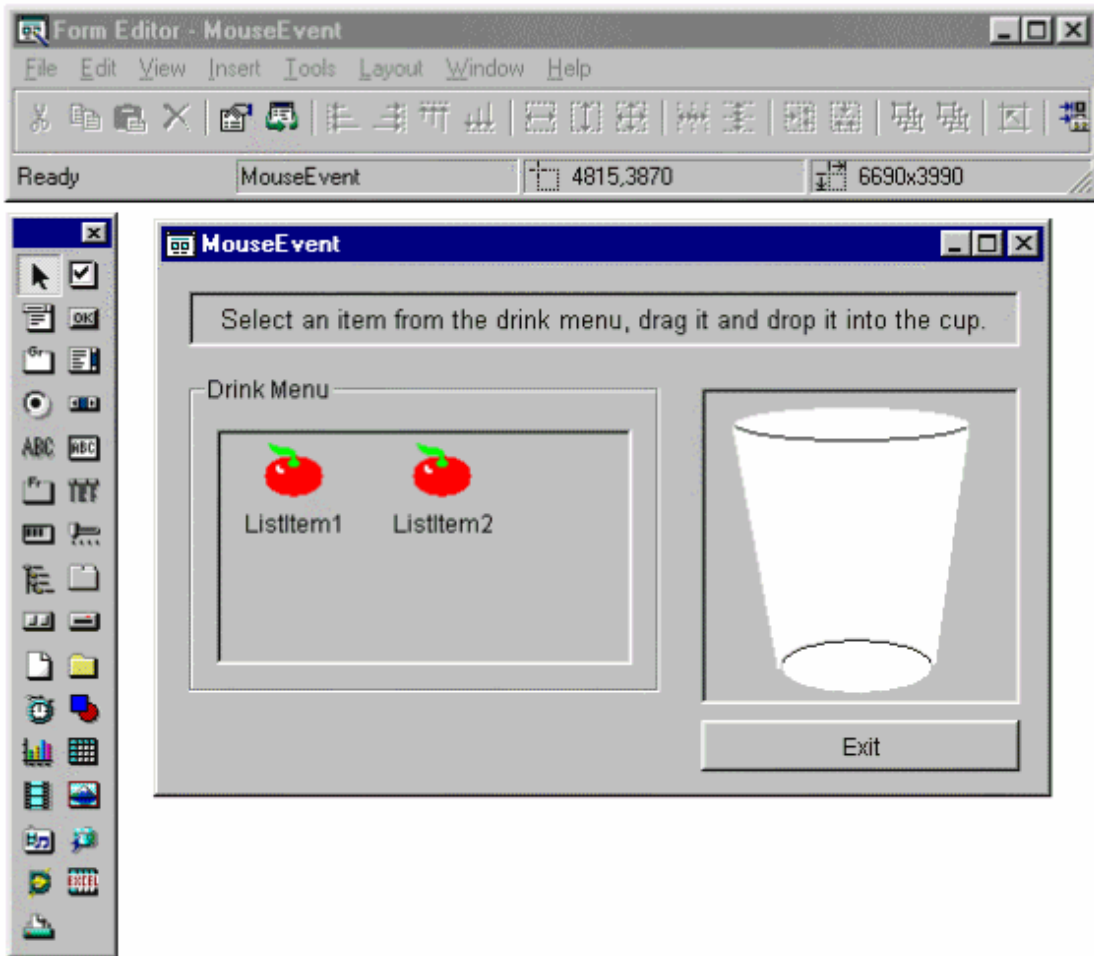
Property Name	Value
Author	PowerCOBOL
BuildMode	0 - DebugMode
Comments	am demonstrating the use of mouse events
CompanyName	FUJITSU LIMITED
Keyword	MouseDown, MouseMove, MouseUp
ProductMajorVersion	5
ProductMinorVersion	10
ProductReleaseNumber	0
Subject	Sample program using mouse event
TargetFolderDebug	
TargetFolderRelease	
Title	MouseEvent
UseDiagnosis	True


### C. Explore the Form

The window or dialog box design is called a "form" in PowerCOBOL. The following steps introduce you to the form editor.

1. In the project tree right-click on "mouseevent [Form]" and select Open from the pop-up menu. This brings up the form editor, displaying the window of the MouseEvent project and a toolbox containing the controls.

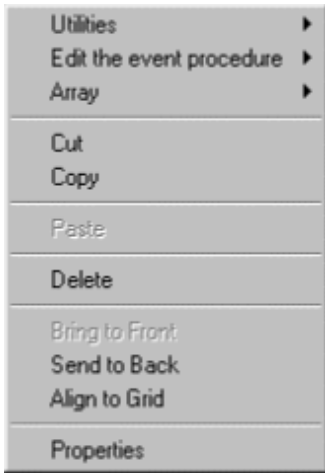
Figure 2.3 The MouseEvent window displayed in the Form Editor



2. The MouseEvent window looks almost like it would if you executed the application except that it doesn't show all the different list item icons and it may have a placeholder, , for the image of the glass (PowerCOBOL uses this placeholder if memory is tight). First select the different objects ("controls") in the window and observe how the status bar, just below the toolbar, displays the name of the control, its position and size. (You'll do more with sizes, positions and editing controls in a few minutes.)

- Now right-click on a control, for example the Listview control (the one displaying two tomatoes). The form editor displays a pop-up menu containing the most frequently used functions:

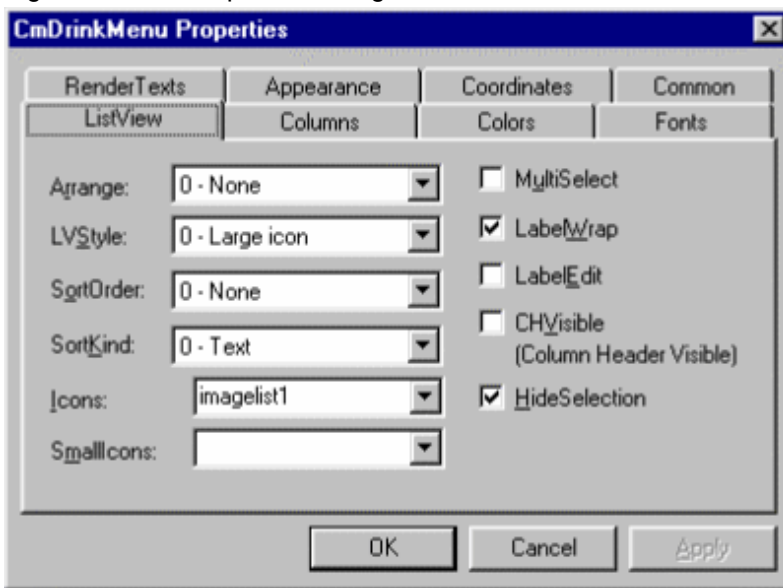
Figure 2.4 A pop-up menu for a control in the Form Editor.



**NOTE:** It is always worth checking the pop-up menus in PowerCOBOL as they provide important functions to you quickly and directly.

- From the pop-up menu select Properties. The form editor displays a dialog box containing all the properties for the selected control.

Figure 2.5 The Properties dialog for a ListView control.



These are the same properties you saw listed in the Properties pane of the project manager, except that the properties are displayed in logical groups, not alphabetically as in the project manager. Each logical group of properties is displayed on a separate tab.

For the ListView control the groups are:

**ListView** - Properties specific to the ListView control, such as the specific style of the ListView control and the file that contains the icons to be used.

**Columns** - When the ListView control has the Report style - in which several columns of data can be displayed - the columns group of properties defines the details of the columns such as width and heading text. (The MouseEvent project doesn't use the Report style, but it could be selected dynamically so all the Columns fields are enabled).

**Colors** - The colors to use in the control. Note that you can select from different sets of colors.

**Fonts** - The font properties to be used for text displayed in the control.

**RenderTexts** - For controls in which textual data is displayed, such as in the Report style of the ListView control, you edit the output text using pre-configured COBOL or Date picture strings. The RenderTexts properties determine the strings to use.

**Appearance** - The background, border and 3D/flat styles.

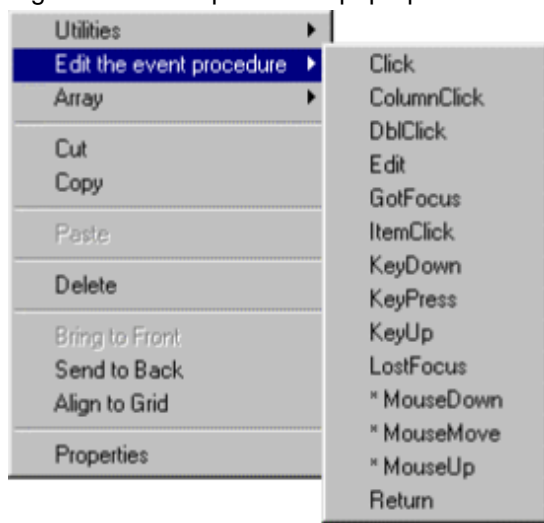
**Coordinates** - The units of measurement to be used for the control along with its position and size expressed in those units. Also contains the ScalingStyle - a property that enables you to configure behavior if the window is resized.

**Common** - Properties that are common to most controls, such as the name to be used for referring to the control from the source code.

Notice how the grouping of the properties makes it easy to understand what can be configured for the control and makes it easy to find a particular property.

5. Cancel the Properties dialog (you can play with property values later, or when you create your own windows).
6. Right-click on the ListView control again, but this time select the "Edit the event procedure" function. This displays a sub-menu containing a list of all the events that can be generated for this control.

Figure 2.6 Event procedure pop-up sub-menu.

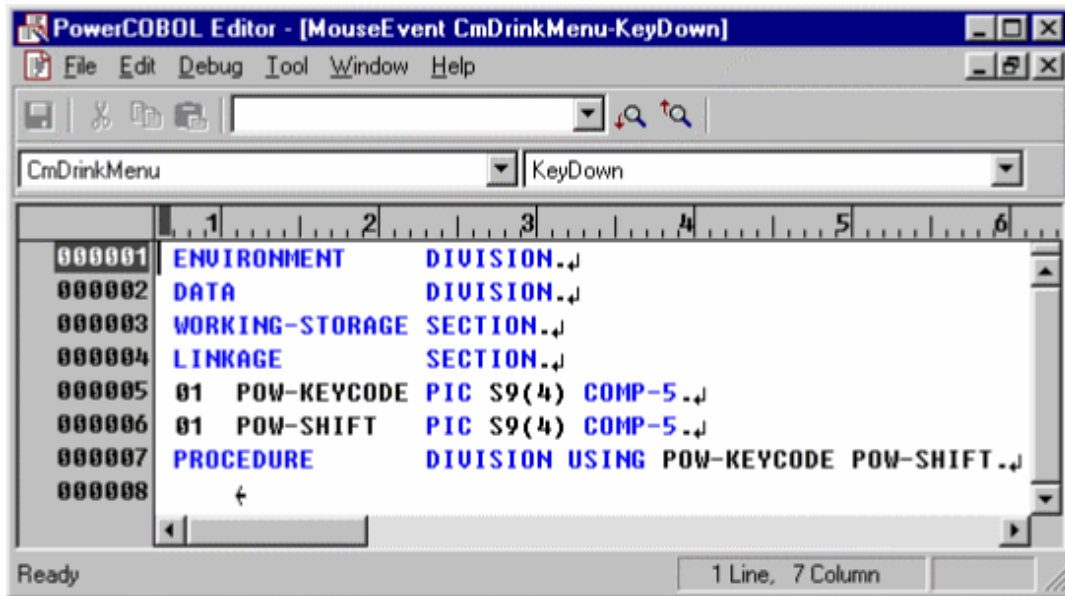


This is a key feature of PowerCOBOL - there is no guesswork involved. It is clear at each stage what is available. This greatly speeds learning and using the product. Also the system takes care of events that do not have application code attached - so applications can be developed a small step at a time: beginning programmers can learn how to handle one event and not worry about the other events they don't yet understand.

Notice how three events in the above pop-up have "\*"s by them (MouseDown, MouseMove, MouseUp). These are the three events that contain application code - all the others are handled by the system, usually taking no action.

- Before looking at the events with application code, we will look at what is provided for a new event. From the events sub-menu select KeyDown. PowerCOBOL displays a text editor window with some skeleton code already inserted.

Figure 2.7 PowerCOBOL Editor showing skeleton event code

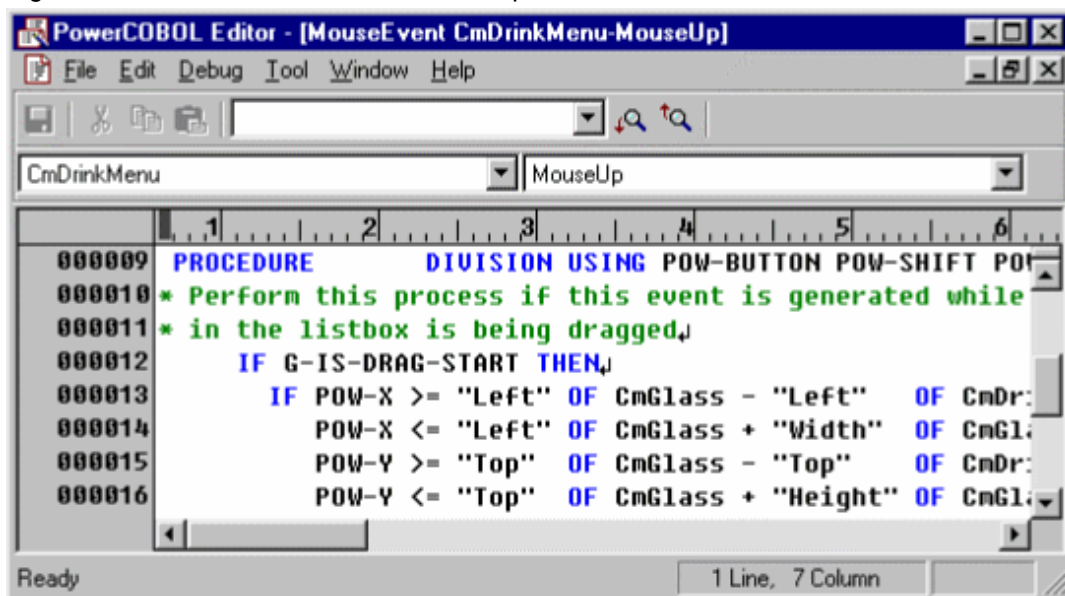


First note that this code is immediately familiar to COBOL programmers - they will have little doubt as to what they need to do next (define any local data items they require and enter the procedure code to handle the event).

Secondly, note that the KeyDown event communicates two values to the procedure code, namely a code indicating the key that was pressed and the status of the shift keys. The skeleton code already has these set up in the LINKAGE SECTION and the USING phrase of the PROCEDURE DIVISION header. Again the guesswork has been removed - the programmer immediately sees the parameters that are available to him and the form in which it is provided (PIC S9(4) COMP-5).

- Now look at some of the code that has already been created. Close the KeyDown event text editor, right-click on the ListView control, select "Edit the event procedure" and this time select the MouseUp event. This time PowerCOBOL displays the existing code within the text editor.

Figure 2.8 Procedure code for the MouseUp event.



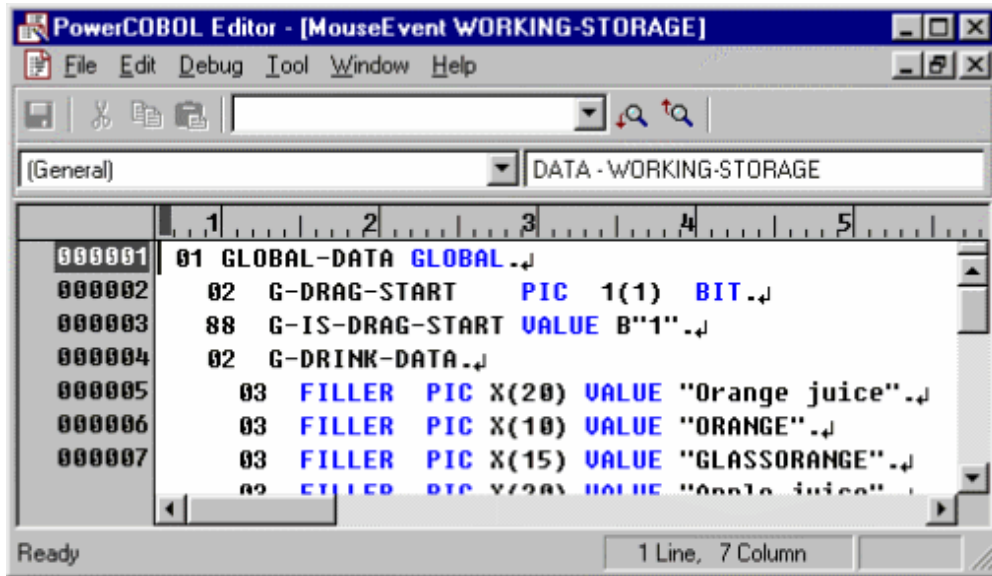
- If necessary enlarge the window by dragging the bottom right corner so that you can see all or most of the code. Because COBOL is designed to be readable, you should quickly get a feel for what is happening in the code. It checks that a drag was started, changes the glass image if the mouse pointer is over the glass, restores the pointer to its original state and resets two

status indicators.

You may wonder where the data items are defined so...

10. Return to the form editor (select "Form Editor - MouseEvent" from the Windows task bar), click on a blank area of the form so that no control is selected and, by default, the MouseEvent form is selected.
11. Right-click on the same blank area, and select "Edit DATA DIVISION" from the pop-up and "\* WORKING-STORAGE" from the sub-menu.

Figure 2.9 WORKING-STORAGE code for MouseEvent



The screenshot shows the PowerCOBOL Editor window titled "PowerCOBOL Editor - [MouseEvent WORKING-STORAGE]". The window has a menu bar with "File", "Edit", "Debug", "Tool", "Window", and "Help". Below the menu bar is a toolbar with icons for file operations and search. The main editing area shows the following COBOL code:

```
000001 01 GLOBAL-DATA GLOBAL .↓  
000002 02 G-DRAG-START PIC 1(1) BIT .↓  
000003 88 G-IS-DRAG-START VALUE B"1" .↓  
000004 02 G-DRINK-DATA .↓  
000005 03 FILLER PIC X(20) VALUE "Orange juice" .↓  
000006 03 FILLER PIC X(10) VALUE "ORANGE" .↓  
000007 03 FILLER PIC X(15) VALUE "GLASSORANGE" .↓  
000008 03 FILLER PIC X(20) VALUE "Apple juice" .↓
```

The status bar at the bottom of the window displays "Ready" and "1 Line, 7 Column".

PowerCOBOL displays the WORKING-STORAGE data defined for the form. Any items declared as GLOBAL can be referenced and set by any of the procedure code "scriptlets" defined for the controls in the form.

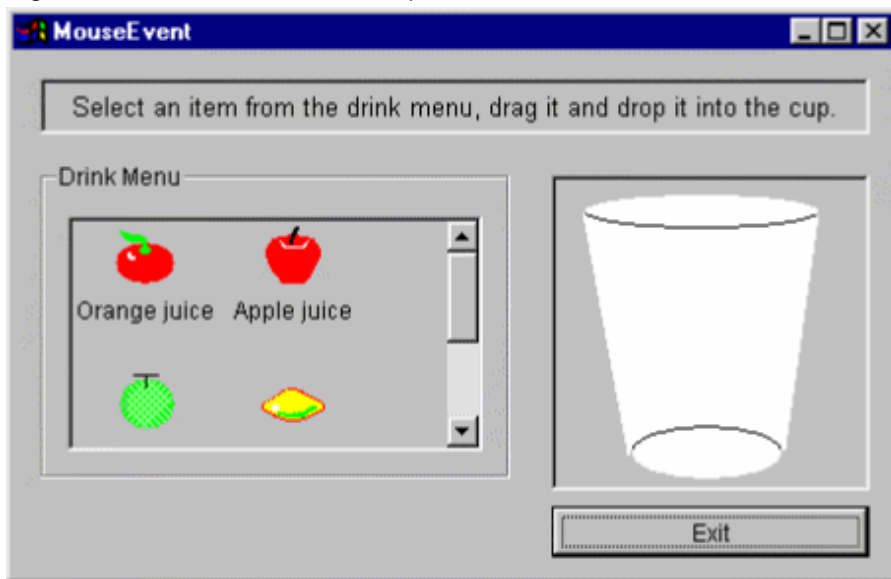
This illustrates the flexibility PowerCOBOL provides in data definition - you can define data that is local to a control or procedure, and data that can be referenced by all procedures within a form.

12. Having seen the design and some of the code for the form, you probably want to see exactly what it does. Return to the project manager by selecting "MouseEvent - PowerCOBOL" from the Windows task bar. Right-click on "MouseEvent [Module]" and select Execute from the pop-up menu.



13. PowerCOBOL will offer to build and save the project for you - respond OK to these messages. You will see PowerCOBOL building the application and switching back to the project tree when the build is successful. PowerCOBOL then starts the MouseEvent module.

Figure 2.10 The MouseEvent sample window.



14. Experiment with the interface, possibly switching back to the form editor to inspect the properties and code that determine the behaviors.

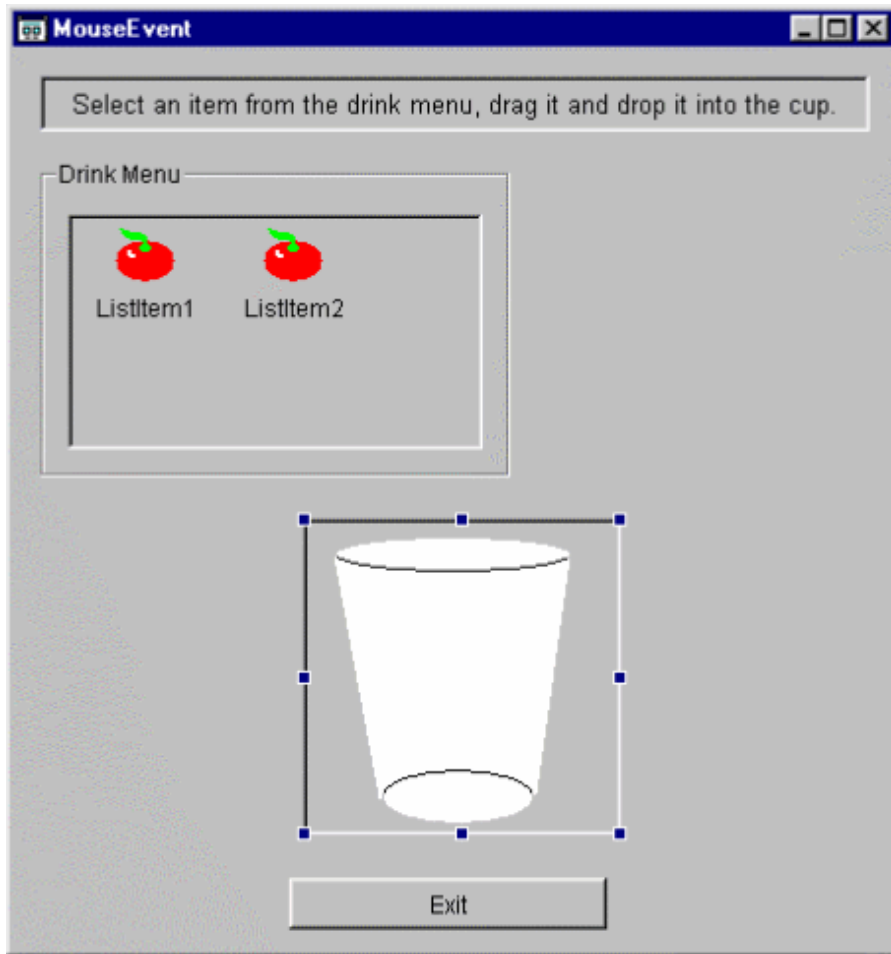
Finally in this quick tour of an existing project, we will look at the sizing, alignment and spacing aids in PowerCOBOL.

If you've ever tried creating a tidy dialog layout, you will know that sizing and aligning items can be one of the more tedious operations, unless your design tool provides you with appropriate functions.

15. Return to the form editor by selecting "Form Editor - MouseEvent" from the Windows task bar.
16. Suppose that you decided to align everything vertically in the window - so that all items stretch the full width of the window, with the Drink Menu group box under the "Select an item ..." static text control, the image of the glass under the group box, and the Exit button under the image. First stretch the window, by clicking on and dragging the bottom border, so that it is long enough for the controls to be stacked vertically.
17. Click on the Exit button and drag it to the bottom of the window.


- Click on the image control and drag it down so that it is between the Drink Menu group box and the Exit button. Your window might look like the one in Figure 2.11 below.

Figure 2.11 Repositioned items for aligning and sizing exercise.



- We are going to perform operations on a group of controls, so we first select that group. The top static text control ("Select an item ...") is going to be the reference point so we select that first by clicking on it. Then press and hold the shift key down and click on the "Drink Menu" group box, image control and "Exit" command button. All the controls, apart from the ListView control, should be selected - indicated by squares ("grapples") at the corners and midpoints of the controls' sides.


- First we will make sure all the items are aligned horizontally.

On the form editor toolbar, click on the Adjust left button: 

All the controls are left aligned.

Notice that, in the same group on the toolbar, there are Adjust right, Adjust top, and Adjust bottom buttons as well.

- Next we make sure all the controls are the same width.

On the form editor toolbar, click on the Width Match button: 

All the controls are resized so that their width matches the "Demonstrates how to ..." static text control (the reference control).

Notice that, in the same group on the toolbar, there are also Height Match, and Size Match buttons.

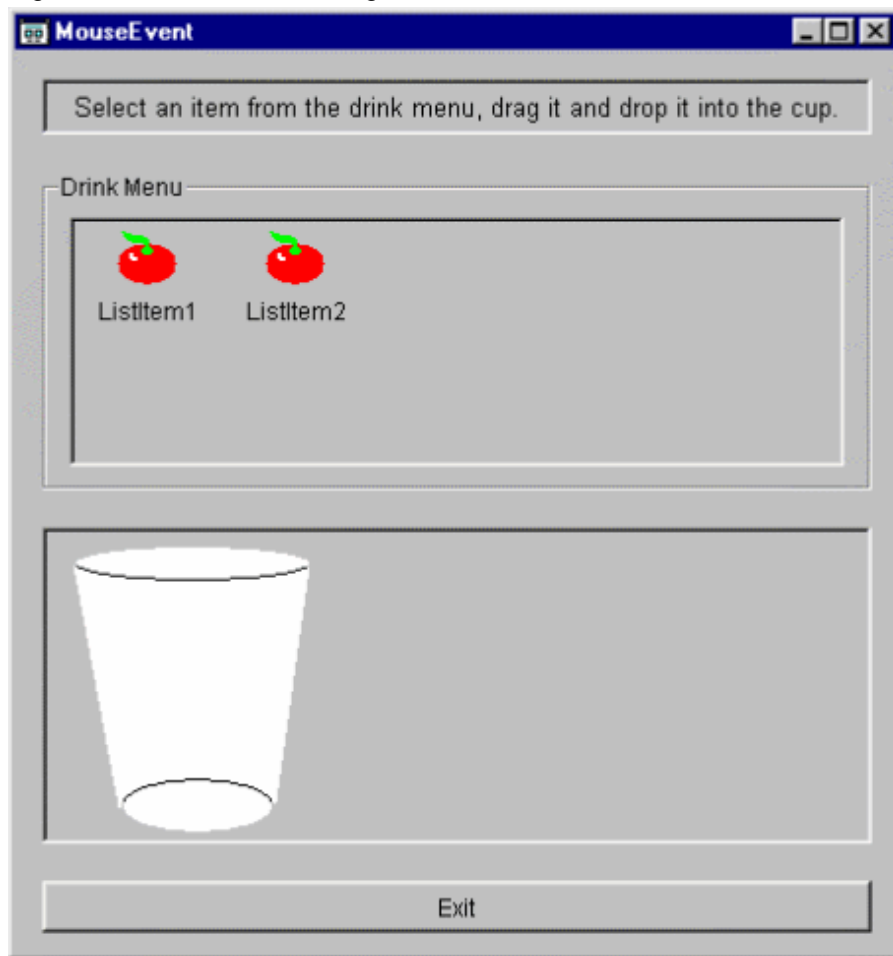
- The last group operation is to ensure that the controls are spaced evenly vertically.

On the form editor toolbar, click on the Vertical Space Equalization button: 

The vertical positions of the controls are adjusted to space them evenly.

23. Finally you reposition and resize the ListView control to match the size of the Drink Menu group box by selecting it and dragging its edges or corners.  
Your final layout should look something like the form in Figure 2.12:

Figure 2.12 Resized, and realigned MouseEvent window.



Although it is unlikely that you would choose this design you can see how most of your positioning requirements have been met with a minimum of effort.

24. There is one other positioning aid that you might expect - an alignment grid. You can:
- Set the size of the grid.
  - Have it displayed by dots on the form or kept invisible.
  - Have objects positioned on the grid.
  - Have objects sized to fit the grid.

Set the grid properties by selecting Tools, Grid from the Form Editor menu.

## 2.2.2 Create Your Own Interface

---

Having explored the MouseEvent project we will now create and test a simple interface. This interface will have a progress indicator that you will control using a command button - we will call it "Manual Progress".

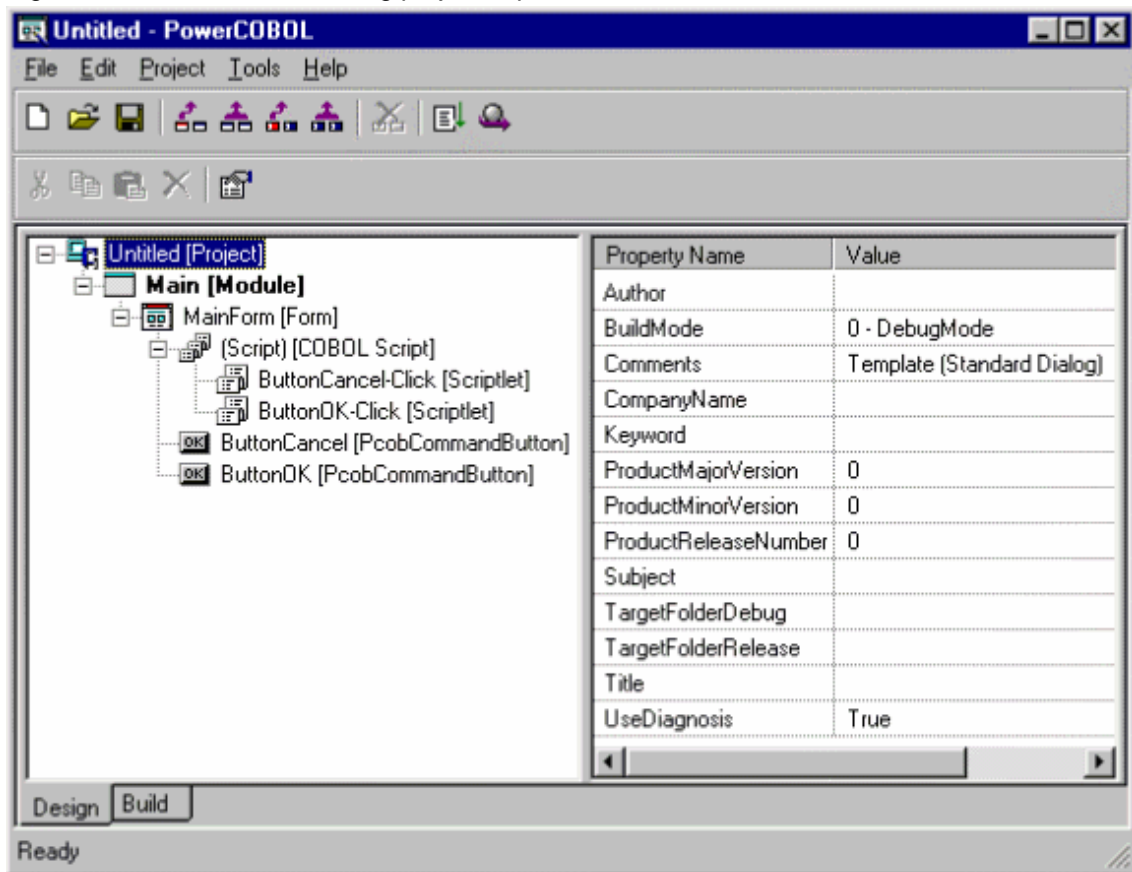
### A. Freshen PowerCOBOL

1. If you still have the MouseEvent project open, return to the PowerCOBOL project manager ("MouseEvent - PowerCOBOL" on the Windows task bar) and select File, Close from the menu bar.  
Suggest you respond "No" to the prompt asking if you want to save your changes.

## B. Start a New Project

1. From the PowerCOBOL project manager menu, select File, New Project.  
This displays the New Project dialog.
2. In the New Project dialog select StandardDialog and click OK.  
PowerCOBOL creates a project called "Untitled".
3. In the project tree right-click on "Untitled [Project]" and select "Expand all" from the pop-up menu.  
Notice that the StandardDialog project already has two CommandButtons and associated scriptlets.


Figure 2.13 The Standard Dialog project expanded.



4. Give a name to the project and save it, by selecting File, Save As from the project manager menu. Navigate to a suitable folder (for example create a folder under the "samples" folder called "Getting Started Example"), enter the name "Manual Progress", and click Save.
5. Open the Form Editor by right-clicking on "MainForm [Form]" and selecting Open from the pop-up menu.  
This brings up the Form Editor, ready for you to start adding controls.

## C. Add Controls to the Form

We will add two controls to the form: a ProgressIndicator control and a CommandButton control. The CommandButton will cause the ProgressIndicator to advance (illustrating coding and event and using methods) and will also change the background color of the ProgressIndicator (illustrating referencing and setting properties).

1. In the form editor toolbox (the long, narrow window with two columns of buttons) select the CommandButton control:  .
2. Move the mouse pointer over the form.  
The pointer changes to the rectangular outline of a CommandButton.
3. Move the pointer to a suitable position for a push button, for example to the left of the OK button, and click.  
The form editor creates a CommandButton called "CmCommand1" at the position you clicked.


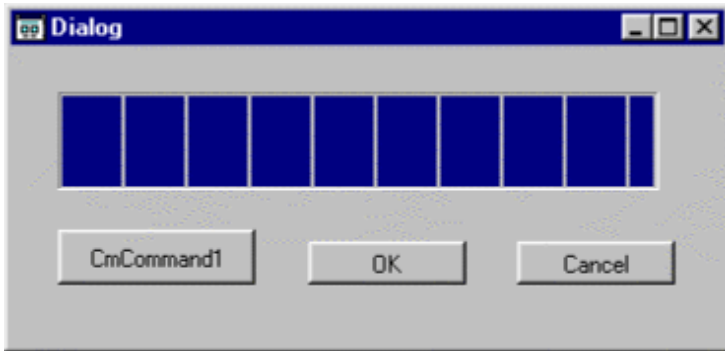
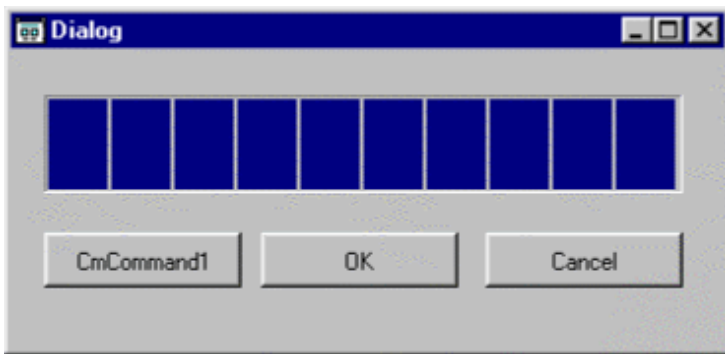
- Next place a ProgressIndicator control on the form by selecting its button, , in the form editor toolbox, moving the pointer to a suitable position and clicking.
- Stretch the ProgressIndicator control to a suitable size by clicking and dragging on the square grapples displayed at the corners and sides of the control when it is selected.  
Your window should look something like the one shown in Figure 2.14 (note this window has been resized to create a smaller snapshot for this guide):

Figure 2.14 The Manual Progress form - first draft!



- Now try using the sizing and alignment tools to make the buttons the same size, evenly spaced, and aligning with the Progress Indicator control, to produce an arrangement like that shown in Figure 2.15.

Figure 2.15 The Manual Progress form - tidied up.



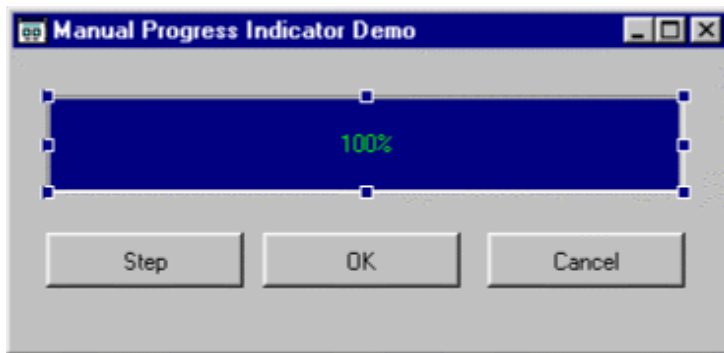
#### D. Edit the Form and Control Properties

Next we adjust the form and control properties.

- Double-click in an unused area of the form to bring up the properties dialog for the form.
- Change the text in the Caption entry field to read "Manual Progress Indicator Demo", and click OK.  
The title bar of the dialog box displays the text you entered.
- Double-click on the CmCommand1 CommandButton to display its properties.
- Change the text in the Caption entry field to read "Step", and click OK.  
The text on the CommandButton changes to "Step".
- Double-click on the ProgressIndicator control to display its properties.
- Check the DisplayRate check box - this will cause a % figure to be displayed in the middle of the progress indicator.
- Check the Smooth check box - this will make the progress indicator use a smooth, rather than a broken bar.
- Click on the Colors tab. From the Properties drop-down list select ForeColor.  
From the ColorSet drop-down list select Standard Colors.  
From the ColorList select Green.  
This makes the % figure stand out on the background and bar.

9. Click on the Common tab and double-click on the name CmProgress1. Right-click on the selected name and select Copy from the pop-up menu.  
We'll use this name when we create code in the next section.
10. Click OK to close the properties dialog.  
Your window should now look something like Figure 2.16:

Figure 2.16 Manual Progress form with text and color changes.



## E. Preview the Window

Now that the window layout is designed let us check generally how it looks and behaves.

1. From the Layout menu, select Preview.

PowerCOBOL displays the window as it would when executing the application. Even with a simple window like the one we are using you can check a few details:

- We can see how the ProgressIndicator control looks at the 0% stage. We could determine to use a color other than green for the text.
- Which button is the default button - the one with the dashed "button-selected" highlight? It is probably better to make Step the default so we will change that in a moment.
- Try pressing the Tab key a few times. You can see how the focus moves from control to control. We need to ensure the order is correct. For a more complex dialog this is an important check as you want the flow from control to control to work well for your users.
- Try clicking on the buttons - the buttons depress but no action is taken. This is because this is only a preview. To see the code executing we need to build and execute the application.

2. Close the preview window by clicking on the "X" button on the right of the window title bar.

Change the default button by:

3. Double-clicking on the "Step" CommandButton to display the Properties dialog.

Click on the Common tab.

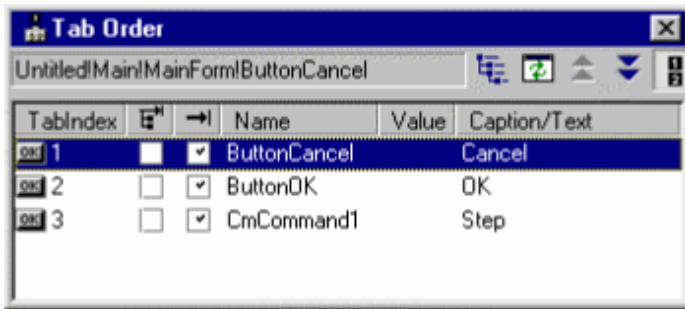
Select "Default Button" in the "Button Style" group box.



Click OK to close the properties dialog.

Change the tab order using the Tab Order dialog by:

- From the Form Editor, Layout menu, select the "Tab Order" function. The Form Editor displays the Tab Order dialog:

Figure 2.17 Tab Order dialog.



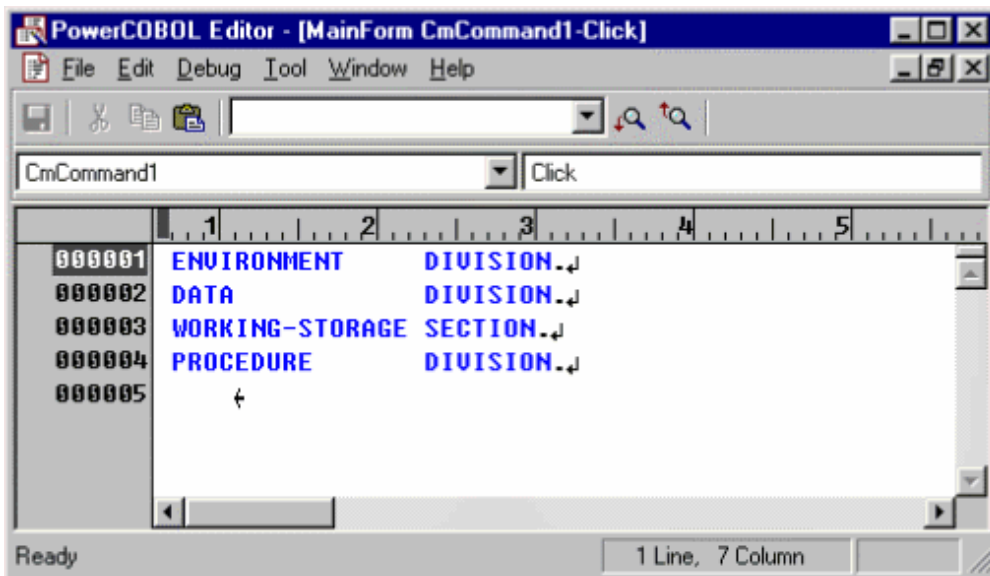
- Move the Step button to the first place in the tab order by selecting the line containing "Step" and using the Up button, , in the dialog's toolbar.
- Ensure OK can Cancel are in the correct order by using the Up or Down buttons.
- Close the Tab Order dialog by clicking on the Close button, , in the top-right corner.
- Select Layout, Preview to confirm that the default button and tab order are correct.
- Close the preview window by clicking on the "X" button on the right of the window title bar.

## F. Add Code to the Step CommandButton

Now we write the code to respond to the button being clicked.

- Right-click on the Step CommandButton control and select "Edit the event procedure" from the pop-up menu.
- Select Click from the sub-menu. PowerCOBOL brings up an edit window with division and section headers already configured.

Figure 2.18 Initial state of the code for the Click event of the Step button.



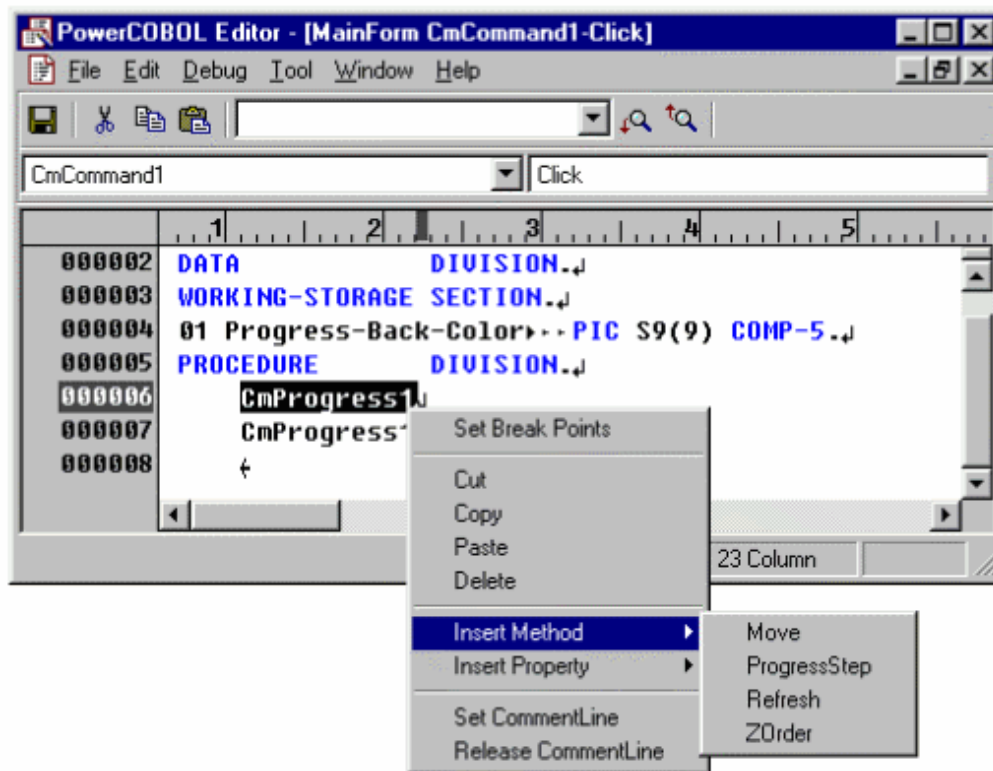
- In the WORKING-STORAGE SECTION insert a single item:

```
01 Progress-Back-Color PIC S9(9) COMP-5.
```

This will be used as a temporary store for the background color of the progress indicator.

4. In the procedure division add a couple of blank lines and paste the ProgressIndicator control name ("CmProgress1") into each of the lines (- this should still be in your clipboard). To paste you can either right-click on the line and select Paste from the pop-up menu, or select Edit, Paste from the edit window menu.  
PowerCOBOL can help generate code using the text strings you have pasted.
5. Double-click on the first "CmProgress1" string to select it.
6. Right-click on the selected string, and select Insert Method from the pop-up.  
PowerCOBOL displays a sub-menu as shown in Figure 2.19:

Figure 2.19 Editor Insert Method menu.



PowerCOBOL recognizes that the selected string is the name of a ProgressIndicator control and displays a list of the methods that are available for ProgressIndicators, namely: Move, ProgressStep, Refresh and ZOrder.

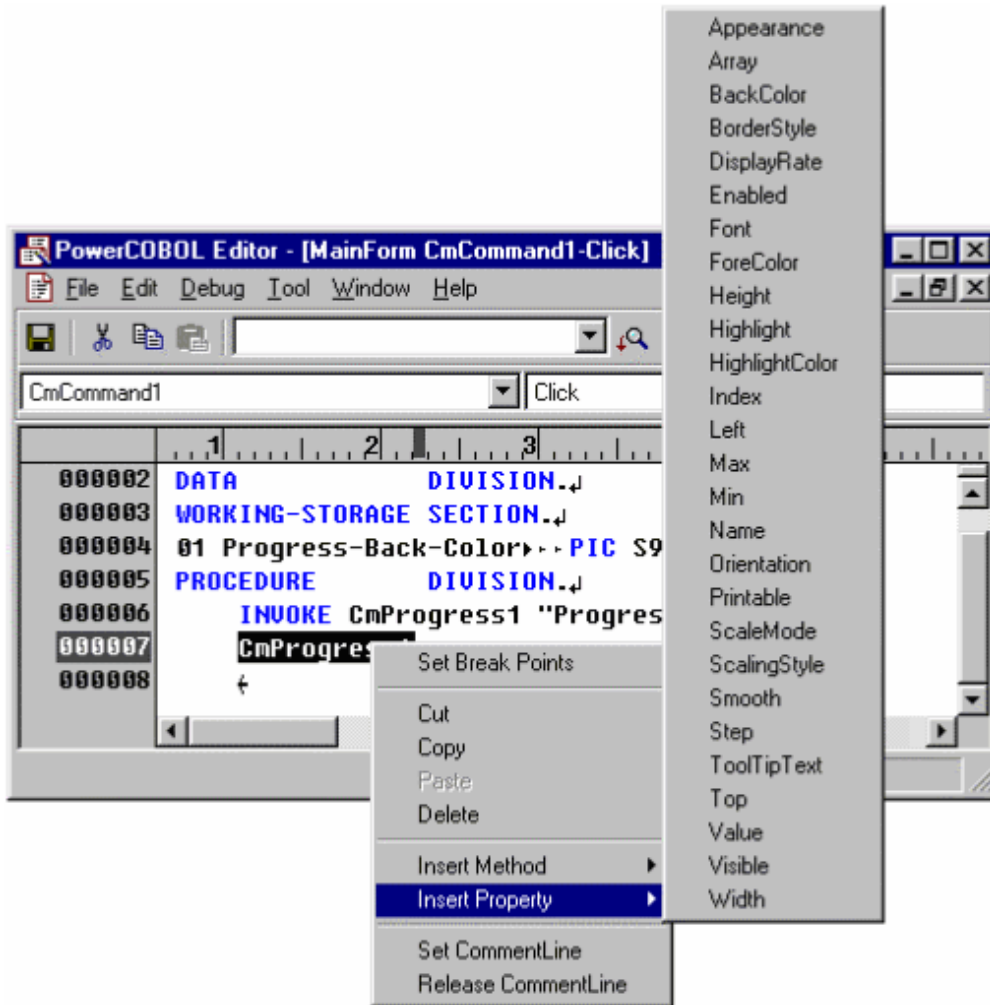
7. From the Insert Method sub-menu select ProgressStep.  
PowerCOBOL generates the INVOKE statement for the ProgressStep method. This method does not require any parameters, but if it did, PowerCOBOL would insert placeholder names to remind you which parameters you need to supply.  
The ProgressStep method advances the progress indicator one "step". You can configure the step size in the Properties dialog. By default the step is 10, in a range from 0 to 100.
8. We will now use the other "CmProgress1" string to help us reference and set the BackColor property.  
Double-click on the second "CmProgress1" string to select it. Right-click on the selected string and select Insert Property from the



pop-up menu.

PowerCOBOL displays a sub-menu containing all the ProgressIndicator control properties as shown in Figure 2.20:

Figure 2.20 Editor Insert Property menu.



9. Select BackColor from the Insert Property sub-menu.

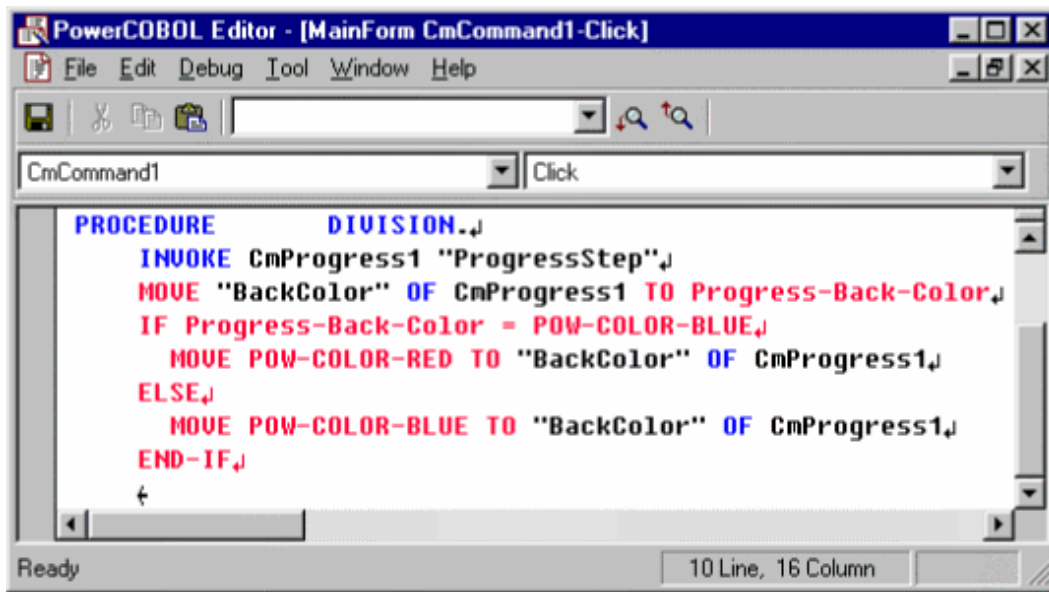
PowerCOBOL inserts the code for referencing the BackColor property.

Notice that, although we haven't generated a lot of code for the method or the property, we haven't had to do any guesswork. We found out the available methods and properties from pop-up menus, and PowerCOBOL inserted the correct code, ensuring that we don't make any typos.

10. Select the string: ' "BackColor" OF CmProgress1 '
11. Copy the string (by using Edit, Copy from the menu, or selecting Copy from the pop-up menu).
12. Create two new lines and paste the string into those lines (by using Edit, Paste, or selecting Paste from the pop-up menu).

13. Now insert the code shown in red (or light color if you are reading this from a printed manual) in Figure 2.21.

Figure 2.21 Inserting code in the Click event procedure.



This has the effect of flipping the background color of the ProgressIndicator control between blue and red every time the Step button is clicked.

14. Select File, Save from the menu to save the code.
15. Select File, "Close All" to close the edit window.

## G. Build and Execute the Application

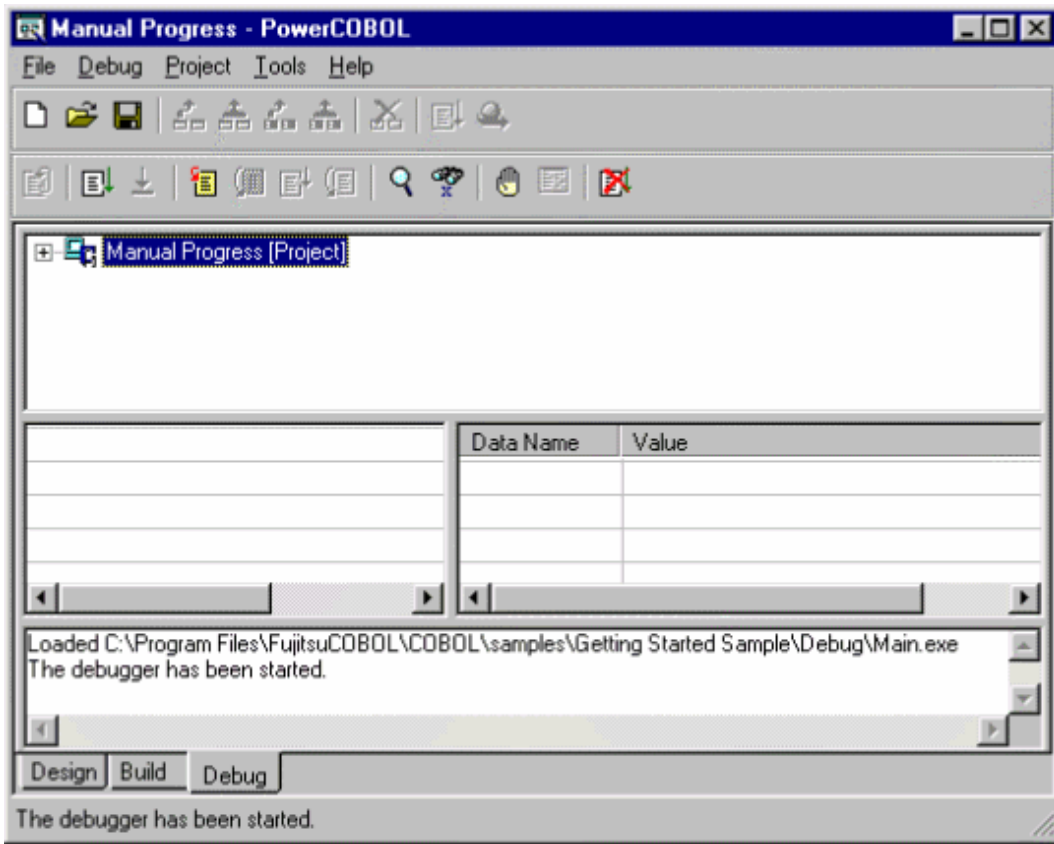
1. Return to the PowerCOBOL project manager by selecting "Manual Progress - PowerCOBOL" from the Windows task bar.
2. From the project manager menu select File, Save, to save all your changes.
3. Right-click on "Main [Module]" and select Execute from the pop-up menu. Respond "Yes" to the question "Do you want to build it?". PowerCOBOL builds an executable module and starts it executing.
4. Click on the Step button a few times to see what it does.
5. Clicking on OK, Cancel or the "X" button, closes the window.


## Debugging the Application

Hopefully the application is working as intended, but we will take a quick look at the PowerCOBOL debugger anyway. We will step execution line by line, then have PowerCOBOL interrupt execution when a data value changes, then set and run to a breakpoint.

1. Right-click on "Main [Module]" in the project tree, and select Debug from the pop-up menu. PowerCOBOL displays the debug window/tab.

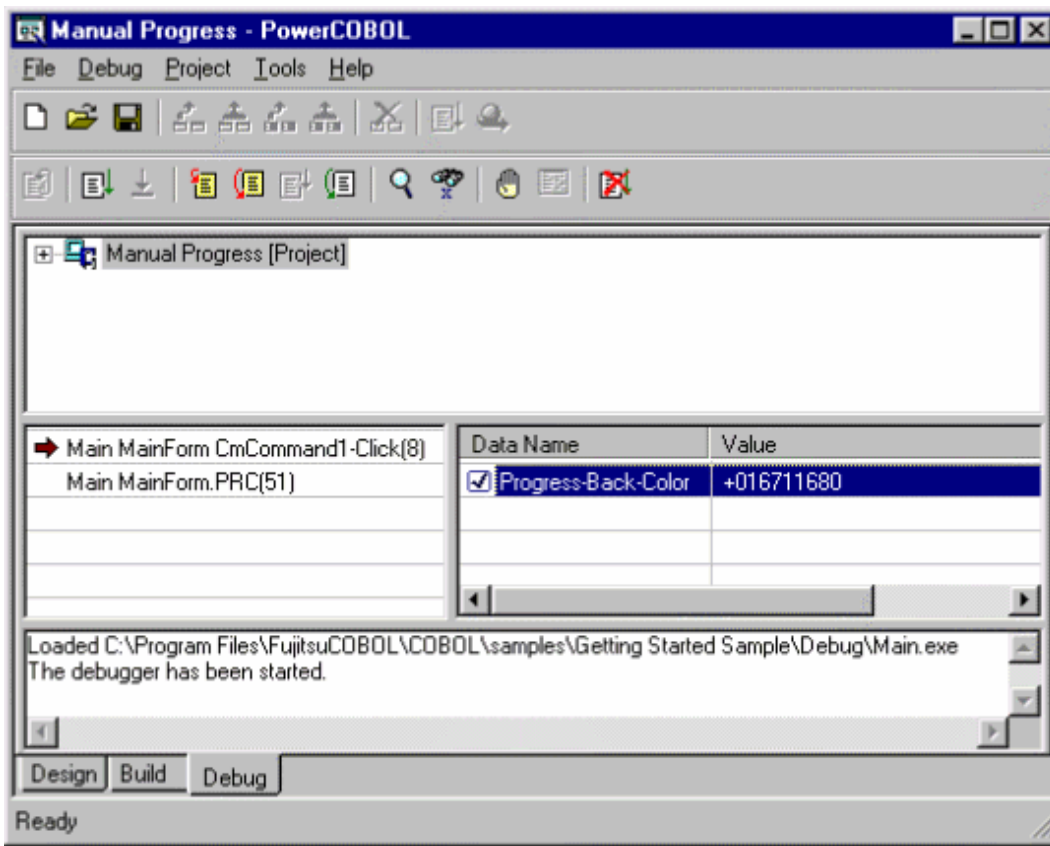
Figure 2.22 Debug tab for the Manual Progress project.





2. Click on the "Step In" button, , on the debug toolbar. This starts the code and your application window displays. Although other code has been executed to display the window, none of the code you have written has been executed so you do not yet see any code.
3. Click on the Step button in the Manual Progress Indicator Demo window. Now PowerCOBOL displays the Click event code you entered. (It is possible that you may need to select the PowerCOBOL Editor window from the Windows task bar to bring it into view.) A ">" sign to the left of the line numbers indicates the current execution line. Also notice that the middle left pane of the debug window displays the name of the procedure being executed, the current line number, and the procedure(s) that invoked this procedure.
4. Double-click on the text "Progress-Back-Color" to select that data item, and right-click on the selected text. The pop-up menu displays debug and edit functions that apply to this item.
5. Select Watch from the pop-up menu. PowerCOBOL displays the name and value of Progress-Back-Color in the middle right pane of the debug window. There is a check box by Progress-Back-Color - if you check this, the debugger will interrupt execution whenever this data item is changed.

6. Check the check box by Progress-Back-Color.

Figure 2.23 Watching "Progress-Back-Color" on the Debug tab.



7. Click a few times on the "Step In" button, , on the debug toolbar to see the execution advance a few lines and the Progress-Back-Color value change.
8. Click on the "Go" button, , on the debug toolbar, to run the code to the next break point.  
The buttons on the toolbar change to their disabled state:

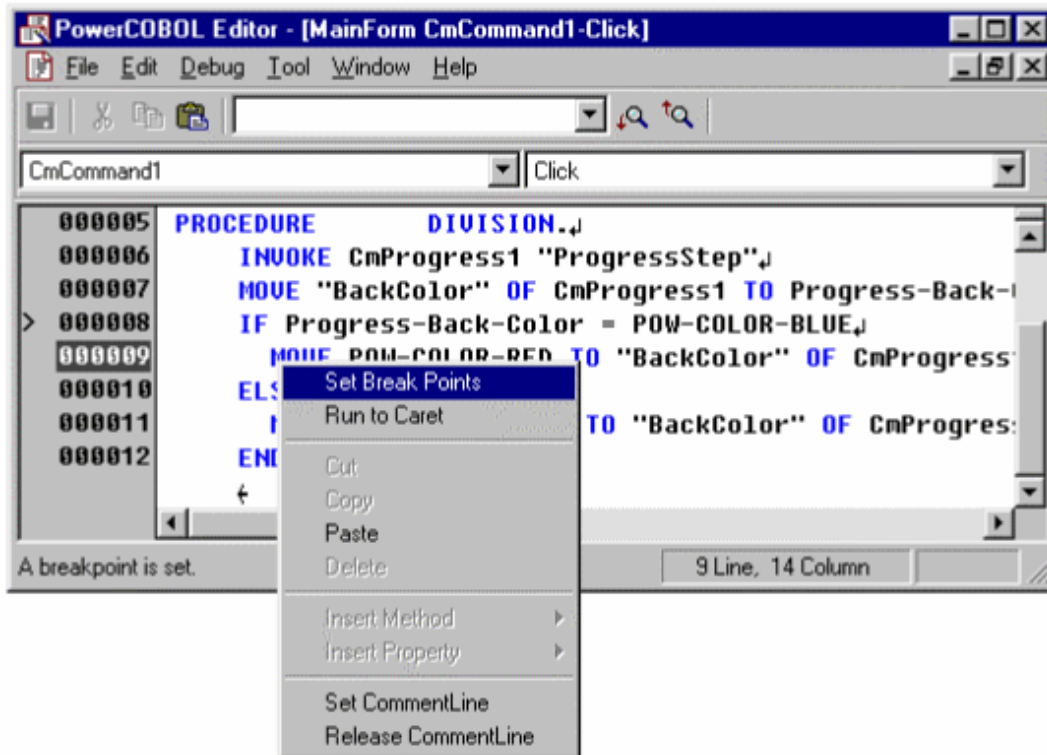



This is because control has been returned to the application window. You need to select that window (Manual Progress Indicator Demo) on the Windows task bar to bring it to the front of the screen.

9. Click on the Step button in the application window.  
PowerCOBOL displays the Click event code, this time stopped at the line after Progress-Back-Color is changed - because we indicated that execution should be interrupted when that data item is changed.  
In the Debug window Progress-Back-Color is highlighted in red to indicate that its value has been changed.
10. In the Debug window, clear the check box beside Progress-Back-Color.
11. Display the source code by selecting "PowerCOBOL Editor - [MainForm CmCommand1-Click]" in the Windows task bar.

- Left-click on one of the MOVE statements within the IF statement, then right-click and select "Set Break Points" from the pop-up menu.

Figure 2.24 Setting a breakpoint using the pop-up menu when debugging



- PowerCOBOL displays a "\*" to the left of the line numbers to indicate that a breakpoint is set.
- Click on the "Go" button, , on the debug toolbar, to run the code to the next break point and, if necessary, select the Manual Progress Indicator Demo from the Windows task bar, and click the Step button one or two times. PowerCOBOL displays the source code with execution stopped at the line where the breakpoint was set.
- To exit from debug select Debug, Debug quit, from the debug menu bar

Although this is a very brief review of the Debug feature, you can see that you are focused on your code and can monitor changes and progress within that code.

## 2.2.3 Exploring the On-Line Reference

If you select Help, Help Topic from the PowerCOBOL project manager you are taken to the PowerCOBOL On-Line Reference. This provides detailed information about all the controls, objects, events, methods and properties of the PowerCOBOL product.

If you would like to experiment further by creating more complex windows you will find this a handy reference to check on precise details of behavior.

To explore the reference:

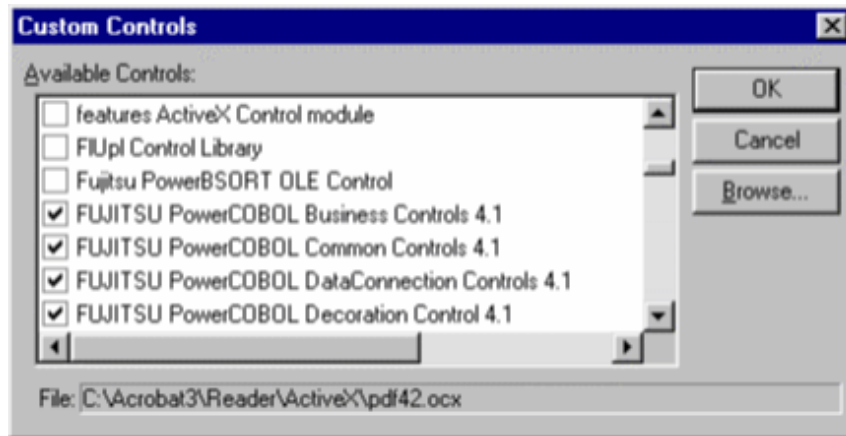
- Select the index of Contents (e.g. Controls and Objects Index, Properties Index, Methods Index, or Events Index). This opens up a list of topics on these subjects. The topics provide introductions to the particular elements and an index to the elements.
- Read the introductions if you want more information on that element.
- Open the index to find an alphabetical index to all the elements of that type.

## 2.2.4 Using ActiveX Controls

## Adding ActiveX Controls

1. To add an ActiveX control first register it to the Windows system - check the documentation provided with your ActiveX control for details on how to do this.
2. Open the form editor by right-clicking on a [Form] in the PowerCOBOL project tree, and selecting Open from the pop-up menu.
3. From the form editor menu select Tools, Custom Controls.  
PowerCOBOL displays a list of the registered controls.

Figure 2.25 Custom Controls dialog



4. Select (check) the controls you would like to use in your PowerCOBOL application and click on OK. PowerCOBOL adds the control to the Insert Control sub-menu, and adds a button to the control toolbox.

### 2.2.4.1 Using ActiveX Controls

You use ActiveX controls just like any other PowerCOBOL control. You place them on the form, create code to handle events, invoke methods and set and reference properties as you have already done with other controls in this guide.

## 2.2.5 Constructing Sample Database

---

This chapter discusses constructing a sample database that is used in PowerCOBOL sample programs. SQL Server Express is available from the Microsoft Download Center.

SQL Server Express must be installed on the local machine with the instance name of "SQLEXPRESS" in order to work with the sample programs.

Refer to the "NetCOBOL V11.0 User's Guide" > "Sample Database" about sample database tables.

The sample database can be used in the "NetCOBOL Getting Started" > "Using SQL with COBOL".

### 2.2.5.1 Creating Sample Database

#### In the case of using SQL Server Management Studio

The following steps describe how to create a sample database in SQL Server Express by using SQL Server Management Studio.

1. Start SQL Server Management Studio and connect to SQL Server Express.
2. Select the Open > File item in the File menu.
3. Select "SampleDatabase\COBOLSample.sql" of the sample programs folder in the Open File dialog box.
4. Click on the Execute button in the toolbar.
5. Verify the "COBOLSample" database is created in Databases in the Object Explorer.

## In the case of using the SQLCMD command in the system console (command prompt window)

The following steps describe how to create a sample database in SQL Server Express by using the SQLCMD command in the system console.

1. Bring up the command prompt window.
2. Move to the SampleDatabase folder in the sample program folder.
3. Set the following command.

```
sqlcmd -S .\SQLEXPRESS -E -i COBOLSample.sql
```

4. Set the following command and check the contents of the stock table are displayed.

```
sqlcmd -S .\SQLEXPRESS -E -d COBOLSample -Q "select * from STOCK"
```

### 2.2.5.2 Defining ODBC Source

Define the ODBC Source as follows from the ODBC Data Source Administrator window (bring up the Windows Control Panel and double-click the on the ODBC Data Sources item).

Driver	SQL Server
Data Source Name	COBOLSample
Server name	.\SQLEXPRESS
Default database	COBOLSample

The following steps describe how to operate in the ODBC Data Source Administrator window.

1. Make sure the User DSN tab is open, and click on the Add button. The Create New Data Source dialog box is displayed.
2. Select the SQL Server as a driver, and click on the Finish button. The Create New Data Source to SQL server dialog box is displayed.
3. Enter **COBOLSample** for the Name and **COBOL Sample Database** for the description. And enter the server name for the Server in which COBOLSample database is created. If SQL Server Express is used, select **.\SQLEXPRESS**. Click on the Next button.
4. Enter the login ID and the password if needed. If SQL Server Express is used, check the With Windows NT authentication using the network login ID is selected. Click on the Next button.
5. Click on the Change the default database to checkbox, and select the "COBOLSample" database name. Click on the Next button and the Finish button.
6. Click on the Test Connection button and check connected correctly. Click on the OK button.
7. Click on the OK button.

## Appendix A Tips

The following tips will help you use PowerCOBOL most effectively:

1. If in-doubt, check the **pop-up** menu by right-clicking the mouse button. As you'll see from the following tips, the most frequently used functions are generally found on the pop-up menus.
2. When you **open a project**, right-click on the project name and select "Expand all" from the pop-up menu. This lets you see all the objects in the project.
3. To **create or edit a form**, right-click on the form name in the project tree-view and select "Open".
4. To **edit** or view **code** or **properties** associated with an item, right-click on the item and select the desired function from the pop-up menu.
5. To see the available **methods** or **properties** for a control, copy or type the name of the control into the PowerCOBOL text editor, select the text and right-click.
6. To **preview** how your controls will work, without building the project, select Layout, Preview from the Form Editor menu.
7. To **build** a module or project select the module or project in the project tree-view and select Build [All] or Rebuild [All] from the Object menu or pop-up menu. (Note that you can go straight to executing a module - PowerCOBOL recognizes if a build is required and performs a build + execute as a single step.)
8. To **execute** your application, select the main module in the project tree-view and select Execute from the Object menu or the pop-up menu.
9. To **set run-time environment variables**, use the Run-time Environment Setup Tool (COBENVUT.EXE) which can be selected from the pop-up menus for the project or module in the project tree.



# Index

---

	[A]	
ActiveX controls.....		25
aligning controls.....		12
	[B]	
breakpoints.....		24
building.....		21,27
	[C]	
code generation.....		19
COM.....		3
controls.....		12,15,16,27
custom controls.....		25
	[D]	
debugging.....		21,24
	[E]	
editing.....		6,9
event.....		9
executing.....		21
Expand all.....		5
	[F]	
form.....		1,27
Form Editor.....		1,2
	[I]	
Insert Method menu.....		19
Insert Property menu.....		20
	[L]	
Listview control.....		8
	[O]	
on-line reference.....		24
	[P]	
pop-up menus.....		8,27
PowerCOBOL.....		2,4,5,10
preview.....		17
project manager.....		1,5
properties.....		6,8,16
	[S]	
sizing controls.....		12
spacing controls.....		12
	[T]	
text editor.....		10
tips.....		27
	[W]	
watch data.....		22
WORKING-STORAGE.....		11,18