

ASNA Visual RPG for Smarties

**By Eduardo Ross and
Julie O'Brien**

**The fast workbook way to learn to
program with Visual RPG!**

**Learn the tips and shortcuts for
developing comprehensive applications.**

**Develop an entire application by the end
of the book—in only nine easy steps!**

ASNA

Visual RPG for Smarties

Information in this document is subject to change without notice. Names and data used in examples are fictitious unless otherwise noted.

No component of ASNA Visual RPG for Smarties may be reproduced, disassembled, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form without the written permission of ASNA (Amalgamated Software of North America).

Copyright ©1997-2002 ASNA - Amalgamated Software of North America. All rights reserved.

Release 4.0 – May, 2002

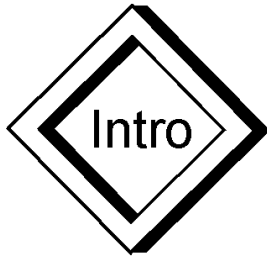
CONTENTS

| | |
|---|-----------|
| Introduction - Learning ASNA Visual RPG..... | 1 |
| Getting Started..... | 2 |
| Using the Tutorial..... | 2 |
| Installing ASNA Visual RPG..... | 3 |
| Starting the IDE..... | 4 |
| The Application..... | 6 |
| Tips in using the Message and Editor Windows | 8 |
| | |
| Step 1 - Forms and Properties..... | 9 |
| Creating a New Project..... | 10 |
| Form Window..... | 10 |
| Property Window | 11 |
| Running the Program..... | 12 |
| Changing the Form's Appearance | 14 |
| Step 1 Summary | 16 |
| More Information | 17 |
| | |
| Step 2 - Adding Controls | 19 |
| The Control Palette..... | 20 |
| Control Properties..... | 22 |
| Changing Properties at Run-Time | 24 |
| The Editor..... | 24 |
| Spec Formats | 25 |
| Referring to a Property within Code | 26 |
| Stretchable Columns..... | 27 |
| The 'Value' Property | 28 |
| The Project Window..... | 28 |
| Saving Your Project | 29 |
| Step 2 Summary | 31 |
| More Information | 32 |
| | |
| Step 3 - Responding to Events..... | 33 |
| Event Driven vs. Traditional Programming..... | 34 |
| AVR Program Structure | 35 |
| Event Subroutines in the Editor..... | 36 |
| Command Buttons | 37 |
| The Click Event..... | 38 |
| Adding Comments..... | 39 |
| Formatting Data using IOFields | 40 |
| Step 3 Summary | 43 |
| More Information | 44 |

| | |
|---|------------|
| Step 4 - Database Access | 45 |
| Selecting and Manipulating Controls | 46 |
| Selecting Multiple Controls..... | 46 |
| Program Behavior..... | 50 |
| File Layout..... | 51 |
| Special Continuation Lines..... | 52 |
| Message Box | 53 |
| Step 4 Summary | 55 |
| More Information | 56 |
| | |
| Step 5 - Methods | 57 |
| Disabling a Control | 58 |
| Methods | 59 |
| Setting the Tab Order | 62 |
| Displaying Pictures..... | 64 |
| Step 5 Summary | 66 |
| More Information | 67 |
| | |
| Step 6 - Menus and Forms | 69 |
| Providing a Menu to the Users | 70 |
| The Menu Editor..... | 70 |
| Components of the Menu Editor | 71 |
| Steps to Create Menus using Menu Editor | 74 |
| Showing a Second Form..... | 77 |
| Step 6 Summary | 81 |
| More Information | 82 |
| | |
| Step 7 - Using a Subfile | 83 |
| Application Structure | 84 |
| The Subfile Control..... | 87 |
| Subfile Properties | 94 |
| Subfile Events..... | 96 |
| Step 7 Summary | 97 |
| More Information | 97 |
| | |
| Step 8 - Graphing Data | 99 |
| Adding a Graph | 100 |
| Receiving Parameters in a Subroutine..... | 104 |
| Overlapping Windows..... | 105 |
| Step 8 Summary | 110 |
| More Information | 110 |
| | |
| Step 9 - Printing | 111 |
| Starting and Using Acceler8DB Print File Editor | 112 |
| Printing in RPG | 118 |
| Step 9 Summary | 121 |
| More Information | 122 |

| | |
|---|------------|
| Appendix A - Converting AS/400 Applications..... | 123 |
| AS/400 Display File (Import Menu)..... | 123 |
| Importing and Converting AS/400 Display Files | 126 |
| | |
| Appendix B - Caviar Free Format..... | 129 |
| Listing for frmMonthRevenue..... | 129 |
| | |
| Index..... | 133 |

This Page Intentionally Left Blank

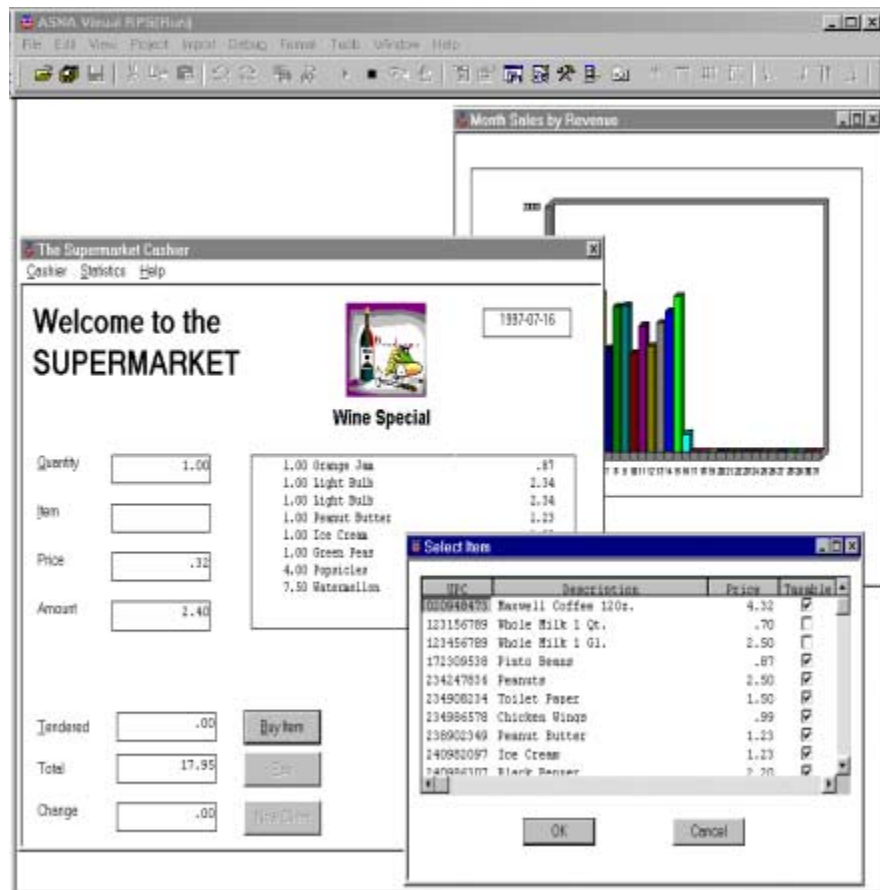


Learning ASNA Visual RPG

This book teaches the fundamentals of programming for Windows using the ASNA Visual RPG (AVR) Integrated Development Environment (IDE) in a “tutorial” format. The tutorial creates a **Supermarket Check-Out Cashier** application that is developed in nine progressive steps. Each step in the application represents a step up in the tutorial’s lessons.

After completing Step 9, you will have a Windows application, including features such as menus, dialog boxes, pictures, subfiles, graphs and more.

At the end of Step 9, the application will look like the following:



Getting Started

This tutorial assumes that you are familiar with RPG/400 and have some prior experience using Windows applications.

Using the Tutorial

This tutorial is designed as a “self-learning” tool and includes 9 chapters, where each chapter is a “Step” in which a portion of the application is created. Each step will build on the previous step, so it is recommended that you start at Step 1 and progressively work towards Step 9. Each step should take you approximately 30 minutes to an hour and a half. The approximate time to complete each step is included at the beginning of each chapter.

At the beginning of each chapter, you will find:

- A summary of the topics that will be covered in that step.
- The approximate time to complete that step.
- A visual representation of what the application will look like at the completion of that step.

At the end of each chapter, you will find:

- A reference section of each task performed in that step, how to perform that task, and the button or shortcut key that is used to perform that task (if any).
- A section indicating where you can find more information about the topics discussed, both in the Visual RPG on-line help and the Visual RPG manuals.

Source Files

The complete source files for the tutorial steps can be found in **\Program Files\Asna\Learnavr40** folder.

Each step of the tutorial is contained in a sub-folder named **Step x** , where x is a number from 1-9, representing the step of the tutorial that is included in that folder. For instance, you can find Step 1’s complete source files in the folder titled **Step1**.





*Even though this tutorial is intended for each step to be done in succession, you can open the source files of the step **prior** to the one you wish to complete. For instance, if you want to do Step 7 – Using a Subfile, you would open the project in folder Step6.*

Saving Your Work

When you save your work for a particular step, place it in the directory called **Cash** instead of saving it directly onto **Stepx**, so the original source files do not get overwritten, and you can compare your application to that of the tutorial.

Manual Conventions

The following is a list of conventions that are used throughout the manual representing particular functions.

- A  beside a headline indicates a step-by-step process that you are to perform.
 - Numbered lists (1, 2, 3, and so on) indicate steps that you should follow.
 - A smaller check mark () indicates a single task for you to do.
 - Menu options, buttons, controls and keys that you are to select, along with words that you are to type will appear in **boldface**.
-
- When instructions tell you to click a button in the toolbar or a control in the control palette, a picture of the button or control will usually be shown in the left margin, next to the instructions. The Run icon to the left of this paragraph is an example.



Run button



The graphics in this book containing “Forms” have been changed to a white background for easier readability when the book is printed or copied. Note that the forms will appear gray when the project is opened.

Installing ASNA Visual RPG

The following section explains how to install Visual RPG IDE on a PC running Windows. The Visual RPG IDE installation will automatically install the PC version of Acceler8DB. The installation steps are as follows:

1. Insert the **ASNA Product Suite Installation CD** into your CD drive. If your CD drive has AutoPlay, a screen will automatically appear.
 - Click on **Products - ASNA Visual RPG and Caviar - Install Visual RPG** to continue the installation process.

Note from this screen that you can also click on **View Installation Notes** and/or **View Release Notes** from the same screen in which you install Visual RPG to find information specific to this particular release of AVR.

- If you do not have AutoPlay installed on your CD drive, start Microsoft Windows and from the Start Menu, select **Run**. During the installation, a prompt for an installation key will appear, and depending upon the key entered, either the full version or a trial version will be installed. At the prompt, enter the following:

E:\AVR40\SETUP.EXE

Installs **either** the full version or trial version of Visual RPG, depending upon the install key entered.

where 'E' is the CD-Rom drive.

2. After the setup file is located, the Welcome screen will display. Follow the instructions on the screen.

Starting the IDE

The IDE (Integrated Development Environment) contains all the tools you need to build powerful GUI (Graphical User Interface), event-driven applications for Windows quickly and efficiently.

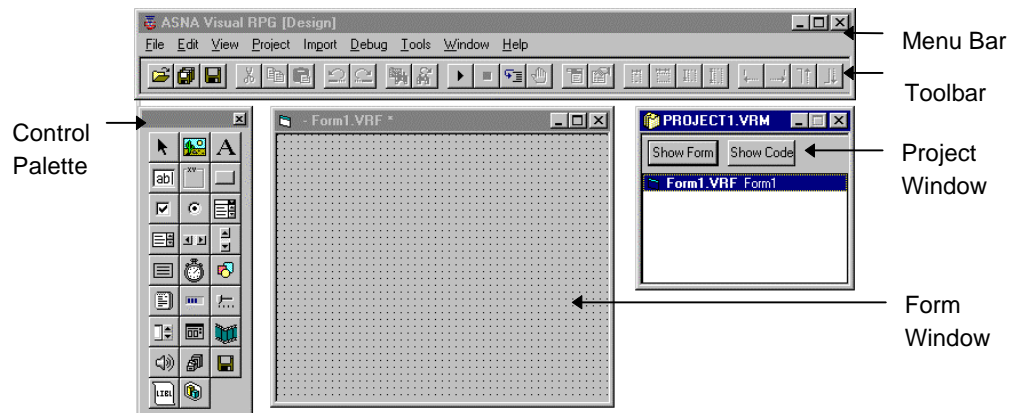
A new program folder for Visual RPG was automatically created when Visual RPG was installed.

Follow the steps below to start Visual RPG:

✓ To Start ASNA Visual RPG's IDE

- Click the **Start** button, select **Programs**, point to **ASNA Product Suite**, point to **ASNA Visual RPG 4.0**, then click on **ASNA Visual RPG**.

The main elements that make up the IDE are displayed after executing ASNA Visual RPG IDE, as shown below (note that the options in the Title Bar and Control Palette may change from release to release).



A brief description of each component of the IDE is listed below. However, each component will be described in more detail throughout the tutorial.

Control Palette

The control palette located on the left contains controls you will add to your forms.

Toolbar

The toolbar contains icons that represent the major functions to perform within Visual RPG. The toolbar can be moved, resized, or re-docked into the top of the Visual RPG window, underneath the menu options.

Project Window

The project window a current list of all files in the project.

A default project (PROJECT1.VRM) is automatically created every time you start Visual RPG with FORM1.VRF displayed in the project window.

Form Window

The form window is the window in which you customize to create your application by adding controls to it.

A blank form (Form1) will automatically appear every time you start ASNA Visual RPG with PROJECT1.VRM displayed in the project window.

Windows Desktop

Notice that the Windows Desktop is still in the background. Since AVR is a Windows Program, each Window of the IDE can be opened, closed, minimized, or maximized while another application is running.

- ✓ To hide the Windows Desktop, double-click anywhere on the ASNA Visual RPG [design] Title Bar.

The background will now be the color set for the Application Background in the Windows System color settings. This color can be changed by selecting **Start** button - **Control Panel** - **Display** - **Appearance**.

- ✓ Double-clicking on the Title Bar again will return the Windows Desktop setting to the background.

The Application

Throughout the tutorial, you will be building a simplistic version of a supermarket check-out cashier. Let's open and run the last step (step 9) to see what your program should look like when it is complete.

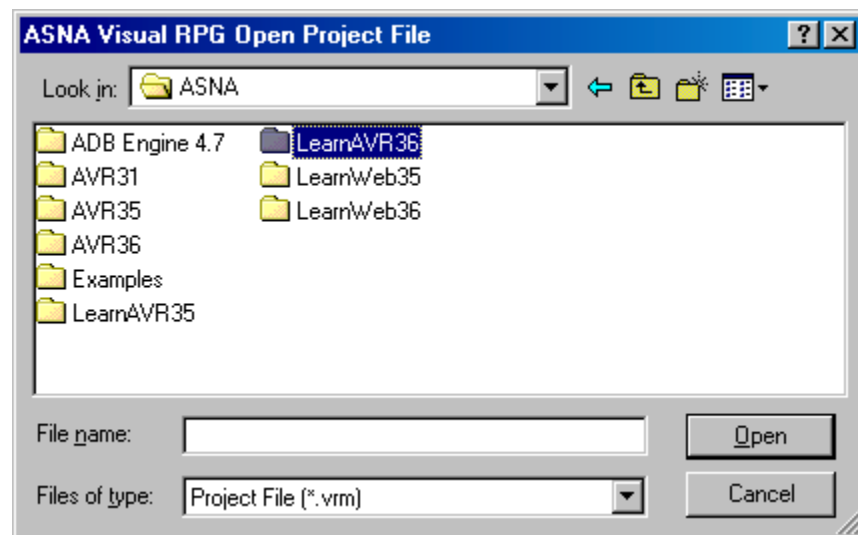
To Open an Existing Project

1. Click the **F**ile menu by using the mouse.

The contents of the File menu appear.

2. On the File menu, click the **O**pen Project option.

The Open Project dialog box appears. This dialog box allows you to open any existing AVR program on your hard disk, attached network drive or floppy disk.



3. In the \ASNA folder, double-click on the **LearnAVR40** folder.

The 9 directories containing the tutorial steps will be displayed.

4. Double-click the **Step9** folder.

5. Select the **Cashier.vrn** project and then click the **O**pen button.

The Cashier.vrn project file loads the user interface forms, the properties and the source code of the program **Cashier**.

6. Click the **R**un button on the AVR toolbar.

The IDE will compile the program, and the application will start running, displaying its first window, as shown below.

7. Within the program, you can do the following:
- Press **F4** or select **Q**uery from the **C**ashier Menu to see a list of available items for sale.
 - Select an item and click the **O**K button.
 - Click the **B**uy Item button.
8. Play with the application for a while, selecting the menu options and buttons. When you are finished, select the **E**xit option from the **C**ashier menu.



If you wish to close Visual RPG at this time, select Exit from the **F**ile menu. Follow the instructions on page 4 to start Visual RPG. Otherwise, continue with Step 1.

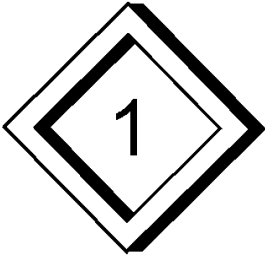
Congratulations:

You have just run your first application in AVR!

Continue with Step 1 to create a form, set some form properties and run your application.

Tips in using the Message Window and Editor Window

- **To Display Help Text for an Error Message**, select the error or line in the message window in which you want to display Help text and press **F1**. You will go directly to the On-Line Help displaying the error message, it's description, causes and resolution.
- **To go directly to a line in source code**, select the line in error in the message window, then either **double-click** or press **F6** to automatically take you to the line in your code that caused the error message. The error message will then display in the left corner of the status bar, displaying first the error number, then the error description. A blue arrow will display to the left of column 1.
- **To copy a block of text**, highlight the cell or lines to copy. Select the **Ctrl+C** keys. Paste the text by going to the desired location and selecting the **Ctrl+V** keys.
- **To stretch a RPG Style Fixed Format column**, place the mouse pointer over a blue arrow below the column headings until it turns into a double arrow. When a double arrow appears, click and drag the mouse until the desired width is achieved. (Only columns marked with a blue arrow can be sized).
- **To go to a particular line in the editor**, select **Edit - GotoLine**, or press the **Ctrl+G** keys.
- **To resize the editor window and store in memory**, move your arrow to any edge of the editor window until the cursor turns into a double arrow. Click and drag the mouse until the editor window is of desired size. The editor window will remain the new size until it is manually changed.
- **To display a list of spec formats for the selected editor**, press **F2** for a listing of the supported RPG 400 or ILE RPG Spec Formats. With the Spec Formats listed, you can select one, and it will automatically insert the appropriate columns for the selected Spec Format.
- **To change the editor type**, select **Tools - Options - Editor** tab, and select the desired default editor.
- **To display the right-mouse popup menu**, click on the right-mouse button anywhere within the editor. A pop-up menu will display. Select a menu option or enter the assigned shortcut keys to the right to quickly carry out the command.
- **To display the current value of the selected expression**, select the expression from the editor window, then select the **Quick Watch** command, press the **Shift+F9** keys, or click the right mouse button and select **Instant Watch** from the drop-down menu..
- **To change Fixed Format editor to default to RPG/400 code or RPG IV code**, select **Options...** from the **Tools** menu or press the **Ctrl+O** keys, then click on the Editor Tab.
- **To assist you with where you are or what to enter in the editor**, use the Status Bar. It is located on the bottom line of the editor window, and contains the following components; cursor status, cursor position, editor type, insert/overwrite mode, caps lock, num lock and current time.



Forms and Properties

What you will learn in Step 1:

- How to create a new project.
- What a form is.
- What a property is.
- How to display the properties of a form.
- How to run/stop a program.
- How to change the appearance of the form's Title Bar and specifying which buttons are displayed.

Approximate Time to Complete Step 1:

30 minutes.

What the Application will look like after completing Step 1:

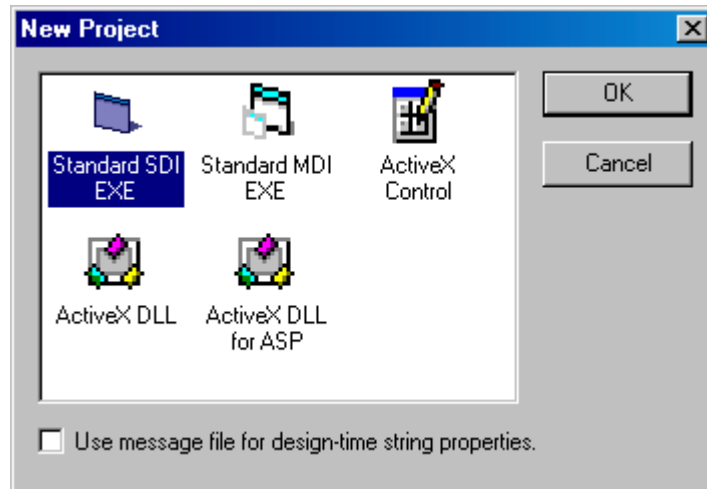


Creating a New Project

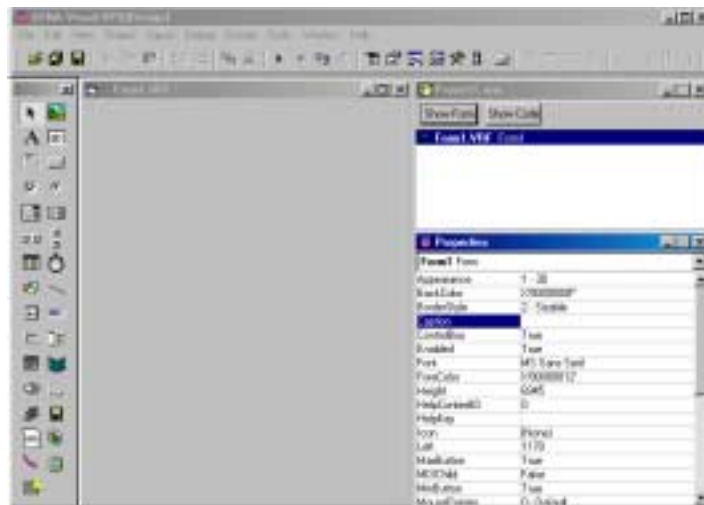
Now that you have read the Introduction chapter, you are ready to begin creating the application.

Let's Start a New Project

1. To start a new project, select the option **New Project** from the **File** menu. A dialog will display with different project types to create, as shown below:



2. Select **Standard SDI EXE**, which creates a standard single document interface executable file, followed by the OK button.
3. The IDE will then display with one blank **form** on it (Form1), as shown below.



Form Window

In AVR, a **form** is a window you customize to create the user interface of your program. A form can contain menus, buttons, ListBoxes, scroll bars, and any of the other items you have seen in a typical Windows-based program.

When you start the IDE, a default form named **Form1** appears with a standard grid (a group of regularly-spaced dots). The grid assists you in creating and aligning interface elements.

In Windows, interface elements are called **Controls**. In the Cashier application, several controls such as **Labels**, **IOFields**, **Command Buttons**, and **Images** were used to create the interface.

Property Window

A form and every control has a set of **properties** which allow you to change their appearance and behavior.

The property window lists all the property settings for the selected form or control. The property name is listed in the left column of the window, and the current setting for each property is listed in the right column.

The property window scrolls like a regular ListBox, and the properties are listed in alphabetical order.

Let's Change the Caption in the Form's Title Bar



1. To view the properties for the form, select **Property** from the **View** menu or press the **F4** function key.



2. Double-click the **Caption** property (the property name in the left column). Both **Caption** (left column) and **Form 1** (right column) are highlighted and the cursor blinks to the right of Form1, ready for you to enter a new caption.
3. Type **The Supermarket Cashier**, then press Enter.

The setting of the Caption property changed from 'Form1' to 'The Supermarket Cashier'. Notice that the caption appeared on the form's Title Bar as it was typed.

4. There are 4 properties that control the size and placement of a window on the desktop.

Top

Left

Height

Width

Top and **Left** control the position of the window, while **Height** and **Width** control its size.

You can change the position of the form by doing one of the following:

- Enter the new values in the property window as you did the **Caption** property.
- Click anywhere in the form's Title Bar and “drag” the form to a new position.
- Move the mouse pointer to any side or corner of the form until a double arrow appears. Click and drag the mouse until the form is the size you want it.

Running the Program

Let's **Run** the program to see the form. You may be asking yourself, 'What program?'. You are right in doubting that you have a program, since you have not written a single line of code yet. Because you have already defined the **appearance** of your window, you can run it to see what it looks like when the program is executing.

The IDE can be operating in two different modes: **Design** and **Run**.

- Design mode is used to design the user interface and write the program code.
- Run mode is used to test the application.

Let's Run the Program



Run button

1. Select **R**un from the **D**ebug menu, press the **F5** key, or select the Run button from the toolbar.

The IDE will compile your program and display the window with the proper caption, as shown below.



Notice how the IDE changes its own title from **ASNA Visual RPG [Design]** to **ASNA Visual RPG [Run]** when it is running the application. You will usually run your application within the IDE before creating an executable program (.EXE) because it gives you access to the debugger and improves the Edit/Compile/Run cycle.

Whenever you are in Run mode, you have two options to go back to Design mode.

- The first option is to close the application by clicking the **C**lose button in the Title Bar.
- The second is to request the debugger to stop the execution of the program by clicking the **S**top button on the IDE toolbar.



↑
Stop button



Stop button

2. Select the **S**top button in the toolbar to stop running the application.

Changing the Form's Appearance

You can allow the user to relocate the form by dragging the Title Bar to a new position on the desktop. The window can also be minimized, maximized and closed by clicking on the respective control buttons in the top left corner of the window. You can even permit the user to resize the window by dragging any of the edges of the window.

The AVR run-time environment and Windows handle the responsibility of responding to all requests for moving and resizing the window. There is no user code involved in these functions.

The form's Title Bar is controlled via two form properties: **ControlBox** and **BorderStyle**.



The **ControlBox** property indicates whether a Control Menu Box and the Minimize, Maximize and Close buttons are displayed on a form at run-time.

The **BorderStyle** property determines whether the form can be sized, the control menu box is displayed, or the minimize or maximize buttons are displayed.

BorderStyle Property Settings for a Form:

| Setting | Description |
|---------|--|
| 0 | None - No border or related border elements. |
| 1 | Fixed Single. Re-sizable using only Maximize and Minimize buttons. |
| 2 | Sizable. Re-sizable using any of the optional border elements listed for setting 1. (Default) |
| 3 | Fixed Dialog. Not re-sizable. Can include Control-Menu box and Title Bar; cannot include Maximize and Minimize button. |
| 4 | Fixed Tool Window. Not re-sizable. Can include Control-Menu box and Title Bar; cannot include Maximize and Minimize button. Under Windows 95, displays the Close button and displays the title bar text in a reduced font size. The form does not appear in the Windows 95 task bar. |
| 5 | Sizable Tool Window. Re-sizable. Can include Control Menu box and Title Bar; cannot include Maximize and Minimize button. Under Windows 95, displays the Close button and displays the title bar text in a reduced font size. The form does not appear in the Windows 95 task bar. |

Both `ControlBox` and `BorderStyle` are properties that do not reflect their characteristics when the IDE is in design mode, but only show their effect when the program is in run mode.

Let's Change Form Properties





1. Give it a try and have some fun playing with the form properties.
2. When you are done, set the **ControlBox** property to **True** and the **BorderStyle** property to **3 - Fixed Dialog**.

Congratulations:

You have just completed Step 1, in which you created a form and set form properties.

In Step 2, you will add some controls to your form and set properties for those controls.

Step 1 Summary

| To | Do This | Button/Keys |
|--|--|---|
| Create a new project | Select N ew P roject from the F ile menu. Then select S tandard S DI E XE, followed by the OK button. | |
| Open the property window | Select P roperty from the V iew menu, or press the F4 key. |  |
| Change the caption of a form | Double-click the C aption property in the property window and enter a new caption. | |
| Move a form to a new position | Click anywhere in the form's Title Bar to move the form to a new position. | |
| Resize a form | Move the mouse pointer to any side or corner of the form until a double arrow appears. Then click and drag the mouse until the form is the size you want it. | |
| Run a program | <ul style="list-style-type: none"> • Select Run from the Debug menu. • Press the F5 key. • Select the Run button from the toolbar. |   |
| Add a control menu box in the Title Bar | Set the C ontrol B ox property to T ru e . | |
| Add the minimize, maximize and close buttons in the Title Bar | Set the B order S tyle property to 2 - Sizeable. | |
| Prevent the form from being resized and exclude the Maximize and Minimize buttons in the Title Bar | Set the B order S tyle property to 3 - Fixed. | |
| Stop a program | Click the S top button in the toolbar. |  |

More Information

To find more information on the topics introduced in Step 1, refer to the following help file topics.

AVR Help File Topics:

- Form window, using
- BorderStyle property
- ControlBox property
- Property window
- Running a project
- Creating a new project
- Stopping execution

This Page Intentionally Left Blank



Adding Controls

What you will learn in Step 2:

- How to add controls to the form.
- How to set properties at design-time.
- How to open and use the code editor.
- How to write code to set properties at run-time.
- How to set the Value property.
- How to open and use the project window.
- How to save your program.

Approximate Time to Complete Step 2:

45 minutes.

What the Application will look like after completing Step 2:

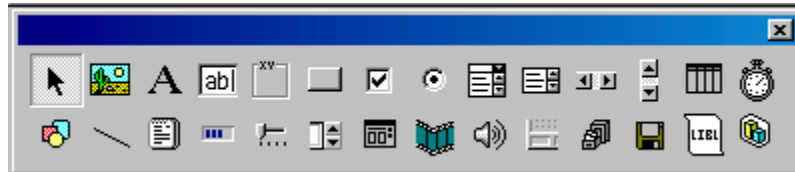
| | |
|----------|---|
| Tendered | <input type="text" value="0000100.00"/> |
| Total | <input type="text" value="0000025.00"/> |
| Change | <input type="text" value="0000075.00"/> |

The Control Palette

Now that we have a form with the right caption, size and style, let's add some controls, like **Labels** and **Fields** to it.












Controls are located in the **control palette** in the IDE. The control palette contains buttons representing each available control.

















- ✓ If the control palette is not in view, select **Control Palette** from the **View** menu or press **Ctrl+T**.



By default, the control palette will be located on the left side of AVR. However, it can be moved or resized to be horizontal, as shown above.

A brief description of the initial Visual RPG controls found in the control palette is listed below.

| Icon | Control | Description |
|---|------------------------------|---|
|  | Arrow | Moves or resizes a control after it has been placed on the form. |
|  | Image | Allows you to place a picture onto the form. |
|  | Label | Displays text a user cannot interact with or modify. |
|  | Check Box | Displays a True/False or Yes/No Option. |
|  | ComboBox | Allows text to be displayed and entered and a drop-down list to be displayed from which the user can select values. |
|  | Command Button | Carries out a command when a user selects it. |
|  | Frame | Displays a rectangle with text at the top to logically group controls on the screen. |
|  | Horizontal Scroll Bar | Allows you to easily search through a list of items. |
|  | Vertical Scroll Bar | Allows you to easily search through a list of items. |
|  | IOField | Allows entry and display of text. |
|  | ListBox | Displays data in a series of row and columns. |

| | | |
|---|----------------------|--|
|  | Option Button | Allows selection of one option from a group of options. |
|  | Subfile | Provides a flexible, capable tool for browsing and editing tabular data that reduces the requisite programming to a minimum. |
|  | Timer | Executes code at regular intervals by causing a Timer event. |
|  | Shape | A graphical control displayed as a rectangle (0), square (1), oval (2), circle (3), rounded rectangle (4), or rounded square (5). |
|  | OSFile | Allows you to create a windows file, or to access an existing one. |
|  | Progress Bar | Displays the progress to complete a task as it is occurring. |
|  | Track Bar | A window containing a slider and optional tick marks. The Track Bar control is useful when you want to select a value or a set of consecutive values in a range. |
|  | Up/Down | Allows the user to scroll up and down to search through a list of items. |
|  | Common Dialog | Provides a standard set of dialog boxes for operations such as opening, saving, and printing files or selecting colors and fonts. |
|  | Line | A graphical control displayed as a horizontal, vertical, or diagonal line. |
|  | AVI | This control makes it easy for you to play and get information about AVI files. |
|  | Wave | This control makes it easy for you to play and get information about WAV files. |
|   | | Provides a window, usually at the bottom of a parent form, through which an application can display various kinds of status data. |
|  | Database | Allows you to create, change and delete databases and database names. |
|  | Archive | Allows you to save and restore archive files, as well as to get the count and contents of an archive. |



Library List

Allows you to get and set a library list, as well as to add, remove, clear and initialize a library list.



Graph

Allows you to easily create graphs. It is a third-party OCX control by Pinnacle.

Control Properties

Just like forms, controls have properties that govern their appearance and behavior. The set of properties available for each control depends on the control type.



All controls have the following properties: **Left, Index, Name, Tag, Top** and **WhatsThisHelpId**.

Most controls also have **AllowClearReset, Enable, Height, HelpContextID, HelpKey, TabIndex, TabStop, Visible** and **Width**.

There are other properties that are unique to a control type like the **Default** property of CommandButtons or the **EditCode** property of IOFields.

As mentioned before, every control has a property called **Name**. This is a special property in that it does not control the appearance or the behavior of the control, but gives the control a name by which it can be referred to within the RPG program. Just like variable names, control names should be significant to their use. In AVR, it is customary to append a short prefix (usually 2 or 3 characters) to the name, thus the OK button would be called **btnOK** and the cancel button **btnCancel**, the Company label **lblCompany** and the Balance IOField **ioBalance**.

Consider the Following When Setting New Property Values:

- For a property that requires entering numbers or text, just type the information in the Settings Box and press Enter.
- For a property that has enumerated or boolean values, click the down arrow  to the right of the Settings Box and select the option you want from the ComboBox or double-click the property Name to cycle through the list of available choices.
- For some properties, a button next to the Settings Box will appear with . Clicking this box will display a dialog box appropriate to the property being set.

Getting Help on a Property

- ✓ To get information on the capabilities and use of a property, click on the desired property in the property window and press **F1**. A help window will pop-up, displaying information about the property.
- ✓ You can also obtain information about a control by clicking on the control and pressing **F1**. A sample is shown on the following page.

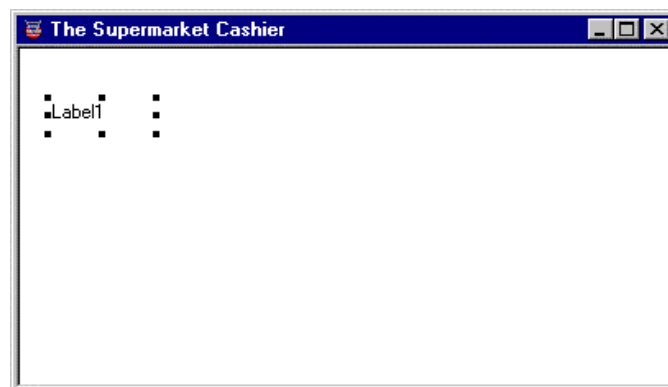


✔ Lets Add a Label to our Form



Label control

1. Click the **Label** control on the control palette.
2. Move the pointer onto the form. (Notice that the pointer becomes a cross-hair (+)).
3. Place the cross-hair where you want the upper-left corner of the Label to be.
4. Drag the cross-hair to the right and down until it is the size you want. (To drag the cross-hair, hold the left mouse button down while you move the mouse).
5. Release the mouse button. The Label will appear on the form, as shown below.



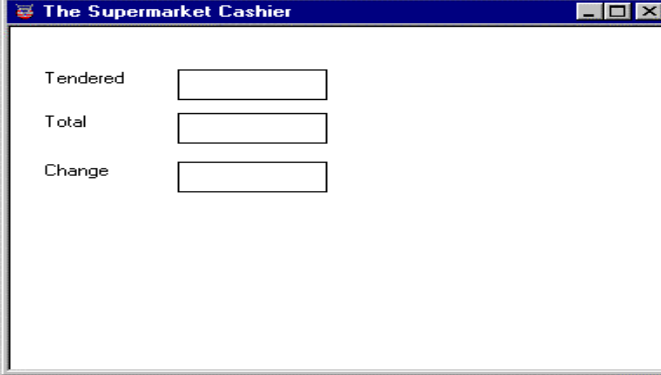
You can make the property window visible by pressing **F4**, or by selecting **Property** from the **View** menu. The property window should now show the properties for the new label and not the properties for the form.

Change the **Caption** property to **Tendered** and the **Name** property to **lblTendered**.

Lets Add IOFields to our Form



1. Add an IOField next to the label. An IOField allows text to be entered and displayed.
2. Set its **Name** property to **ioTendered**.
3. Add two more labels: **lblTotal**, **lblChange** and two more IOFields: **ioTotal** and **ioChange**. Your form should now look like the following:



The screenshot shows a window titled "The Supermarket Cashier" with a white background and a blue title bar. Inside the window, there are three rows of text labels followed by empty rectangular input fields. The first row has the label "Tendered", the second row has "Total", and the third row has "Change".

Changing Properties at Run-Time

Most property values can be modified at both design-time and at run-time. There are, however, some properties that can only be set in one of these modes. The **Name** property can only be set at design-time. That is, you cannot **rename** a control at run-time.

Other properties like the **Value** property of an IOField can only be modified at run-time. Properties of this type are **not** shown in the property window since you cannot set them at design-time.

To get a list of all the properties available for a control, click either on the button on the control palette or on a control on the form, then press **F1**. The help topic for the control will display. Click on the **Properties** link. A listing of all the properties available for that control will display. The properties may be separated into design-time and run-time, design-time only (indicated with an *), and run-time only.

The Editor

So far, we have changed property values at design-time using the property window and the mouse.

To change a property at run-time, you have to write some code. Included in the IDE is a powerful **Editor** in which you will enter and edit your code.

AVR contains two editors, **RPG Style Fixed Format** and **Caviar Free Format**.

The **RPG Style Fixed Format editor** also allows you to enter Caviar Free Format by adding a blank, or blank spec in column 1. The RPG Style Fixed Format editor can contain any combination of Caviar Free Format and RPG style Fixed Format. The default editor is RPG Style Fixed Format.

- To enter code in RPG Style Fixed Format, enter a spec type in the first column.
- To enter code in Caviar Free Format, enter a blank in the first column.

When the **Caviar Free Format** editor is selected, you can only enter code in Caviar, and it cannot be converted to RPG Style Fixed Format.



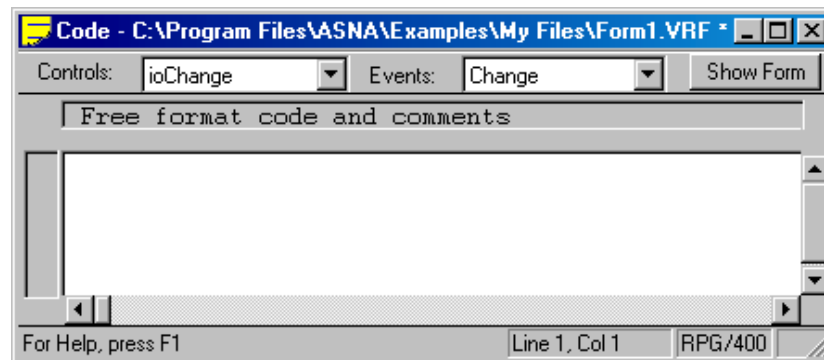
*The RPG Style Fixed Format Editor is the default editor, in which you can enter both Fixed and Caviar Free format code. The editor type is changed by selecting Options from the Tools menu, then selecting the **E**ditor Tab. Refer to the Visual RPG Help file for more information. When entering code in RPG Style Fixed Format, you can also select to use the **RPG/400** or **RPG/IV** language.*

✓ Let's Open the Editor

1. Select the IOField on your form called **ioChange** (if not already selected).
2. Open the editor by selecting **Code** from the **V**iew menu, the **Show Code** button in the project window, or the **F7** function key. The following editor window will display.



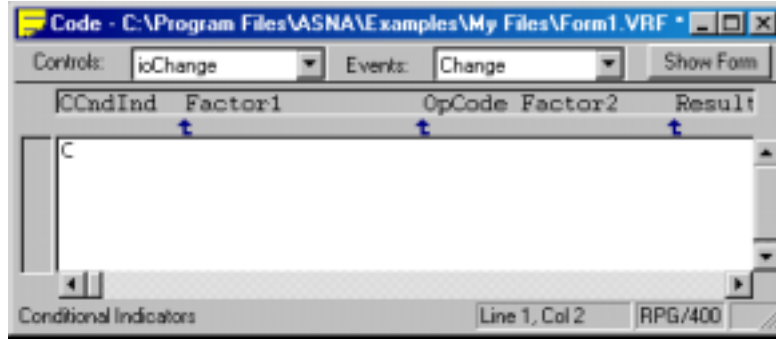
View Code



Spec Formats

Notice the editor does not show columns 1-5 (Sequence) since they have no meaning to AVR. So the first column shown is actually the **Spec** column.

- ✓ Type a 'C' in the first column to request a **Calc Spec** on the first line. You will notice how the Column Headings change to show the name for each of the C-Spec columns.



F2
Spec Types

- ✓ You can press **F2** in the first column for a listing of the supported Spec Formats.

The following are the available 400/RPG Spec Format types:

Free Format - Enter Caviar Free Format code
 E-Spec - Extension
 C-Spec - Calculation
 F-Spec - External File
 K-Spec - Continuation Line for F-Spec
 S-Spec - Data Structure
 B-Spec - Data Structure SubField
 N-Spec - Named Constant

Typing the Spec type on the first column will yield a line of that type with the column headings set appropriately.

To move from one column to the next, use the **Tab** key.



To get a listing of RPG/IV spec formats, first change the editor type, then select F2 in the first column. Refer to the AVR help file for more information.

Referring to a Property within Code

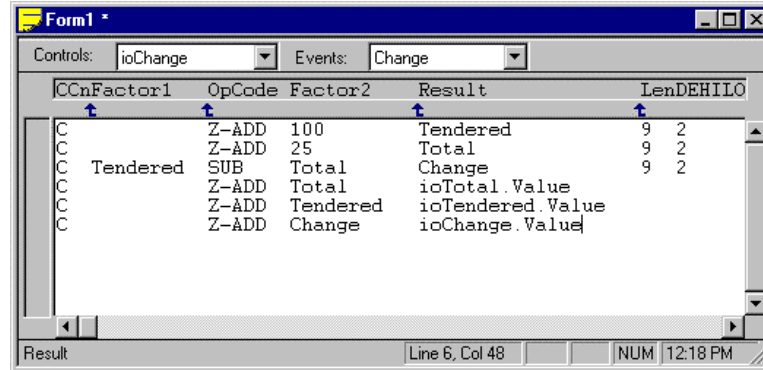
To refer to a property within the code, type in the **Name** of the control, followed by a period and the property name. For instance, to refer to the **Caption** of a label, you would type:

lblTotal.Caption

To access the **Value** property of an IOField, you would type:
ioChange.Value

✓ Let's Refer to a Property in Code

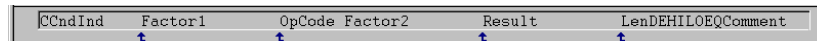
1. Enter the following code to initialize the **ioTendered** and **ioTotal** fields with 100 and 25 respectively, and the **ioChange** field with the subtraction of the two, as shown below.



Stretchable Columns

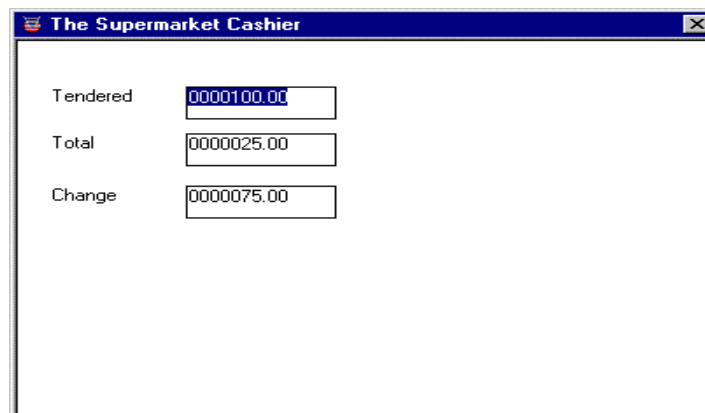
The fixed-size columns of the RPG syntax presented a problem to designers of AVR when it came to referring to properties for controls. As we mentioned, accessing a property meant typing into the Factor or Result column of a C-Specs a long operand, such as **ControlName.Property**. This could easily exceed the 10 or 14 characters assigned to these columns.

To solve this problem, the Fixed Format editor of AVR was given a capability similar to that found in Word Processor and Spread Sheet applications: fixed columns of variable size or stretchable columns. As you probably have already noticed, there are some 'tab stop' symbols following some of the column headings of the editor. You can slide these tab stops to enlarge or reduce these columns.



If a particular line in a stretchable column has an entry too long to be displayed in the width of the column, the contents simply scroll within the cell formed by the line and column. When the cursor is not positioned on that cell, the editor displays a small red right-arrow (→) to show that the contents of that column are larger than what can be displayed.

- ✓ Run the program this time by selecting **Run** from the **Debug** menu, or by pressing **F5**. Your screen should look similar to the following:



The 'Value' Property

Every control has one property which is considered its '*value*' or default property (not to be confused with the property **Value**). Whenever you refer to a control within your code and do not specify a specific property, its value property is used. The value property for a Label is its **Caption**, and for an IOField is its **Value**.

Each control **type** determines its **value** property. If the control has a **Value** property, it is usually also its value property, but it may not. (The Command Button has a property named **Value**, yet its value property is actually **Caption**).

The last 3 lines of code we just wrote could have been:

```
Z-ADD      Total      ioTotal
Z-ADD      Tendered  ioTendered
Z-ADD      Change    ioChange
```

 **Go ahead and change your code to match the above 3 lines.**

The Project Window

An AVR program can be made up of several files that are linked together to make the program run. A project file can contain any number and combination of the following:

- One File for each form (Form1.VRF)
- One File for an MDI Form (MDIForm1.VRF)
- One File for each program (Untitld1.VRP)
- One File for each message text file (Msgs1.VMT)
- One File for each user class (Class1.VRC)
- One File for each user control (Controll1.VRX)


The project file that maintains the list of all the supporting files in a programming project is called the **Visual RPG Project (.VRM)** file.

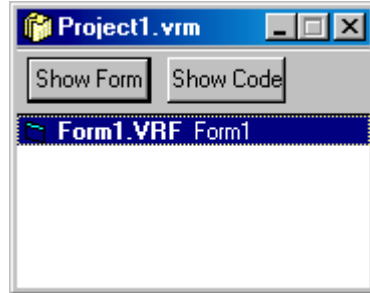
A default project (PROJECT1.VRM) is automatically created every time you start AVR with **FORM1.VRF** displayed in the project window.

You can access the files that make up the project by selecting the **Show Form** and **Show Code** buttons.

You can add, remove and save individual files to a project by using options on the File menu. As changes are made in a project, they are reflected in the project window.

So far in the example, you have only been dealing with one file, that containing the form.

 Display the project window (if not already displayed) by selecting **Project** from the **View** menu. The following window will display.

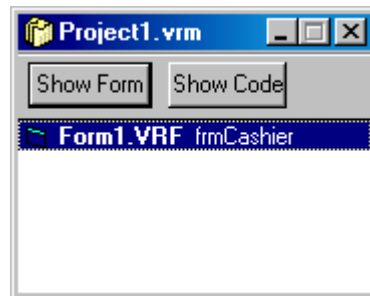


The first name in the project window (**Form1.VRF**) is the name of the file and the second **Form1** is the **Name** property of the form.

Let's Change the Name Property

1. Let's change the **Name** of the form in the property window. Open the property window by either clicking on the **Show Form** button, selecting **Property** from the **View** menu, or by pressing **F4**.
2. Double-click the **Name** property, enter **frmCashier**, and press **Enter**.

After you have pressed Enter, the project window will display, as shown below.



Saving Your Project

Now that you have completed the second step of the application, you should save it to disk.

The saving of a project is done in two steps. First, AVR saves your form's layout and code in one file, and then it saves the project components shown in the project window in another file.



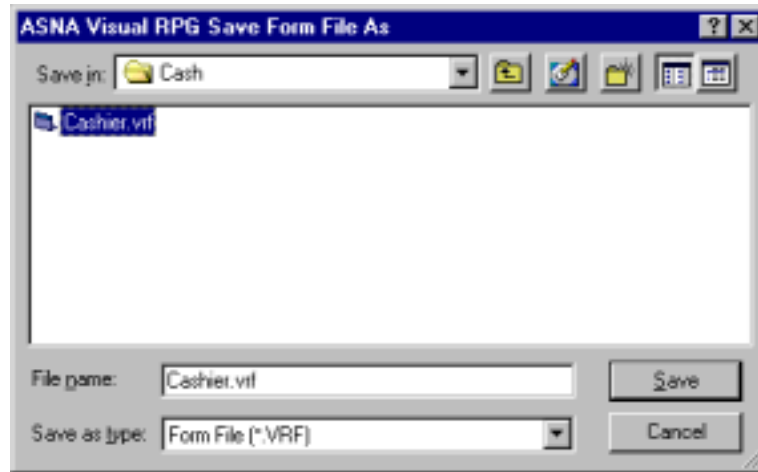
*You can use these component files individually in other programming projects by selecting **Add Files To Project** from the **File** menu.*

Let's Save the Project




Save Project

1. Select **Save Project As** from the **File** menu or click the **Save Project** button on the toolbar.
2. Click **Yes** when prompted to save changes to Form1.VRF. The Save Form File As dialog box appears, as follows:



The folder that displays is the folder in which a project was last saved, and may be different than the one shown above. Regardless of the folder shown, the following steps will still apply.












3. Click the **Up One Level** button  if needed to get to the **LearnAVR40** directory.
4. Double-click the **Cash** folder. You'll save your project in the practice folder **Cash** that the installation program created on your hard disk. (You can create and specify a different folder if you'd like.)
5. Type **Cashier** in the File Name text box, and press Enter. You do not need to enter an extension, the frmCashier form is saved as Cashier.vrf.
6. Click **Yes** when prompted to save changes to project1.vrm.
7. Change project1.vrm to **Cashier** and press Enter.

The Cashier project is saved under the name Cashier.vrm.



*To load this project again later, select **Open Project** from the **File** menu and click **Cashier** in the Open Project dialog box. You can also load a recently-used project by clicking the project name at the bottom of the ASNA Visual RPG **File** menu.*

Step 2 Summary

| To | Do This | Button/Keys |
|--|---|---|
| Display the control palette | <ul style="list-style-type: none"> Select Control Palette from the View menu. Press Ctrl+T. | Ctrl+T |
| Change a property value | <ul style="list-style-type: none"> For properties that require entering numbers or text, just type the information in the Settings Box and press Enter. For properties that have enumerated or Boolean values, click the down arrow to the right of the Settings Box and select the option you want from the ComboBox. For properties with the more button, click to display a dialog box appropriate to the property being set. |    |
| To get information on a property | Select the desired property in the property window and press F1 . |  |
| To get information on a control | Select the desired control in the control palette and press F1 . |  |
| Create a label | Select the Label control in the control palette and “draw” to the desired size on the form. |  |
| Create an IOField | Select the IOField control in the control palette and “draw” to the desired size on the form. |  |
| Display the editor window | Select Code from the V iew menu, the Show Code button in the project window, or F7 . |  |
| Display the C-spec columns | Enter a ‘C’ in the first column of the editor window. | |
| Display a list of supported spec types | Within the editor window, press F2 in the first column. |  |
| Move from column to column in the editor | Press the Tab key. |  |
| To refer to a property within code | Type in the Name of the control, followed by a period and the property name. i.e. <code>lblTotal.Caption</code> | |
| Save your project | <ul style="list-style-type: none"> Select Save Project As from the File menu. Click the Save Project button on the toolbar. |  |

More Information

To find more information on the topics introduced in Step 2, refer to the following help file topics.

Help File Topics:

- Setting control properties
- Adding controls to form
- Label control
- Name property
- Changing properties
- Fixed Format Editor
- Editor Tab, tools options
- Spec Types
- Stretchable columns
- Project window
- Saving your program



Responding to Events

What you will learn in Step 3:

- The difference between event-driven and traditional programming.
- How to write code to respond to a button being clicked.
- How to add comments to your code.
- How to select controls in the Editor Control Box.
- How to select events in the Editor Event Box.
- The Visual RPG program structure.
- IOField properties to enhance data entry and formatting.

Approximate Time to Complete Step 3:

45 minutes.

What the Application will look like after completing Step 3:

| | | |
|----------|-------------------------------------|-------------------------------------|
| Tendered | <input type="text" value="100.00"/> | |
| Total | <input type="text" value="25.00"/> | <input type="button" value="Pay"/> |
| Change | <input type="text" value="75.00"/> | <input type="button" value="Done"/> |

One of the main advantages the *Graphical User Interface* brought into computing is the possibility of creating applications that respond to the actions of the user. This enabled the user to *direct* the execution of the application, instead of having to follow a pre-defined set of steps encapsulated in the program.

Event Driven vs. Traditional Programming

In *traditional* programming, the code executed in a particular run of a program depends only on the inputs to it, starting with the first line of code and following a pathway through the program, calling subroutines as needed.

Event-Driven programming extends this model of execution to account for the actions the user takes in response to the program, and to handle the communication between the program itself and the system. These actions are called *events*, and when they occur, trigger the execution of an *event subroutine*.

In AVR, code is executed in response to an event. Each form and control has a predefined set of user and system events. If one of these events occurs, AVR invokes the code in the associated event subroutine. An event subroutine is the code that you write to respond to an event.

This is what happens in a typical event driven application:

1. The project starts and automatically loads and displays the Main form.
2. A form or control receives an event. The event can be caused by the user, the system, or indirectly by your code.
3. If there is an event subroutine corresponding to that event, it executes.
4. The application waits for the next event.

Examples of events associated with a form are:

- The mouse moves across the form. (MouseMove)
- The mouse gets clicked over the form. (Click)
- The form gets unloaded from the screen. (Unload)

Examples of events associated with an IOField are:

- A key got pressed. (KeyPress)
- The IOField gets the focus. (GotFocus)
- The field gets double-clicked. (DoubleClick)

AVR Program Structure

In AVR, as in traditional RPG, a program is predefined to follow this order:

File Specs:

- F-Spec - External File
- K(F)-Spec - Continuation Line for F-Spec

Data Specs:

- S(I)-Spec - Data Structure
- B(I)-Spec - Data Structure SubField
- N-Spec - Named Constant
- E-Spec - Extension for array definition

Calculations Specs:

- C-Spec - Calculation

Blank specs can appear anywhere in the program. Blank specs are used for comments and to write Caviar Free Format commands.

Notice that the **I** and **O** specs are not supported. All file access, including printing, is done via an **externally described file**, as you will see in Step4.

Within the C-Specs, any code preceding the subroutine portion of a program is considered the **Main C-Spec** lines. The code in these lines is executed the first time the form gets loaded into memory and before the program drops into its Wait-For-An-Event loop. Typically in the main C-Specs, files get opened, run-time-only properties are initialized, and other bookkeeping tasks are performed.

Subroutines follow the Main C-Specs. Subroutines are classified in two types: **Event Handling subroutines** and **Procedural subroutines**.

- **Event handling subroutines** are those called by the AVR run-time in response to an event caused by the user, the system, or indirectly by your code. You can also execute an event handling subroutine directly by providing in the **EXSR** the name of the control in Factor 2 and the name of the event in Factor 1.
- **Procedural subroutines** are the type of subroutines you have traditionally programmed in RPG.

Event Subroutines in the Editor



The Editor contains in its Header Bar two drop-down ListBoxes: **The Control Box** located at the top-left of the window, and the **Event Procedure Box** located at the top-right of the window.

The **Control Box** displays the current form and all the controls on the current form. Click on the arrow to the right of the ListBox to display a list of all controls associated with that form.

The **Event Procedure Box** lists the event procedures for the selected control. Click on the arrow to the right of the ListBox to display all the events available for the selected control.

If you select an event from the ListBox, AVR will add an event subroutine to your code. If an event subroutine already exists in your program for that control and event, the editor will position the cursor on that subroutine. The event subroutine generated by AVR contains the names of the control and event in a **BEGSR** line and a corresponding **ENDSR** line.

The syntax for **BEGSR** in AVR has been extended to accept an optional Factor 2. AVR determines if an event subroutine exists for a particular event, if there is a subroutine in the program with the control name in Factor 1, and the event name in Factor 2.

You only need to write event-handling subroutines for those events that you need. Whenever an event occurs and there is no RPG event handler for it, the event is still handled with a default behavior by Windows and the AVR run-time. For instance, double clicking on an IOField which doesn't handle the double-click event, selects the contents of the IOField.

Command Buttons

Now we will add a `CommandButton` to allow the user to compute the change to be given back to a customer based on the `Total` and the amount `Tendered`.

Let's Add a Command Button



CommandButton

1. Select the **CommandButton** in the control palette and place on the form.
2. Set the **Caption** property to `Pay`, **Name** to `btnPay` and **Default** property to **True**.

The form should look like the following:

The screenshot shows a window titled "The Supermarket Cashier". Inside the window, there are three text labels on the left: "Tendered", "Total", and "Change". To the right of "Tendered" is a text input field. To the right of "Total" is another text input field. To the right of "Change" is a third text input field. To the right of the "Total" input field is a rectangular button with the text "Pay" on it.

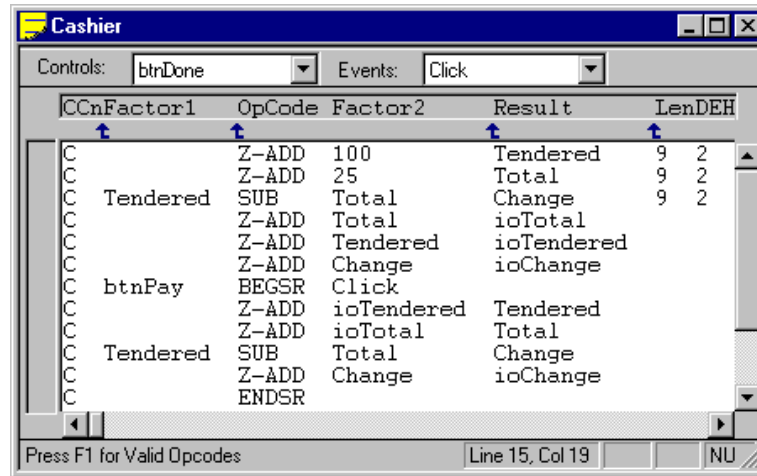
3. Open the Editor and type the following subroutine as it appears **after** the last line of code in the editor window.

The screenshot shows the "Cashier" editor window. At the top, "Controls:" is set to "btnPay" and "Events:" is set to "Click". Below this is a table with columns: CCFactor1, OpCode, Factor2, Result, and Le. The table contains the following code:

| CCFactor1 | OpCode | Factor2 | Result | Le |
|------------|--------|------------|----------|----|
| C btnPay | BEGSR | Click | | |
| C | Z-ADD | ioTendered | Tendered | |
| C | Z-Add | ioTotal | Total | |
| C Tendered | SUB | Total | Change | |
| C | Z-Add | Change | ioChange | |
| C | ENDSR | | | |

At the bottom of the editor, the status bar shows "Result" and "Line 11, Col 41".

Your program should look like the following:



4. **Run** your program.
5. After entering new values in the IOFields, click on the new **Pay** button. AVR will invoke the event-handling subroutine producing a new value for the **ioChange** field.

The Click Event

As you have seen, using the mouse to click on a button is one way of generating a **Click** event on a **CommandButton**. Another way is by giving the 'focus' to the button (usually by Tabbing to it) and pressing either the space bar or Enter key.

- If the command button has its **Default** property set to **True**, then pressing Enter anywhere on the form will also cause the button to get a click event.
- If the command button has its **Cancel** property set to True, then pressing the **Esc** key on the form will issue a click event on the button.

Finally, there is one more way of selecting the buttons of your program. Windows treats the ampersand character (&) as a special character when it finds it on a **Caption** property. When Windows sees the ampersand, it underlines the very **next** character of the caption. When the underlined character is pressed while the **Alt** key is being held down, it generates a click event for the **CommandButton**.

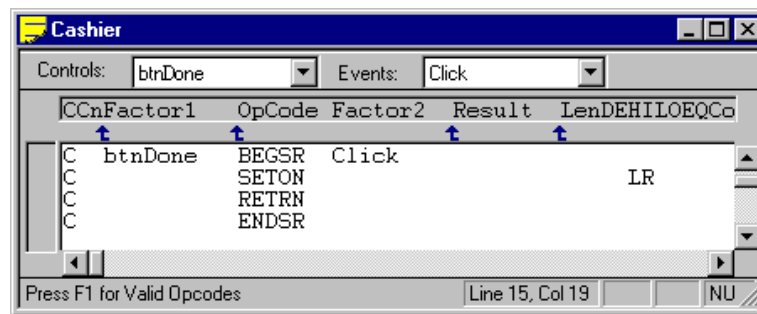
- ✓ Change the Caption of btnPay to **&Pay**.
You will notice how the **P** has been underlined.
- ✓ **Run** the program and exercise the different ways of generating clicks for the Pay button.

✓ Let's Add Another CommandButton



CommandButton

1. Add another **CommandButton** to allow an exit in a more controlled manner.
2. **Name** the button **btnDone**, set its **Caption** to **&Done** and its **Cancel** property to **True**, so that pressing the **Esc** key will generate a click on it.
3. Add the following event-handling subroutine for **btnDone**.



Returning from an event subroutine with the indicator **LR** set to **ON** will cause the program to terminate.

Adding Comments

Comments can be added to your program in AVR by using what is called a **Blank-Spec**, which is a line with a blank in the first column. Comments in AVR start with a `/*` and terminate with a `*/`. You can also start a comment with a `//`, in which case the comment runs until the end of the line.

You can see examples of these comments in the example below. Notice how it is possible to have a blank line by using a Blank-Spec.

```

Cashier
Controls: btnDone Events: Click
|CCFactor1| OpCode| Factor2| Result| LenDEHILOEQ|
|-----|-----|-----|-----|-----|
C      | Z-ADD| 100    | Tendered| 9 2|
C      | Z-ADD| 25     | Total   | 9 2|
C Tendered| SUB  | Total  | Change  | 9 2|
C      | Z-ADD| Total  | ioTotal |   |
C      | Z-ADD| Tendered| ioTendered|   |
C      | Z-ADD| Change | ioChange|   |
/* Compute the change owed to the customer */
C btnPay BEGSR Click
C      | Z-ADD| ioTendered| Tendered|   |
C      | Z-ADD| ioTotal  | Total   |   |
C Tendered| SUB  | Total  | Change  |   |
C      | Z-ADD| Change  | ioChange|   |
C      | ENDSR|         |         |   |
// Close down application
C btnDone BEGSR Click
C      | SETON|         |         | LR|
C      | RETRN|         |         |   |
C      | ENDSR|         |         |   |
Press F1 for Valid Opcodes Line 2, Col 14 NUM

```

Formatting Data using IOFields

Most of the IOField control properties govern the format of the **Value** of the control when it gets output. They also provide information to the control so that some validation can be done when the user inputs a new value.

The Following is a List of Properties to Consider:

For controls containing numeric values:

| | |
|----------------------|--|
| NumericLength | Defines the number of numeric digits of a variable. When NumericLength is set to a value greater than 0 and used in conjunction with the Decimals and EditCode Property, you can only enter numeric values, not to exceed the value set in this property. |
| Decimals | Defines the number of digits within the NumericLength that are to the right of the decimal point. When NumericLength and EditCode Property are set to values greater than 0, the user is allowed entry only of values not to exceed the number of decimals set in this property. |
| EditCode | Allows you to punctuate numeric fields, including \$ signs, commas, periods, minus sign, and floating minus according to the standard RPG edit code rules. Use the EditCode property to not only format the display of the Value property, but also to provide user input editing in conjunction with the NumericLength and Decimals properties. |

EditWord The EditWord property allows you to incorporate literals such as blanks, symbols and character text to edit numeric data according to the standard RPG rules for edit words.

For Controls Containing Character Values:

MaxLength Use the MaxLength property to limit the number of characters a user can enter into an IOField. The MaxLength Property overrides any other property relative to the number of characters that can be entered or displayed in an IOField. Even if an EditCode or EditWord evaluates the text to be more than MaxLength characters, or the NumericLength is more than MaxLength, only MaxLength will be displayed and will be truncated to the right.

PasswordChar Determines whether text typed by the user is displayed in the IOField, or if placeholder characters, such as an asterisk (*) are displayed.

UpperCaseOnly Forces the entry of uppercase characters.

For Any Type of Data:

Alignment Controls whether text appears in the left, center or right of the display area of the control. If the MultiLine property is False, the Alignment property is ignored for edited values using the EditWord or EditCode properties.

MultiLine Determines whether an IOField can accept and display multiple lines of text.

FieldAdvance Allows the "+", "-", and "Enter" keys on the numeric pad to act as a Tab key to jump to the next field in the Tab sequence. Tab sequence is an ordering of controls that indicates which control gets the focus when the Tab key is pressed.

FieldAutoAdvance When set to True, control will automatically advance to the next IOField control in the tab sequence when the number of characters entered in the IOField at run-time equals the maximum length set for that control with the MaxLength property.

OutOnly Determines whether the user can input text into the IOField.

OverWrite Set to True if you wish to have existing text overwritten or replaced with new text entered by the user.

✔ Let's Set the Following Properties:

1. Set the following properties as shown:


| ioTendered | <u>Property</u> | <u>Value</u> |
|-------------------|------------------------|---------------------|
| | Alignment | 1 |
| | Decimals | 2 |
| | EditCode | 5 |
| | FieldAdvance | True |
| | MultiLine | True |
| | NumericLength | 7 |

| ioTotal | <u>Property</u> | <u>Value</u> |
|----------------|------------------------|---------------------|
| | Alignment | 1 |
| | Decimals | 2 |
| | EditCode | 5 |
| | FieldAdvance | True |
| | MultiLine | True |
| | NumericLength | 7 |

| ioChange | <u>Property</u> | <u>Value</u> |
|-----------------|------------------------|---------------------|
| | Alignment | 1 |
| | Decimals | 2 |
| | EditCode | 5 |
| | MultiLine | True |
| | NumericLength | 7 |
| | OutOnly | True |

2. After you have made the changes shown above, **Run** your program and practice entering different values for **Tendered** and **Total** and compute the Change.
3. When you are done, save your program by selecting **Save Project** from the **File** menu. This time, AVR will not do any prompting, saving the form (.vrf) and project (.vrm) files in the same directory and names as specified at the end of Step 2.

Step 3 Summary

| To | Do This | Button/Keys |
|--|---|---|
| Create a Command Button | Select the Command Button control in the control palette and “draw” to the desired size on the form. |  |
| Add a comment in the code editor | Line must be a blank spec line (i.e., column 1 must be blank). Enclose the comment with a /* and */ or start with //. | |
| Align text to the left, center or right of an IOField | Set the Alignment property. | |
| Define the number of digits to the right of a decimal point | Set the Decimals property. | |
| Format and punctuate numeric fields | Set the EditCode property. | |
| Specify keys to act as a Tab key and jump to the next field | Set the FieldAdvance property. | |
| Specify whether an IOField can accept and display multiple lines of text | Set the MultiLine property. | |
| Format and punctuate numeric fields | Set the NumericLength property. | |
| Specify whether a user can input text into an IOField | Set the OutOnly property. | |

More Information

To find more information on the topics introduced in Step 3, refer to the following help file topics.

Help File Topics:

- Event driven programming
- Event subroutines
- Comments, adding
- Control box in editor
- Event procedure box, editor
- Properties, listing of
- Alignment property
- Decimals property
- EditCode property
- FieldAdvance property
- MultiLine property
- NumericLength property
- OutOnly property



Database Access

What you will learn in Step 4:

- How to manipulate multiple controls simultaneously.
- How to change properties common to multiple controls.
- How to access records in a file using Acceler8DB Database Manager.
- How to browse records in a file using Acceler8DB File Viewer.
- How to code an F-Spec and special continuation lines.
- How to code a message box.

Approximate Time to Complete Step 4:

1 hour.

What the Application will look like after completing Step 4:

At the end of Step 4, the application will look like the following:

The screenshot shows a window titled "The Supermarket Cashier" with a close button (X) in the top right corner. The main content area displays "Welcome to the SUPERMARKET". Below this, there are several input fields and buttons:

| | | |
|----------|-----------------------------------|---|
| Quantity | <input type="text" value="1.00"/> | |
| Item | <input type="text" value="2343"/> | |
| Price | <input type="text" value="1.50"/> | |
| Amount | <input type="text" value="1.50"/> | |
| Tendered | <input type="text" value=".00"/> | <input type="button" value="Buy Item"/> |
| Total | <input type="text" value="4.50"/> | <input type="button" value="Pay"/> |
| Change | <input type="text" value=".00"/> | <input type="button" value="New Client"/> |

Selecting and Manipulating Controls

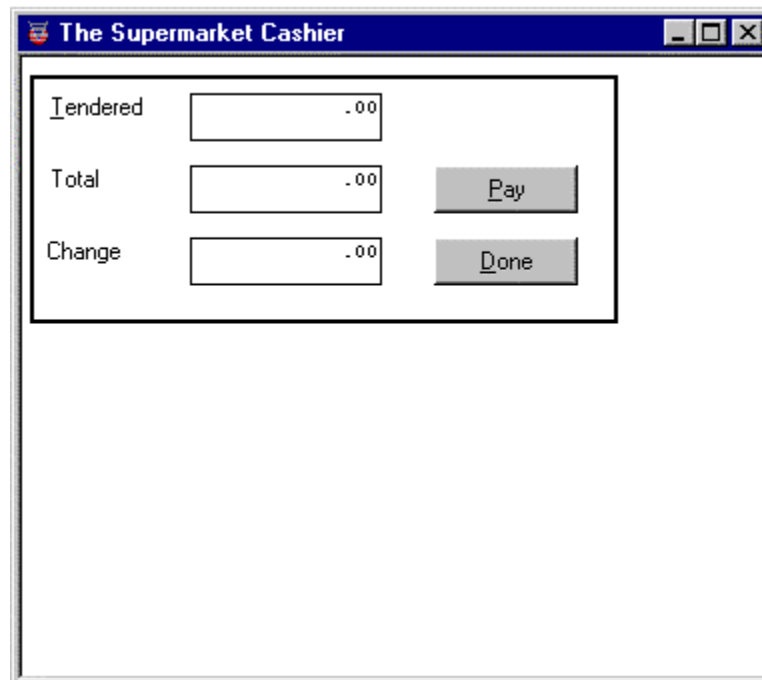
Once a control is placed on a form, it can be copied, moved, deleted or resized.

- To select a control - simply click on it. The control will be surrounded by 8 black squares. With the control selected, you can move it by dragging it over the form.
- To copy a control - select the control to copy, then select **C**opy from the **E**dit menu, or use the **Ctrl+C** keys.
- To delete a control - select the control to delete, then press the **Del** key.
- To resize a control - drag one of the black squares in the direction you want to grow or shrink the control.

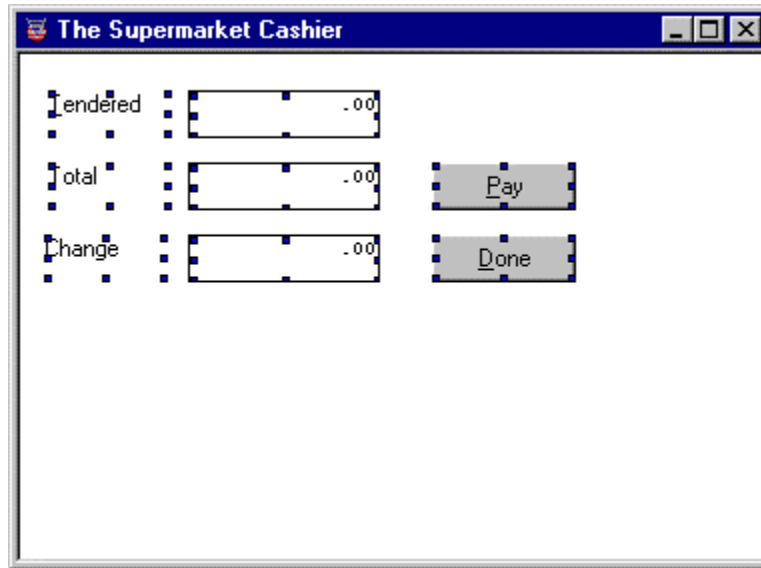
Selecting Multiple Controls

You can also copy, move, delete or resize a group of controls. There are two ways of selecting multiple controls.

1. Click on one of the controls you want to select on the form. With the control selected, press and hold down the **Ctrl** key, then click **each** additional control you want to select.
2. Click the arrow on the control palette, place the cursor on an area of the form where there is no control and **drag a rectangle around** or at least touch **all the controls** you want to select, as shown below.

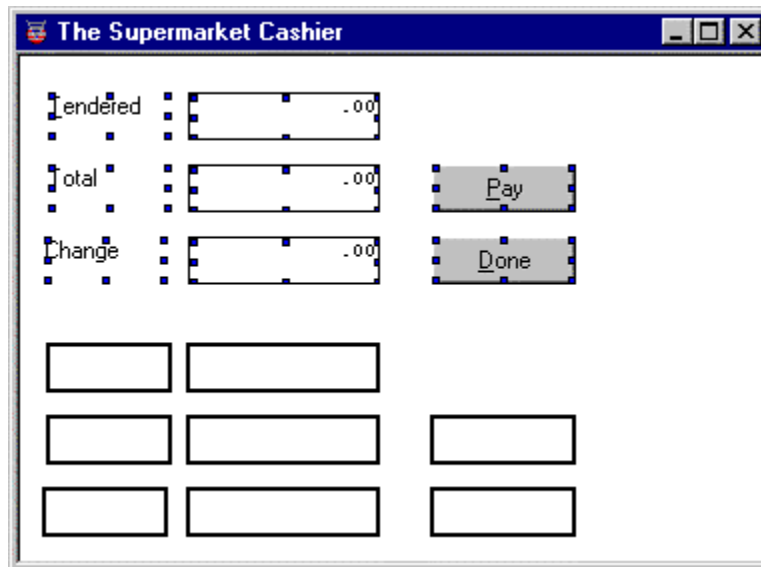


- When you release the mouse button, you will notice that each control will have 8 blue squares surrounding the control. To cancel the selection of a control, select a control again, with the Ctrl key pressed.




- To move the selected controls, place the arrow over one of the selected controls and drag it to its new position.

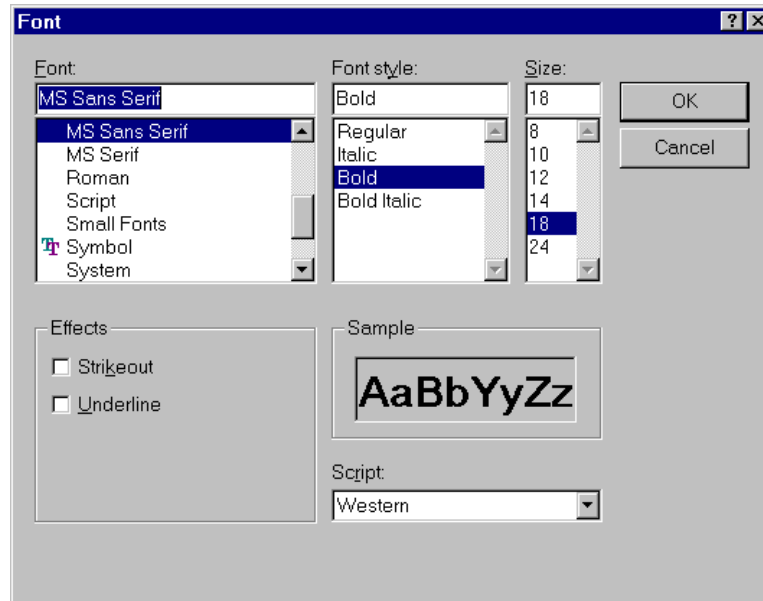
A gray box representing each control will be moving along with the mouse until the mouse button is released, then the controls will be placed in their new position, as shown below.



✔ Let's Continue with our Application

1. **Move** all the controls of your program to the bottom of the form, as shown on the previous page.
2. Add a Label named **lblWelcome** at the top of the form. Set its **Caption** to **Welcome to the Supermarket**.
3. To make this label stand out, set its font size to **18** and the font style to **Bold**.
 - To do this, click on the **Font** property in the property window.
 - Click the  button next to the font property.

The following dialog will appear.



After selecting the new values, press the **OK** button. (You may need to resize the label on the form to allow it to display the whole phrase).

4. Add 4 more Labels:

| <u>Name</u> | <u>Caption</u> |
|-------------|----------------|
| lblQuantity | &Quantity |
| lblItem | &Item |
| lblPrice | Price |
| lblAmount | Amount |

(Notice the ampersand on the first 2 labels).

5. Add 4 more IOFields: **ioQuantity**, **ioItem**, **ioPrice** and **ioAmount**.

We want to give to several properties the same values for fields **ioQuantity**, **ioPrice** and **ioAmount**. To assign the same value for a common property on multiple controls, do a multiple selection as previously described. The property window will list all of the properties that the selected controls have in common, and set the new value.

- Set the common values for **ioQuantity**, **ioPrice** and **ioAmount**, as shown below.

| <u>Property</u> | <u>Value</u> |
|-----------------|----------------|
| Alignment | 1 |
| Decimals | 2 |
| EditCode | 5 |
| MultiLine | True |
| NumericLength | 7 |
| Text | (blank it out) |

- For **ioAmount**, set **OutOnly** property to True.
- For **ioQuantity** and **ioItem**, set **FieldAdvance** to True.
- Finally, set the other properties of **ioItem** as follows:

| <u>Property</u> | <u>Value</u> |
|-----------------|----------------|
| Alignment | 1 |
| Decimals | 0 |
| EditCode | 4 |
| MultiLine | True |
| NumericLength | 12 |
| Text | (blank it out) |

- Delete the button **btnDone** and it's **Click** subroutine.

To delete lines of code in the editor, select them by pressing the mouse button down on the first line, then dragging the mouse to the last line to be deleted. All of the selected lines will be highlighted, then press the Delete key.

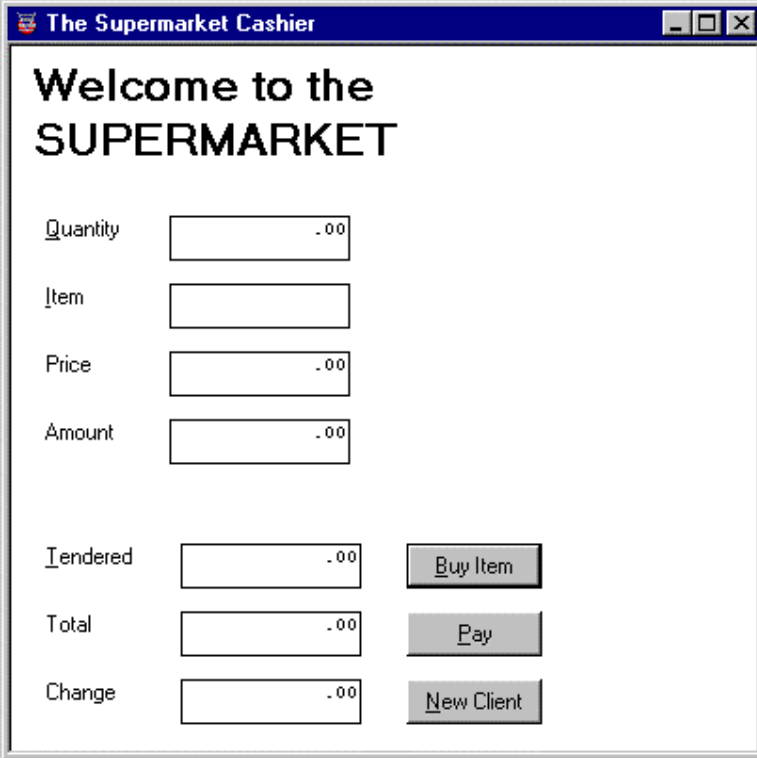
- Add a couple of new buttons, **btnBuy** and **btnNew** and set the properties as shown below.

| btnBuy | <u>Property</u> | <u>Value</u> |
|---------------|-----------------|--------------|
| | Caption | &Buy Item |
| Default | True | |

| btnNew | <u>Property</u> | <u>Value</u> |
|---------------|-----------------|--------------|
| | Caption | &New Client |

12. Since originally the Pay button was set as the default button, change its **Default** property to false, as the Buy Item button is now the default button.

When you are done, your form should look like the following:



The screenshot shows a window titled "The Supermarket Cashier" with a blue title bar. The main content area has a white background with the text "Welcome to the SUPERMARKET" in bold. Below this, there are seven input fields, each with a label and a ".00" value on the right. The labels are: Quantity, Item, Price, Amount, Tendered, Total, and Change. To the right of the "Tendered", "Total", and "Change" fields are three buttons: "Buy Item", "Pay", and "New Client".

Program Behavior

We are going to let the cashier enter a quantity for the number of items to be bought and the Item's UPC (Universal Product Code). We will have a file with the information about each item that the supermarket carries. When the cashier clicks on the **Buy** button, the program will find the entered UPC, compute the amount to be charged for this item, and accumulate it in the total.

The cashier will enter the amount tendered and click the **Pay** button to compute the change to be given back to the client. Finally when the **New** button is clicked, the program will clear the total to get ready for a new customer.

Before we write the program's code, let's look at the **Item file** layout and the database system we are going to use.

The AVR run-time relies on database operations from a couple of products from ASNA called **Acceler8DB** and **DataGate/400**.

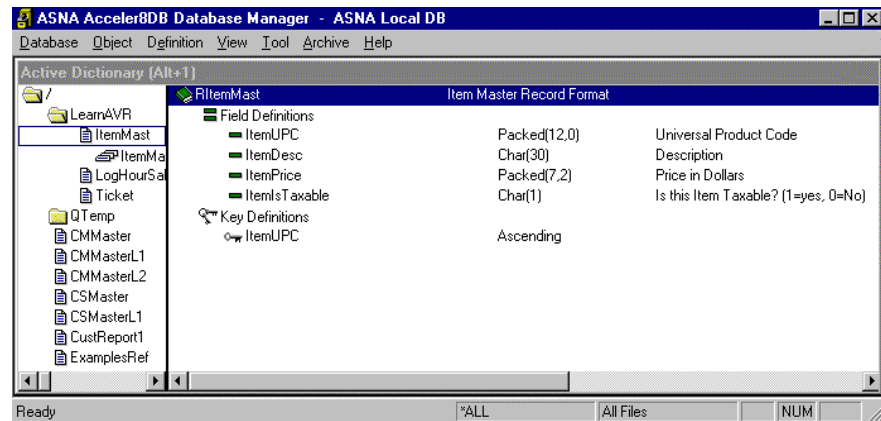
- **Acceler8DB (ADB)** is a Database Management System very similar in capability to that found on IBM's DB2/400. It provides the same type of database operations and files for Windows-based PCs. It can run stand-alone on your PC or as a Server on a Windows NT PC.
- **DataGate/400** is a product that runs on an AS/400 providing access to its DB2/400 files. Acceler8DB makes access to the PC database or the AS/400 database transparent to your program.

File Layout

As part of the installation of AVR, an Acceler8DB database called **ASNA Local DB** was installed on your PC. This database contains the necessary files for our application.

✓ Let's View a Database in Acceler8DB

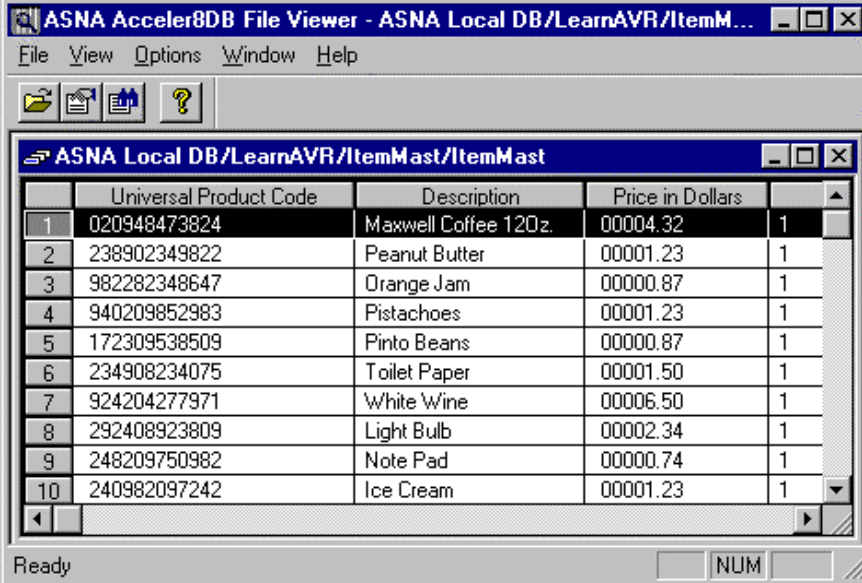
1. To view an Acceler8DB database, either select **Acceler8DB Database Manager** from the **Tools** menu in AVR or press the **Start** button - **Programs - ASNA Product Suite - Acceler8DB 5.0 - Acceler8DB Database Manager**.
2. Select **Open** from the **Database** menu.
3. From the database name list, select **ASNA Local DB** and click **OK**. The contents of the database will be displayed.
4. Double-click on the **LearnAVR** folder, select the **ItemMast** file on the left panel and **double-click** the record format on the right panel to display the record layout. Your screen will look like the following:



Notice there are 4 fields in the format: **ItemUPC**, **ItemDesc**, **ItemPrice** and **ItemIsTaxable**. The file is keyed on ItemUPC.

5. To browse the data in a physical file, use Acceler8DB File Viewer. **Double-click** the ItemMast file member (ItemMast). You can also select **Browse Data** from the **Tool** menu when the file is selected on the left panel.

- Acceler8DB File Viewer will automatically display the records of ItemMast in arrival order. Use the scroll bars in the Window to browse. When you are finished browsing, select **Exit** from the **File** menu.



The screenshot shows a window titled "ASNA Acceler8DB File Viewer - ASNA Local DB/LearnAVR/ItemM...". The window contains a table with the following data:

| | Universal Product Code | Description | Price in Dollars | |
|----|------------------------|----------------------|------------------|---|
| 1 | 020948473824 | Maxwell Coffee 120z. | 00004.32 | 1 |
| 2 | 238902349822 | Peanut Butter | 00001.23 | 1 |
| 3 | 982282348647 | Orange Jam | 00000.87 | 1 |
| 4 | 940209852983 | Pistachoes | 00001.23 | 1 |
| 5 | 172309538509 | Pinto Beans | 00000.87 | 1 |
| 6 | 234908234075 | Toilet Paper | 00001.50 | 1 |
| 7 | 924204277971 | White Wine | 00006.50 | 1 |
| 8 | 292408923809 | Light Bulb | 00002.34 | 1 |
| 9 | 248209750982 | Note Pad | 00000.74 | 1 |
| 10 | 240982097242 | Ice Cream | 00001.23 | 1 |

Special Continuation Lines

Now we can code the F-Spec to include the database file.

✓ Let's Code an F-Spec

- To insert a line **before** the first C-Spec in the code editor, place the cursor in Line 1, Column 1. Then, while holding down the **Ctrl** key, press Enter.
- Code the line as follows:

```
FItemMastIF E K DISK
```

This disk file will be used for Input only, is Full Procedural, Externally Described and Keyed. There is, however, one more piece of information we need to provide the compiler and that is - where does this file reside?

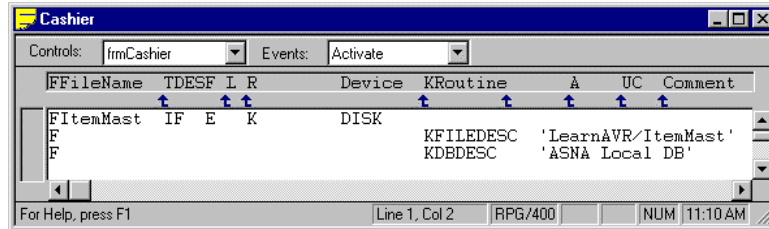
AVR has added a couple of continuation lines for this purpose:

- DBDESC** tells the compiler which database to use to find the description of the file.
- FILEDESC** specifies the name of the file to be used. This file name includes the Directory path, or library in the case of an AS/400 file. If the FILEDESC continuation line is missing, AVR uses the name used in the F-spec for the file name and *LIBL for the path.

In our program, we'll direct AVR to the **ASNA Local DB** database and we'll use the **LearnAVR/ItemMast** file.

To request a continuation line format in the editor, type a '**K**' on the first column of the continuation line. The editor will display an '**F**' with the proper format.

3. Enter the complete F-Spec code as shown below.



(Notice how the K-Spec has a different format than the F-Spec, even though both show an '**F**' in their first column).

Message Box

AVR has an op code called **MSGBOX**, which is used to pop-up a simple message box on the screen. **MSGBOX** displays the message in a dialog box, and waits for the user to click the OK button. The message to be displayed is specified in Factor 2.

The following line of code:

```
MSGBOX 'Invalid UPC' yields the following message box:
```



and `MSGBOX 'Not Enough Money Tendered'` yields



For simplicity in our testing, the program will find the Item with a UPC closest to the one entered by doing a SETLL and a READ like the following:

```
C      ItemUPC      SETLL  ItemMast
C                               READ  ItemMast
```

✔ Let's Complete the Program

1. Type in **all the code** as shown in the next figure.
2. **Save** the program, then **Run** it.



*If you have any problems compiling and running your code, open the project in the **LearnAVR40/Step4** directory and compare it with your code.*

```




Cashier *
Controls: frmCashier  Events: Activate
FFileName TDES F I R Device KRoutine A UC Comment
FItemMast IF E K DISK KFILEDESC 'LearnAVR/ItemMast'
F KDBDESC 'ASNA Local DB'
/* Handle the event Click for the button Buy */
/* ***** */
C btnBuy BEGSR Click
/* Validate the UPC */
C Z-ADD ioQuantity Quantity 7 2
MOVE ioItem ItemUPC P
ItemUPC IFEQ 0
MSGBOX 'Invalid UPC'
RETRN
ENDIF
/* Read Item details */
C ItemUPC SETLL ItemMast
READ ItemMast 91
ItemPrice MULT Quantity Amount 7 2
ADD Amount Total 9 2
MOVE ItemPrice ioPrice
MOVE ItemUPC ioItem
MOVE Amount ioAmount
MOVE Total ioTotal
/* Clear input fields */
C Z-ADD 1 ioQuantity
MOVE *BLANKS ioItem
ENDSR
/* Clear form and display different 'Special Of The Day' for new customer */
/* ***** */
C btnNew BEGSR Click
Z-ADD 0 Total
Z-ADD 1 ioQuantity
Z-ADD 0 ioPrice
Z-ADD 0 ioAmount
Z-ADD 0 ioTotal
Z-ADD 0 ioTendered
Z-ADD 0 ioChange
ENDSR
/* When the Pay button gets clicked, compute the change owed to the customer */
/* ***** */
C btnPay BEGSR Click
Z-ADD ioTendered Tendered 9 2
Tendered IFLT Total
MSGBOX 'Not Enough Money Tendered'
RETRN
ENDIF
Tendered SUB Total Change 9 2
MOVE Change ioChange
ENDSR

```

Externally Defined File Spec - Press F2 to select valid spec Line 1, Col 1 RPG/400 NUM 11:15 AM

Notice how the main C-Specs have been removed, and how the **btnPay-Click** subroutine has been changed.

Step 4 Summary

| To | Do This | Button/Keys |
|--|---|---|
| Select a control | Click on the control. The control will be surrounded by 8 black squares. | |
| Copy a control | Select C opy from the E dit menu. | |
| Delete a control | Press the Del key. |  |
| Resize a control | Drag one of the black squares in the direction you want to grow or shrink the control. | |
| Select multiple controls | <ul style="list-style-type: none"> Click on a control, and press and hold down the Ctrl key, then click each additional control you want to select. Click the arrow on the control palette, place the cursor on an area of the form where there is no control, and drag a rectangle over all the controls you want to select. |  |
| View a database | <ul style="list-style-type: none"> Select Acceler8DB Database Manager from the Tools menu in Visual RPG. Press the Start button - Programs - ASNA Product Suite - Acceler8DB 5.0 - Acceler8DB Database Manager. | |
| Browse data in a physical file | <ul style="list-style-type: none"> Click on a file in Acceler8DB and double-click a file member (). Click on a file member in Acceler8DB and select Browse Data from the Tool menu. | |
| Include a database file in a program | Code an F-Spec by entering an F in the first column in the code editor. | |
| Include a continuation line in the code editor | Enter a K in the first column of the continuation line. | |
| Display a message in a dialog box | Use the MSGBOX op code, i.e. MSGBOX 'Invalid UPC' | |

More Information

To find more information on the topics introduced in Step 4, refer to the following help file topics.

Help File Topics:

- Selecting multiple controls
- MSGBOX
- Continuation lines
- Spec formats, listing of



Methods

What you will learn in Step 5:

- What a method is.
- How to call methods in a control.
- How to disable a control.
- How to add a ListBox to a form.
- How to control the order followed by the Tab key.
- How to add a picture to a form.

Approximate Time to Complete Step 5:

1 and ½ hours.

What the Application will look like after completing Step 5:

The screenshot shows a window titled "The Supermarket Cashier" with a blue title bar. The main content area is divided into several sections:

- Welcome to the SUPERMARKET**: Large text on the left side.
- Wine Special**: A section on the right featuring a small image of a wine bottle and a list of items with their prices.

| Wine Special | | |
|--------------|----------------|-------|
| 1.00 | Sugar | .98 |
| 1.00 | Blue Berry Jam | .95 |
| 6.00 | Light Bulb | 14.04 |
| 2.50 | Pinto Beans | 2.17 |
- Input Fields**: On the left, there are four text boxes labeled "Quantity", "Item", "Price", and "Amount". The "Quantity" box contains "1.00", "Price" contains ".87", and "Amount" contains "2.17".
- Summary and Controls**: At the bottom left, there are three text boxes labeled "Tendered", "Total", and "Change". "Tendered" contains ".00", "Total" contains "18.14", and "Change" contains ".00". To the right of these are three buttons: "Buy Item", "Pay", and "New Client".

Disabling a Control

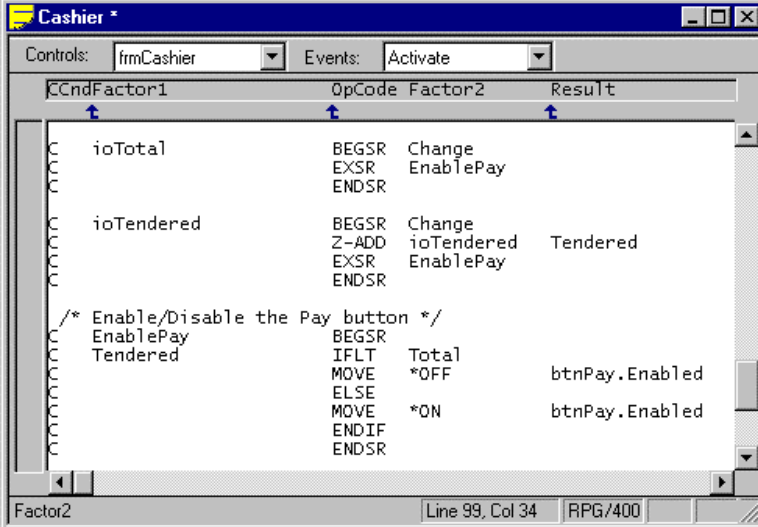
As we have seen, providing our users with a message box indicating what data entry error they have introduced is trivial. However, our application would be better if it did not allow the user to commit such an obvious transgression.

Since our program cannot really handle the click of the Pay button if the amount tendered is less than the total amount, the Pay button should be disabled until the user has entered the proper tendered value.

IOFields fire an event *every* time the user or the program changes its contents, so we will handle the **Change** event for the **ioTendered** and the **ioTotal** fields. In each handler, we will call a regular subroutine (EnablePay) where we will check to see if the amount entered is correct. If it is correct, we will enable the button, otherwise, we will disable it. Disabling a control is as simple as setting its **Enabled** property to False.

Let's Code Event Handlers

1. Let's code two event handlers for the **EnablePay** subroutine, as shown below.



```

CndFactor1      OpCode  Factor2      Result
-----
C  ioTotal      BEGSR      Change
C                                     EXSR      EnablePay
C                                     ENDSR
C
C  ioTendered   BEGSR      Change
C                                     Z-ADD    ioTendered  Tendered
C                                     EXSR      EnablePay
C                                     ENDSR
C
C /* Enable/Disable the Pay button */
C EnablePay    BEGSR
C Tendered     IFLT      Total
C                                     MOVE     *OFF      btnPay.Enabled
C                                     ELSE
C                                     MOVE     *ON      btnPay.Enabled
C                                     ENDIF
C                                     ENDSR

```

Factor2 Line 99, Col 34 RPG/400

2. In the event handler for the Click of btnPay, we do not need to validate the amount tendered any more. Once the cashier has chosen to Pay, we should disable the Buy and the Pay buttons and enable the New Client button. Let's code the **btnPay-Click** subroutine to look like the following:

```

Cashier *
Controls: frmCashier Events: Activate
CnCndInFactor1 OpCode Factor2 Result LenDEHILLOEQComment
/* *****
/* When the Pay button gets clicked, compute the change owed to the customer */
/* *****
C btnPay BEGSR Click
C Z-ADD ioTendered Tendered 9 2
C Tendered SUB Total Change 9 2
C MOVE Change ioChange
C MOVE *OFF btnBuy.Enabled
C MOVE *OFF btnPay.Enabled
C MOVE *ON btnNew.Enabled
C ENDSR
C ioTotal BEGSR Change
C EXSR EnablePay
C ENDSR
Factor1 Line 71, Col 11 NUM 03:11 PM

```

When the cashier clicks on the **New Client** button, besides clearing the IOFields, the handler should disable btnNew and enable the btnBuy. We should also disable btnPay and btnNew at design-time.

Methods

So far in previous chapters, you have learned that:

1. Controls have properties which determine the behavior of the control.
2. Controls generate events, which are the means by which a program knows about the user interaction with that control.
3. Controls also have **Methods** associated with them. Methods are subroutines that manipulate the properties of controls, and can be called from within the program to alter the control's appearance and behavior.

For example, the ComboBox and ListBox controls have a method called **AddItem**, which adds a new line of data to the list.

The following is a list of the most common methods and the controls on which they operate:

| | |
|-------------------|---|
| AddItem | Adds a new item to a ListBox or ComboBox, or adds a new row to the Subfile and Grid controls. |
| ClearObj | Clears all the data from the IOField, ListBox, ComboBox, Subfile, Grid, Command Button, Option Button and Scrollbar controls. |
| MoveObj | Moves any control, including the form, to new coordinates in the workstation window. |
| Refresh | Forces the immediate repaint of any control, including the form. |
| RemoveItem | Removes a specific item from the ListBox or ComboBox, or removes a specific row from the Subfile or Grid controls. |
| SetFocus | Sets the input focus on any control, including a form. |

✓ Let's Add a ListBox



ListBox

1. Add a **ListBox** to your form by selecting the ListBox button in the control palette. In the ListBox, we will keep the running receipt for the customer.
2. Set the ListBox's **Name** to **lstReceipt** (that is LstReceipt not FirstReceipt) and its **Font** property to **Courier New** size **7**.

(You can enter a number directly in the size field in the Font's dialog box even if it is not contained in the Size ListBox).

There are two types of fonts in Windows: **Monospaced** and **Proportional**.

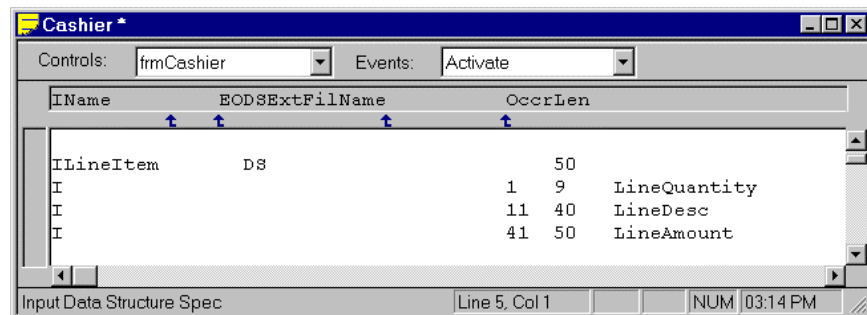
- **Monospaced** fonts are those which take the same amount of space per character regardless of what the character is, so a **W** is as 'wide' as an **i**. Courier, Courier New, and Lucida Console are examples of Monospaced fonts.
- In **proportional** fonts, wider characters take more space on the screen or page than the thinner ones. This sentence is printed in Times New Roman, which is proportional.

For each item bought by the client, we want to add a line to lstReceipt containing the Quantity, Description and Amount paid. All this information is already available in the subroutine handling the Click of btnBuy. We will set up a data structure with 3 fields and use the AddItem method to put it on the list. In AVR, you invoke a method for a control by using the **EVAL** statement as follows:

```
C                               EVAL  lstReceipt.addItem(LineItem)
```

✓ Let's Add a Data Structure

1. Let's add the data structure **LineItem**, as shown below:



Remember that you will have to enter an **'S'** in the first column of the editor to request a Data Structure specification and a **'B'** for a Data Structure Field specification.

As each item is added to the bottom of the list, it appears on the screen. Once the list displays enough items to cover its area on the screen, it will automatically add a vertical scroll bar to itself. Using this scroll bar, the cashier will be able to scan the list for products on the receipt.

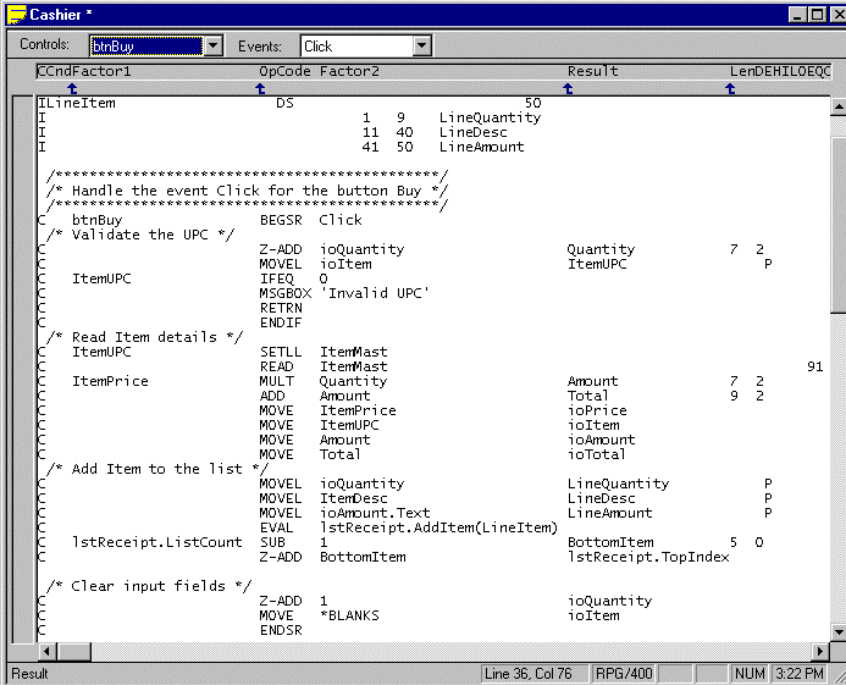
When adding items to the bottom of a `ListBox`, it is nice to always show the end of the list on the screen. There are two properties that help in achieving this. Both of these properties are available at run-time only.

- **ListCount** - tells how many items the list contains.
- **TopIndex** - determines the zero-based item in the `ListBox` that will appear in the top-most position (if there are enough items below it to fill the visible portion of the list).

Since `TopIndex` starts counting at 0, we will subtract one from the `ListCount` to determine the new value of `TopIndex`. With this we set `TopIndex` to be the last item in the list. The `ListBox` will take care of adjusting `TopIndex` so there are no empty lines in the `ListBox` window.

✓ Let's Complete our Code

1. Let's change our code to a new version of the **Click** handler for `btnBuy`, as shown below:



```

Cashier *
Controls: btnBuy Events: Click
CndFactor1 OpCode Factor2 Result LenDEHIOEQC
HLineItem DS 50
I 1 9 LineQuantity
I 11 40 LineDesc
I 41 50 LineAmount
/* Handle the event Click for the button Buy */
/* Validate the UPC */
C btnBuy BEGSR Click
/* Validate the UPC */
Z-ADD ioQuantity Quantity 7 2
MOVE ioItem ItemUPC P
ItemUPC IFEQ 0
MSGBOX 'Invalid UPC'
RETRN
ENDIF
/* Read Item details */
ItemUPC SETLL ItemMast
READ ItemMast 91
ItemPrice MULT Quantity
ADD Amount Total 7 2
MOVE ItemPrice ioPrice 9 2
MOVE ItemUPC ioItem
MOVE Amount ioAmount
MOVE Total ioTotal
/* Add Item to the list */
MOVE ioQuantity LineQuantity P
MOVE ItemDesc LineDesc P
MOVE ioAmount.Text LineAmount P
EVAL 1stReceipt.AddItem(LineItem)
1stReceipt.ListCount SUB 1 BottomItem 5 0
Z-ADD BottomItem 1stReceipt.TopIndex
/* Clear input fields */
Z-ADD 1 ioQuantity
MOVE *BLANKS ioItem
ENDSR
Result Line 36, Col 76 RPG/400 NUM 3:22 PM

```

There is one more thing to make the `ListBox` fully functional. When we are processing the click event on `btnNew`, we should clear the list.

2. To clear the list, invoke the **ClearObj** method, as shown below:

```

Cashier
Controls: frmCashier Events: Activate
CCondInFactor1  OpCode  Factor2          Result          LenDEHILOEQComment
-----
/* *****
/* Clear form and display different 'Special Of The Day' for new customer */
/* *****
C      btnNew      BEGSR  Click
C      EVAL      lstReceipt.ClearObj()
C      MOVE      *OFF          btnNew.Enabled
C      MOVE      *ON          btnBuy.Enabled
C      Z-ADD     0              Total
C      Z-ADD     1              ioQuantity
C      Z-ADD     0              ioPrice
C      Z-ADD     0              ioAmount
C      Z-ADD     0              ioTotal
Calculation Spec          Line 55, Col 1          NUM 11:00 AM

```

Do not forget to add the parenthesis () after ClearObj. The parenthesis tells the compiler that you want to call the method.

Setting the Tab Order

The **TabIndex** property determines the order in which AVR highlights IOFields, buttons and other controls at run-time. When a control is highlighted, it is said to have the **focus**. The focus is important for keyboard operations. When the user presses Enter, the control that has the focus is selected, or activated, in the program. The user can switch the focus to another control in the form by pressing the Tab key or by clicking the object.

The control with its TabIndex set to 0 receives the focus as soon as the program starts. If the user presses Tab, the control with a TabIndex of 1 gets the focus next, and so on. The first control that you create has a TabIndex of 0. The second control you create will have a TabIndex set to 1, and so on.

You can change the order in which controls receive the focus in the form by changing each control's TabIndex property. Whenever you change the TabIndex property of a control, AVR automatically renumbers the TabIndex of your other controls. It is impossible for two controls to have identical TabIndex values.

The TabIndex property controls which IOField will get the focus if the user types one of the mnemonic keys of labels. Mnemonic keys in labels are those characters that are underlined in a label due to an ampersand in the Caption. When the user holds down the **Alt** key while pressing one of these underlined characters, the focus is given to the first IOField with a TabIndex **higher** than the one on the label.

TabStop is another property that participates in the effect the Tab key has on your form. Only those controls which have TabStop set to **True** will receive focus when the user presses the Tab key. If a control has TabStop set to False, then it gets skipped when the user tabs through the form. The control can still get the focus by clicking it with the mouse.

If you have created several controls, you can set the TabIndex and TabStop properties for them quickly and easily by following these steps:

Let's Set TabIndex and TabStop Properties

Follow the steps below to set the TabIndex and TabStop properties shown on the following page.

1. Click the control you want to get the focus *Last*. That is, the object that will have the *highest* TabIndex property value.
2. Set the TabIndex property to **0**.
3. Click the control you want highlighted second-to-last.
4. Repeat steps 2 and 3 until you have set the TabIndex properties for all your controls to 0.

Set the TabIndex and TabStop Properties as follows:

| Control | TabIndex | TabStop |
|-------------|----------|---------|
| lblQuantity | 0 | N/A |
| ioQuantity | 1 | True |
| lblItem | 2 | N/A |
| ioItem | 3 | True |
| lblPrice | 4 | N/A |
| ioPrice | 5 | False |
| lblAmount | 6 | N/A |
| ioAmount | 7 | False |
| btnBuy | 8 | True |
| lblTendered | 9 | N/A |
| ioTendered | 10 | True |
| btnPay | 11 | True |
| btnNew | 12 | True |
| lblTotal | 13 | N/A |
| ioTotal | 14 | False |
| lblChange | 15 | N/A |
| ioChange | 16 | False |
| lstReceipt | 17 | False |
| lblWelcome | 18 | N/A |

If you followed these steps, the first control you clicked is lblWelcome and the last control you clicked, lblQuantity, will have a TabIndex of 0. The second-to-last control you clicked, ioQuantity, will have a TabIndex of 1, and so on.


Displaying Pictures

The Image control can display a picture from a bitmap, icon, or metafile. Use the **Picture** property to specify the graphic to be used. A dialog box will display for you to browse the directories to locate the desired picture. The Picture property is a design-time only property. However, AVR has a **LOADPIC** op code to set a new picture for the Image control.

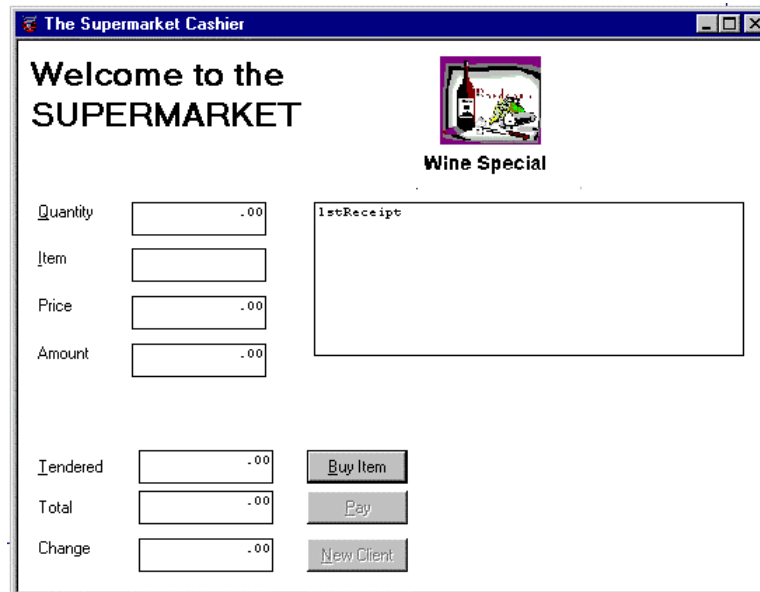
Let's Add a Picture



Image Control

1. Select the **Image** Control from the control palette and add it above IstReceipt. Set its **Name** to **imgSpecial**.
2. Using the  button of the **Picture** property, find the file **wine.bmp** in directory **FoodMap** within the **LearnAvr40** directory and double-click on it.
3. Notice that the size of the image control is probably not large enough to hold the entire graphic. In order to resize the graphic to fit within the control, regardless of the size, set the **Stretch** property to **True**.

Your screen should look like the following.



Showing a Picture on a Form

The **FoodMap** directory contains 5 bitmaps with food pictures. We will store the names of these files in an array called **ImageName** and use them to change the picture every time we get a new customer.

1. Code the subroutine **LoadFood** to initialize the array and the variable **ImgIdx** to be used as an index into the array, as shown below.

```

Cashier *
Controls: frmCashier Events: Activate
C CndInFactor1 OpCode Factor2 Result LenDEHILOBQComme
/* *****
/* Load in an array the names of the bitmaps containing the 'Special
/* *****
C LoadFood BEGSR
C MOVEL 'turkey.bmp' ImageName,1
C MOVEL 'icecream.bmp' ImageName,2
C MOVEL 'wine.bmp' ImageName,3
C MOVEL 'meat.bmp' ImageName,4
C MOVEL 'cheese.bmp' ImageName,5
C Z-ADD 0 ImgIdx 2 0
C ENDSR
Calculation Spec Line 101, Col 1 NUM 03:23 PM
    
```

2. The array is being declared in an E-Spec, as we will call the LoadFood subroutine from the main C-Spec. Let's add the following code:

```

Cashier
Controls: btnBuy Events: Click
E Name EPATLenMDSComment
I 41 50 LineAmount
E ImageName 5 20
C EXSR LoadFood
/* *****
/* Handle the event Click for the button Buy */
/* *****
C btnBuy BEGSR Click
Table or Array name Line 10, Col 25 NUM 04:27 PM
    
```

3. Let's add the following code to the end of the subroutine **btnNew** event **Click**:



*Note in the code below that the directory within the **CAT** operation is being used to concatenate the path of the directory with the individual bitmap file names stored in the array.*

Notice the entry for the FoodMap directory. It will be searched for in the current directory, or the directory in which AVR was installed. The default directory path is:

\Program Files\Asna\Learnavr40\FoodMap

```


Cashier
Controls: btnNew Events: Click
C CnFactor1 OpCode Factor2 Result LenDEHILO
C ADD 1 ImgIdx
C ImgIdx IFEQ 6
C Z-ADD 1 ImgIdx
C ENDF
C '..\FoodMap\' CAT ImageName,ImgIdx BitMapFile 64
C LOADPICBitMapFile imgSpecial
C ENDSR
Factor2 Line 50, Col 31 RPG/400 NUM 11:34 AM
    
```

LOADPIC tells the image **imgSpecial** to change its picture to the new **BitMapFile**.



*If you get an error saying 'Access is denied' when running the program, check that you have entered the correct path to the **FoodMap** directory and that you didn't forget the trailing slash.*

Step 5 Summary

| To | Do This | Button/Keys |
|--|---|---|
| Add a new item to a Listbox or ComboBox | Use the AddItem method. | |
| Add a new row to the Subfile or Grid control | Use the AddItem method. | |
| Clear all data from a control | Use the ClearObj method. | |
| Move any control to a new coordinate in the workstation window | Use the MoveObj method. | |
| Force the immediate repaint of any control, including the form | Use the Refresh method. | |
| Remove a specific item from the Listbox or ComboBox , or remove a specific row from the Subfile or Grid controls | Use the RemoveItem method. | |
| Set the Input Focus on any control, including a form | Use the SetFocus method. | |
| Create a Listbox | Select the Listbox control in the control palette. |  |
| Add a Data Structure specification | Enter an S in the first column of the code editor. | |
| Add a Data Structure Field specification | Enter an B in the first column of the code editor. | |
| Resize a graphic to fit a control | Set the Stretch property for an Image control to True . | |

More Information

To find more information on the topics introduced in Step 5, refer to the following help file topics.

Help File Topics:

- Change event
- Enabled property
- Methods, listing of
- ListCount property
- ClearObj method
- TabIndex property
- TabStop property
- LOADPIC
- Image control
- Stretch property

This Page Intentionally Left Blank



Menus and Forms

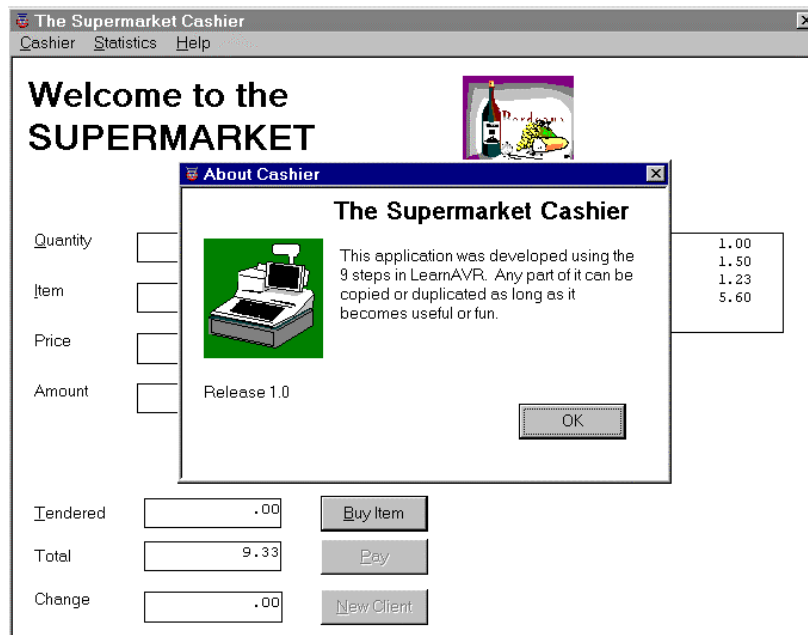
What you will learn in Step 6:

- How to add menu options on a form using Menu Editor.
- How to add a second form to a program.

Approximate Time to Complete Step 6:

45 minutes.

What the Application will look like after completing Step 6:



Providing a Menu to the Users

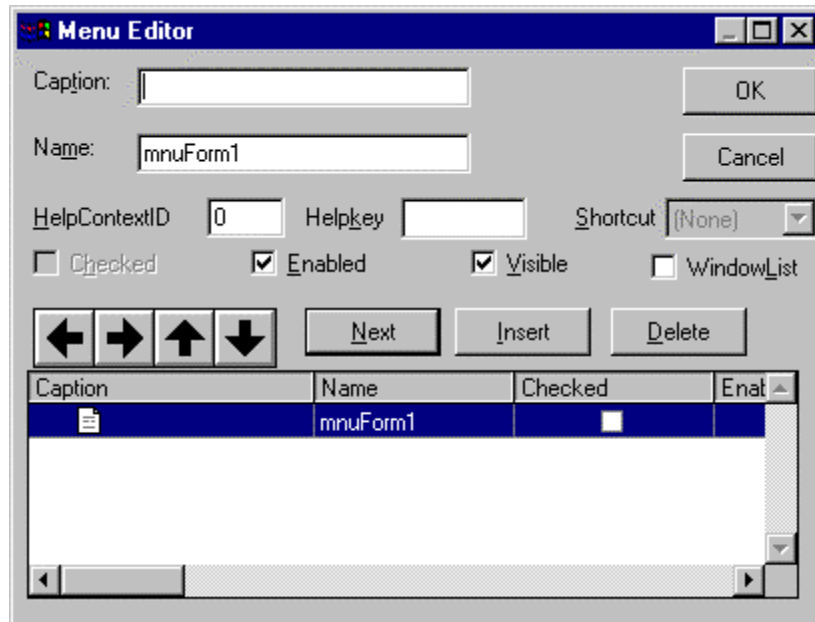
You can give your applications a professional look by providing menu options to surface the functionality of your program. When a user runs a new application, it is common to browse through the menu options to view the capabilities of the program. Menus not only serve as a way of driving the application, but also as a road map to it.

A **menu** or menu option is conceived within AVR as a **control**. As a control, it has properties, methods and can trigger events. Each form can have its own menu and options. A menu control will either display a custom menu for your application, or specify the Root Menu control for a form.

The properties of a menu can be manipulated at run-time and at design-time. You can set menu properties at design-time using the property window like any other control. There is, however, a tool to manipulate menus at design-time called the **Menu Editor**.

The Menu Editor

The **Menu Editor...** menu option from the **Tools** menu allows you to quickly and easily create custom menus. Menu items are added to a menu at design-time by creating menu controls and setting their properties.



When Menu Editor is opened, a default Caption (mnuForm1) and name will display. MnuForm1 represents the “**root**” of the menu tree and its caption will not appear on the form. This “**root**” is the first menu item in the tree list. **Do not delete this menu item.** However, you can change its Caption and Name. Changing properties of the “**root**” menu item affects all menu items on the menu tree. For example, setting its **Visible** property to False makes all menus and menu options on the form invisible.

For an analogy of “root” menus and menus, think of a menu as a tree. The root of the tree is the entire menu bar. Each menu in the menu bar is a branch of the tree, and each menu option is either a sub-branch or a leaf.

If there are already menu items defined for the selected form, those menu items will display when Menu Editor is opened, so changes can be made.

The Menu Editor window can be resized or maximized to display all the columns in the bottom portion of the Menu Editor. The columns can also be resized by selecting and dragging the separator bar in the column headings.



Menu Editor

- To Open Menu Editor, select the desired form, then select **Menu Editor...** from the **T**ools menu, press the **Ctrl+U** keys, or select the **Menu Editor** icon from the Toolbar.
- To Close the Menu Editor, select the OK or Cancel buttons. To accept changes in the Menu Editor, select OK.

Components of the Menu Editor

Caption is the name of the menu or menu option as it will appear on the form.

Caption:

- You can add an access key to a menu item to give the user keyboard access to that menu item. To do this, type an ampersand (&) in front of the character that the user can enter for easy access. The character **after** the ampersand will be underlined. The user can access that menu or menu option by pressing the **ALT** key and the character that is underlined. Enter two consecutive ampersands if you wish to display a single ampersand in a menu or menu option.
- To add a separator bar to your menu, type only a single hyphen (-).

Name is the name to reference the menu control from your **RPG code**; it will not appear in a menu.

Name:

It is recommended to try to keep these names somewhat consistent. For example, all the names for menu options under File could start with mnuFileXYZ, all menu options under Edit start with mnuEditXYZ, etc.

HelpContextID is the number that you assign as a unique value for the help context ID. This value is used to find the appropriate Help topic in the Help file.

HelpContextID

HelpKey is a keyword that identifies the requested Help topic for a control to provide context-sensitive Help for your application when the user presses the F1 key while focus is on the control. You can also change it by using the HelpKey property in the property window.



Shortcut allows you to specify a key, or a combination of keys that the user can type to activate a menu option. A drop-down ListBox is provided from which you can select the desired shortcut key. You can also use the **Shortcut** property in the property window.



Check Boxes:

- Checked:** When selected, this option puts a check mark (✓) next to a menu option, indicating the current selection.
- Enabled:** When selected, this enables the user to access the menu option. A disabled option will appear gray, and will not be available to the user.
- Visible:** When selected, this allows the option to be visible on the screen.

You can also enable or disable these options by double-clicking the box of the corresponding row and column in the **bottom portion** of the editor window.

WindowList, when selected, determines if the menu control contains a list of open MDI child forms in an MDI application.

Use the **up and down keys** to change the position of a menu item in the list.



Use the **right and left keys** to change a menu level. For example, selecting the right arrow will indent a menu, creating sub-menus.



You can have as many levels of sub-menus as you would like. Selecting the left arrow will move the menu item to a higher level.

The **ListBox**, the bottom portion of Menu Editor lists all of the information entered or selected in the upper section of Menu Editor. When you type a menu name in the Caption text box, that name will also appear in the ListBox. Selecting an existing menu item from the ListBox allows you to edit the properties for that control. You can change any field located from within this section.

The position of the menu in the ListBox determines whether the control is a menu (first indent), menu option (2nd indent) and subsequent indent levels. You can have as many menu levels as you would like. However, for usability, it is recommended to have no more than 4 levels of menus.

You can also select multiple items using the Shift key to select a range, or the Ctrl key to select individual items. This is useful when deleting items or moving items using the arrow keys.

When Menu Editor is opened, only the Caption, Name, Checked and Enabled columns will display. To view all columns on the screen, you can:

- maximize the screen
- resize the window
- use the horizontal scrollbars
- resize the columns by selecting and dragging the separator bar in the column headings

You can change the **Caption** and **Name** fields within the ListBox by double-clicking the name and entering a new name. The Checked, Enabled and Visible selections can be enabled or disabled by double-clicking the appropriate box.

The Menu Editor also has the following buttons:

- Next: Takes you to the next existing menu option. If another menu option does not exist, a new menu line will be created.
- Insert: Adds a new line below the menu option that is highlighted.
- Delete: Deletes the currently selected line.
- OK: Closes Menu Editor and applies all changes to the active form.
- Cancel: Closes Menu Editor and cancels all changes.

Steps to Create Menus using Menu Editor

The following summarizes the steps to follow to add menus to your forms. Refer to this procedure when adding menus to your application.

To Create a Menu

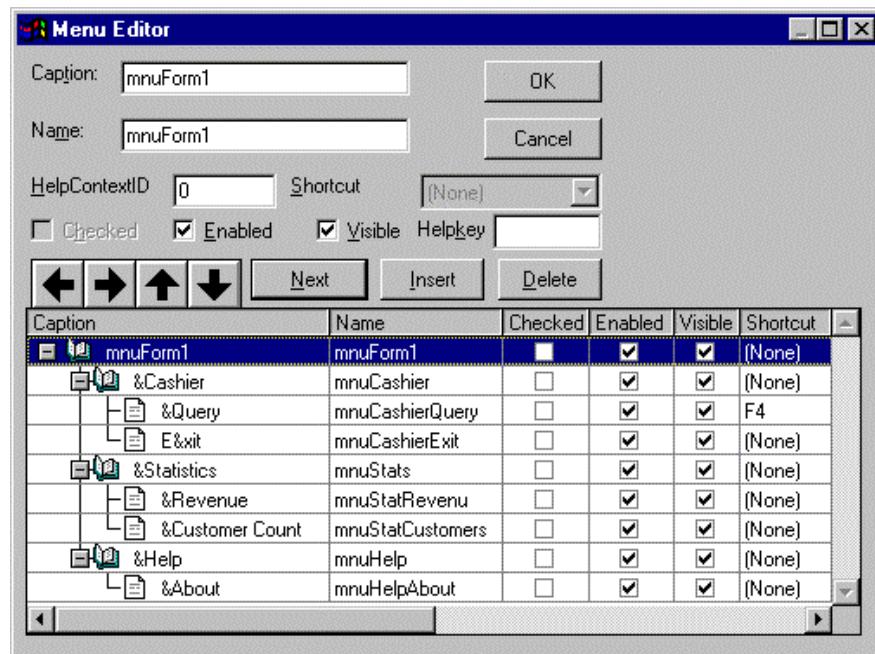
1. Select or open the form you wish to add or change a menu to.
2. To Open Menu Editor, select the desired form and then select **Menu Editor...** from the **Tools** menu, press the **Ctrl+U** keys, or select the Menu Editor icon from the Toolbar.
3. If this is a new menu, press the Next button to create a line for the first menu. Otherwise, select the line where you want to create the new menu and press the **Insert** key.
4. In the **Caption** text box, type the text for a menu as you want it to appear on the menu bar. Enter an **&** in front of the letter you want to be the access key. At run-time, this letter is underlined, and the user can access the menu or menu options by pressing **ALT** plus the access key.

To add a separator bar to your menu, type a single hyphen (-) in this box. The menu text is displayed below in the ListBox.
5. In the **Name** text box, type the name that you will use to refer to the menu control from within your RPG code. This name can be of any length. (You must enter a unique name for each menu option).
6. In **HelpContextID**, if this menu item will be linked to a help file topic in a help file, enter the context id number assigned to that topic when the help file was created. (Note, if the help authoring person generated the context id number, or the number is not known, get with the appropriate person for the number(s)).
7. Select a **Shortcut** key for this menu item, if desired.
8. Set the options for the menu as desired (checked, enabled, visible).
 - Select the **Checked** box if you want a small check mark to appear at the left of a menu item to indicate to the user that the option is selected.
 - Select the **Enabled** box if you want the menu item to initially respond to events. Disable the box if you want the menu item to be unavailable and appear dimmed on the menu.
 - Select the **Visible** box if you want the menu item to appear on the menu.
9. Select the **WindowList** option to determine if the menu control contains a list of open MDI child forms in an MDI application.

10. Select the **Insert** button to add another menu line.
11. Repeat steps 4-10 for each menu item to add.
12. Select **OK** to close the menu editor window when you have created all the menu controls for that form.
13. The menus you created will display on the form when the form is run. Click on a menu to display its menu options.

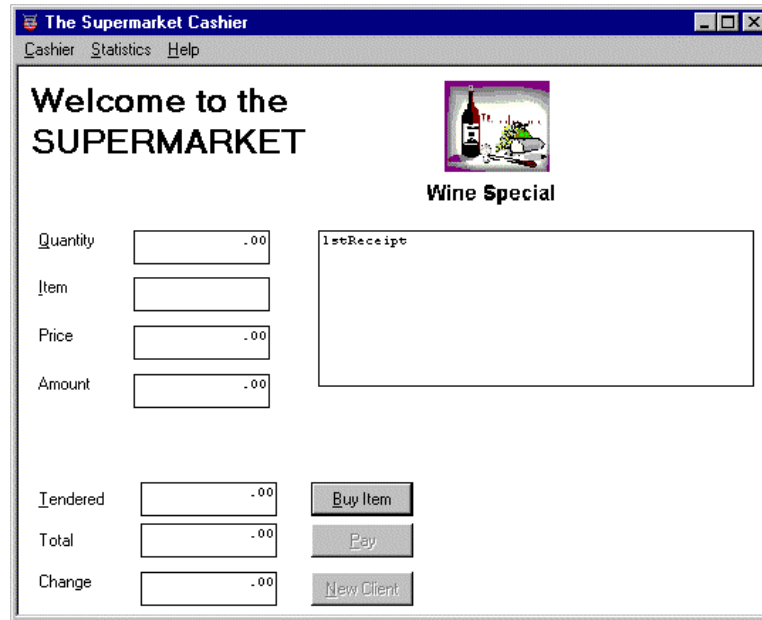
✔ Let's Add Menus to our Form

1. Add the following menu structure to your application.



Leave the help fields to their defaults.

2. After you have created all menus and menu options, press the **OK** button. Your form should look like the following when you get back to the IDE in design mode:

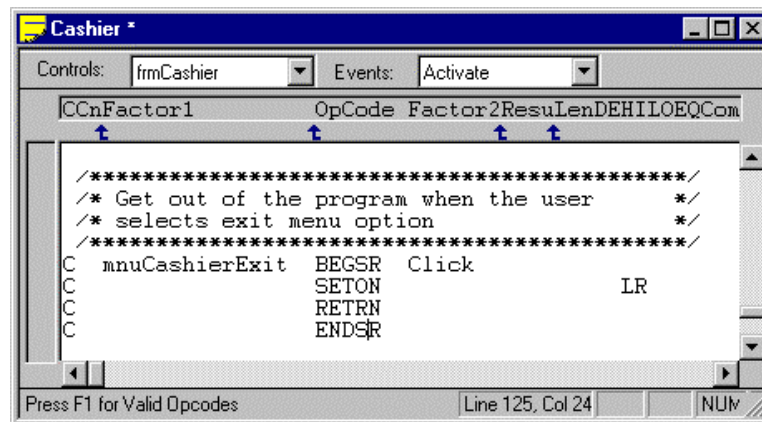


Menus, sub-menus and menu options are **controls**. All menu controls and their properties are displayed in the property window. Since there is no control on the form to select to display its properties, select the desired menu control in the Objects List at the top of the property window.

Menus and sub-menus do not have events. Menu options trigger only one event - **Click**.

✔ Let's Add a Menu Event

1. Let's add the following handler for **mnuCashierExit**:



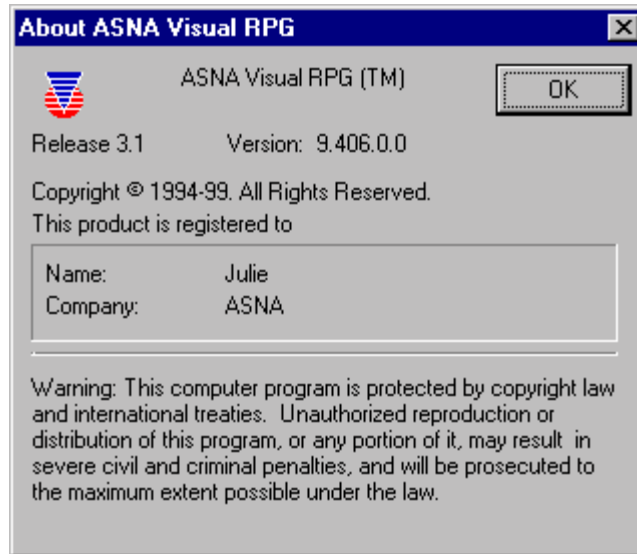
2. **Run** the program.

Except for the **Exit** option, when you select any other menu option, nothing should happen because we have not provided any handlers for the other options.

Showing a Second Form

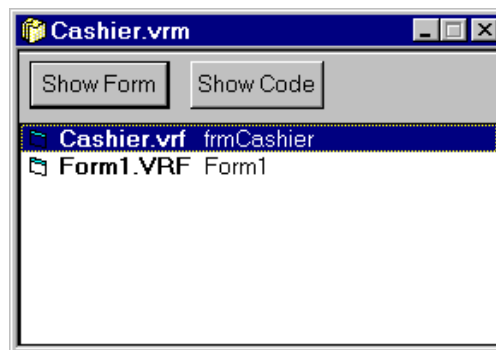
A single application may have more than one form. In this section, you are going to create an About Box for your application and show it when the user selects the menu option **A**bout from the **H**elp menu.

To see what an About Box looks like, display the one in the AVR IDE:



Let's Add a Second Form

- To add a new form to your application, select **N**ew **F**orm from the **I**nsert menu. When you do, two things will happen.
 - First, a new blank form (Form1) will appear in the IDE with the focus on it.
 - Second, a new entry will appear in the project window showing that the application consists of two forms, as shown below.



2. Change the new form's properties as shown below:

| frmAbout | <u>Property</u> | <u>Value</u> |
|-----------------|-----------------|---------------|
| | BorderStyle | 3 |
| | Caption | About Cashier |

3. Add a Command Button, setting its **Name** to **btnOK** and **Caption** to **OK**.
4. Add an **Image** control and using the **Picture** property, insert **Register.BMP** from the **\Learnavr40\FoodMap** directory. (You may need to set **Stretch** to True to ensure the picture fits within the size of the Image control).
5. Add **3 Labels** in which a title, text description and release number will be placed.

| lblName | <u>Property</u> | <u>Value</u> |
|----------------|-----------------|---------------------------|
| | Caption | The Supermarket Cashier |
| | Font | MS Sans Serif - Bold - 12 |

| lblDescription | <u>Property</u> | <u>Value</u> |
|-----------------------|-----------------|---|
| | Caption | This application was developed... (see below for the rest of the text) |
| | Font | MS Sans Serif - Regular - 8 |

| lblRelease | <u>Property</u> | <u>Value</u> |
|-------------------|-----------------|-----------------------------|
| | Caption | Release 1.0 |
| | Font | MS Sans Serif - Regular - 8 |

Your About Box should now look like the following:



In the next step, you will add a **Click** event when the **OK** button is selected.

✓ Let's Add an Event Handler for the Form

1. Open the code editor by either double-clicking anywhere on the form, selecting **C**ode from the **V**iew menu, or pressing **F7**.
2. When you bring up the editor for this new form, it will be blank. Let's add the following code:

```

Form1 *
Controls: btnOK Events: Click
CCFactor1OpCode Factor2ResuLenDEHILOEC
/* Close this about box */
C btnOK BEGSR Click
C SETON LR
C RETRN
C ENDSR
Press F1 for Valid Opcodes Line 4, Col 13

```

3. Add a handler for the Click event of **mnuHelpAbout** in the **frmCashier** program. Double-click on the Cashier form to display its code, as shown below:

```

Cashier *
Controls: mnuHelpAbout Events: Click
CCnFactor1 OpCode Factor2 ResuLenD
/* Show the about box */
C mnuHelpAbout BEGSR Click
C EXFMT frmAbout
C ENDSR
Factor2 Line 115, Col 29

```


EXFMT has been adapted in AVR to show a form on the screen and wait for the user to close it down before execution continues. In that sense, it is very similar to what EXFMT does in traditional RPG. Showing a form and waiting for it to be closed in Windows is called a **'modal'** show.

Another way of showing a form in which the rest of the application does not become 'frozen' while the window is up is called **'modeless'**. AVR has an op code called **SHOW** which shows a form in this fashion.

Let's Show a Form in Code

1. Experiment running the application, and then change **EXFMT** to a **SHOW** and see the difference in behavior.
 - Do not pay attention to what you can do with the about box, but to what can be done on the Cashier form.
 - When you are finished playing with the application, make sure the form will be shown as a modal window using EXFMT.
2. When you **Save** your project, AVR will prompt for a file name for the new form. Call it **About.vrf**.

Step 6 Summary

| To | Do This | Button/Keys |
|--|---|--|
| Add a menu to a form | Select Menu Editor... from the Tools menu. |  Ctrl+U |
| Assign a character within a menu option as an access key | Type an ampersand (&) before the character in which you want the user to select in combination with the Alt key. | |
| Add a separator bar to separate menu items | Type a single hyphen (-) in the Caption field of the Menu Editor. | |
| Assign a shortcut key to a menu option | Select the key combinations from a drop-down menu in Shortcut in Menu Editor, or use the Shortcut property in the property window. | |
| Assign context-sensitive help to a menu option | In HelpContextID in the menu editor or property window, enter the context ID number assigned to the topic in a help file. | |
| Add another form to an application | Select New Form from the Insert menu. | |
| Show a modal form | Use the EXFMT op code. | |
| Show a modeless form | Use the SHOW op code. | |

More Information

To find more information on the topics introduced in Step 6, refer to the following help file topics.

Help File Topics:

- Menu Editor
- Shortcut property
- HelpContextID property
- HelpKey property
- Creating menus
- Showing forms
- EXFMT
- SHOW



Using a Subfile

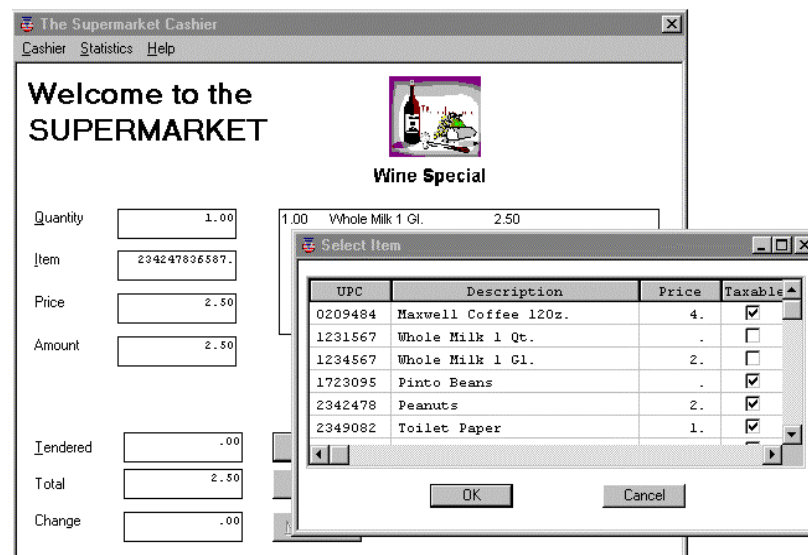
What you will learn in Step 7:

- The structure of an application.
- How to code a procedural program.
- How to define and do I/O on a Subfile.
- The various properties and events available to a Subfile.

Approximate Time to Complete Step 7:

1 hour, 15 minutes.

What the Application will look like after completing Step 7:



Application Structure

Only the simplest of applications will consist of a single form. In the previous step, you learned how to add a second form to an application and how to make it available to the user with the EXFMT and SHOW op codes. Let's look at how AVR organizes an application with multiple forms.

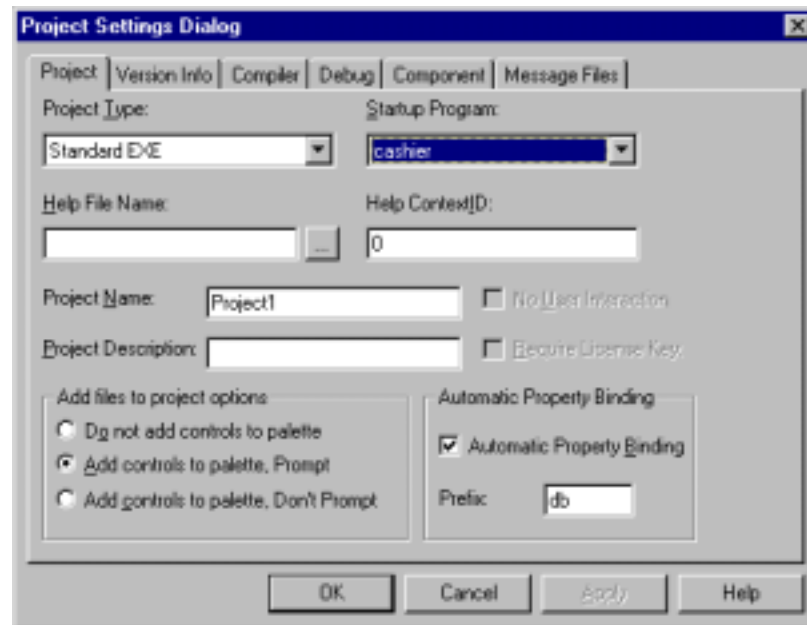
Each form is composed of two sections: **visual** and **logic**.

- The **visual** section is the form definition and all the controls within it.
- The **logic** section consists of the file and variable definitions, the code in the main C-Specs, and the subroutines.

In AVR, a form is actually a program; to be precise, a **Form Program**. There is another type of program known as a **Procedural Program**, which does not contain a visual section, indicating it has no form or controls attached to it.

As we have seen, the project file (.VRM) lists the names of the Form Programs (.VRF) which make up the application. If the application includes any Procedural Programs (.VRP), they will be also listed in the project file.

By default, the **First** program created is the one to obtain control when the application starts, and will display first in the project window. However, this first or **Main** program can be set to any of the programs by selecting **Project Settings** from the **Project** menu, as shown below.



The **Main** program, besides being the **Startup** program, is the one controlling the end of the application. When the Main program returns with the indicator LR set to on, the entire application is closed down. Another important role the Main program plays is being the container of variables and subroutines that are *global* and accessible to *all* of the other programs.

Any variable, including the files defined in the F-Specs of the Main program, can be seen and modified by any Form or Procedural program in the application! However, be careful. Having global variables makes communication between programs very easy, but at the same time, can make certain kinds of bugs hard to track. For instance, when a program stores a value on one of these global variables, this new value will be seen by all other programs. If another program was using that variable, the old value would be gone. Thus, whenever using global variables, be aware that *any* other program within the application may change its value on you.

Because the Main program's variables have this global behavior, you will want to keep them to a minimum and have their use well known, or should we say, well documented.

Given that a form is actually a program, it can be “*called*” and parameters passed to it. When used with Procedural programs, the **CALL** and **PARM** op codes behave the same way as in traditional RPG. However, when used with a form program, before control is given to the main C-Specs, the form will be shown on the screen and focus will be given to it. When the form program executes the **RETRN** statement, or hits the end of the main C-Specs, the form disappears from the screen if *INLR is on. If *INLR is off, the form stays on the screen. When a form is shown because of a **CALL**, it uses the same modal behavior as **EXFMT**.

So far in our application, we have not needed global variables, so let's add a new procedural program to play the role of the Main program, thus hiding the variables of the Cashier form. The new program is very simple, containing a call to the Cashier's form and a line to set on *INLR.

Let's Add a Procedural Program

1. To add a new procedural program to the application, select **New Program** from the **Insert** menu.

Because the procedural program has no form associated with it, the IDE immediately displays the code editor.

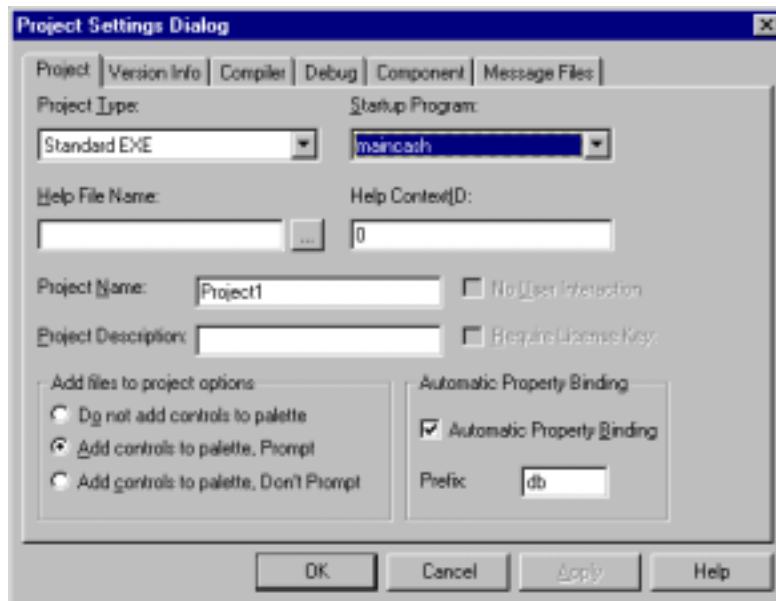
2. Type in the following lines:

```

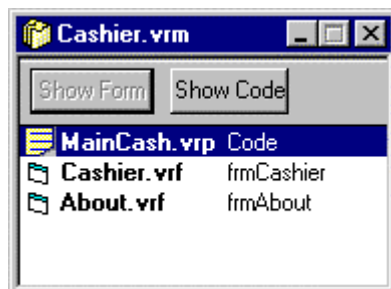
CCnFactor1 OpCode Factor2 ResLen DEHILOEQ Comment
/* ***** */
/* Cashier application's main program */
/* ***** */
C CALL frmCashier
C SETON LR

```

3. Select **S**ave from the **F**ile menu to save the program and give it the name **MainCash**.
4. Select **P**roject **S**ettings from the **P**roject menu. The **P**roject tab will display. Set the Startup Program to **MainCash**, by clicking on the arrow to the right.



5. The project window will now display the following:



Notice that MainCash is now listed first in the project window, representing the Startup program.

6. **Run** the program.

Notice how the user-side of the application has not changed. You can still view the About Box, enter items in the UPC field and compute the change owed to the client.

We have, however, isolated variables of the Cashier from the rest of the project, thus reducing the possibility of an inexperienced programmer introducing bugs when maintaining the application.


The Subfile Control

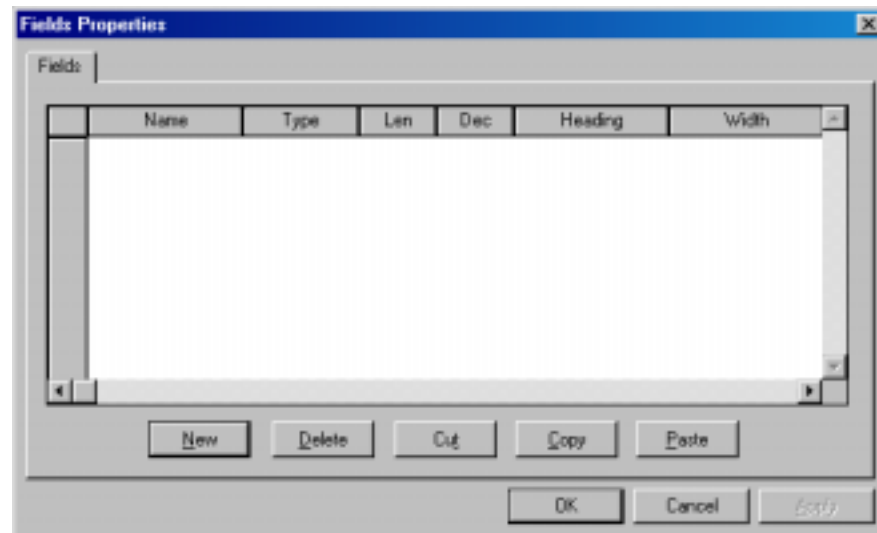
Next, we want the Cashier clerk to be able to search through a list of all available products and select one. For this, we will create a new form and use a Subfile control.

The **Subfile** is a powerful control allowing the user to browse and edit data on a grid of IOFields and CheckBoxes. This control is tightly integrated into the AVR language. The op codes: **CHAIN**, **READC**, **UPDATE** and **WRITE** are used to communicate records to and from the control. The Subfile has a special property called **Fields** where the description of its fields are entered.

It is common in RPG to name the fields of a Subfile with names of fields in a database file so that a simple loop reading from the database file and writing to the Subfile will populate it with records.

Let's Add a Subfile Control

1. Insert a new form and **Name** it **frmSelectItem**. Set its **Caption** to **Select Item**.
2. Select the **Subfile** control from the control palette and place it onto the form.
3. In the property window for the Subfile, select the **Fields** property, then click on the  button. The following dialog box will display.



Every entry in the **Fields Properties** window defines a **column** in the Subfile control. Here you specify the field names with their respective Type, Length, Decimals, Column Heading, Column Width, Edit Word, Edit Code, Hidden, Browse, UpperCase, Alignment and DropDown properties.

Not all field properties columns can be displayed at the same time. Use the horizontal scroll bar to display more columns. Notice that the name column is frozen and does not scroll.

Press the **New** button to begin adding fields to the Subfile. Each column in the dialog box refers to an existing Subfile property, as listed below.

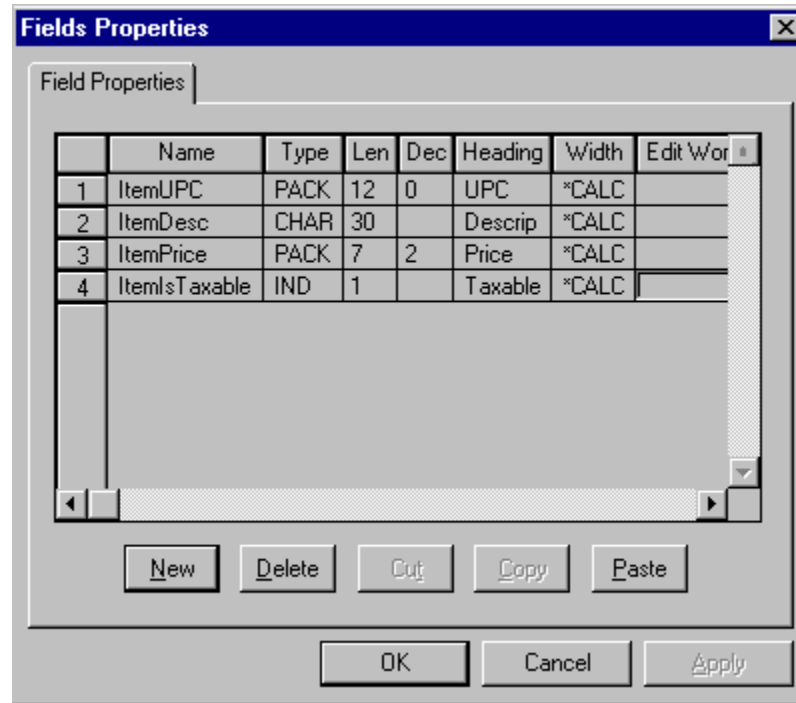
| | |
|-----------------|---|
| Name | References the previous FldNames property, which no longer exists. This is the name of a Subfile field. |
| Type | References the previous FldTypes property, which no longer exists. This is the data type of the Subfile field. CHAR is the default. Use the drop-down combo to display the other types: IND, ZONED, PACKED or BINARY, or key in the first letter of the type (e.g., key "P" for a packed data type). |
| Len | Enter the length of the Subfile field. |
| Dec | Enter the number of decimals for the Subfile field, if numeric. |
| Heading | References the FldColHeading property, which is also a run-time property. If set to blanks, the Heading defaults to the field name. |
| Width | References the FldColWidth property, which is also a run-time property. *CALC is the default, which automatically calculates the column width based upon the greater of the field and Heading lengths. Width may also be an absolute value (expressed in Twips). If set to an absolute value, the corresponding Subfile column width will not be automatically resized during run-time. |
| EditWord | References the FldEditWord property, which is also a run-time property. The EditWord edit mask is the same as documented for the EditWord and NumEditWord properties. Fields are not edited with the Edit Code/Word until a record is written to the Subfile. Previously written records will continue to display field values with the Edit Code/Word in affect at that time. |
| EditCode | References the FldEditCode property, which is also a run-time property. Specify an Edit Code value for the field or select from the drop-down combo. |

| | |
|----------------------|---|
| Hidden | References the FldHidden property, which is also a run-time property. Check the box to set the property to True. The field will be hidden at design-time and at run-time. |
| Browse | References the FldBrowse property, which is also a run-time property. Check the box to set the property to True. The field may be browsed by the user, but may not be modified. The FldBrowse property affects the entire column of cells referenced. |
| UpperCaseOnly | References the UpperCaseOnly property, which forces the entry of UPPERCASE characters. Check the box to force the characters to Uppercase. |
| Alignment | References the Alignment property. Select the desired alignment of 0 - left, 1 - right or 2 - center by clicking on the arrow to the right and selecting the desired choice. This property is not accessible at run-time. |
| DropDown | References the FldDropDown object. Select the desired alignment of 0 – None, 1 – Simple List, 2 – Drop down list, or 3 – Drop Down Combo. |

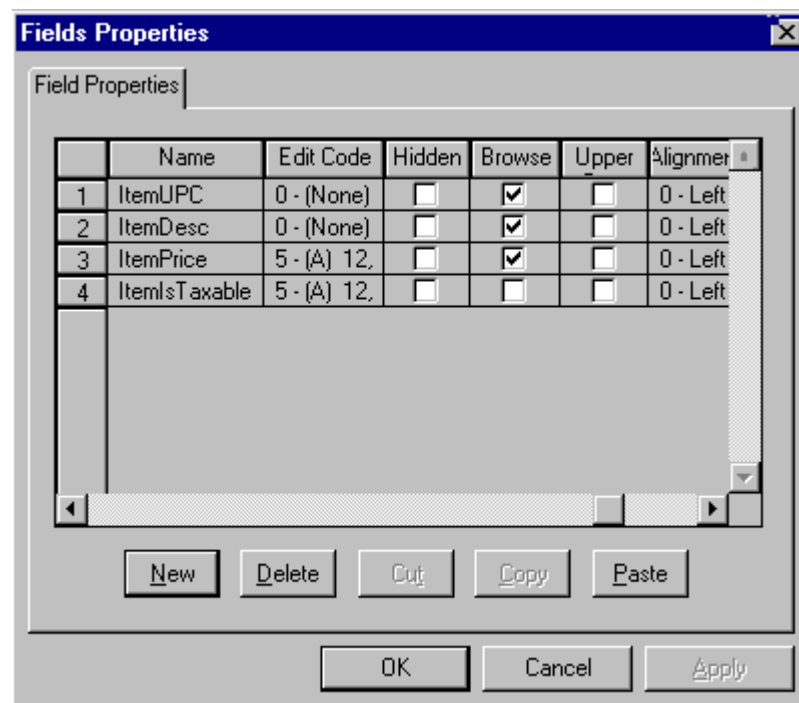
The fields dialog box also contains the following buttons.

| | |
|--------|--|
| Delete | Select to delete the current line, or all lines selected. |
| Cut | Select to cut the selected fields. To select fields to cut or copy, click on the row button to the left of the field name. Multiple rows are selected with Shift+Click. |
| Copy | Select to copy the selected fields. To select fields to cut or copy, click on the row button to the left of the field name. Multiple rows are selected with Shift+Click. |
| Paste | Select to paste the fields cut or copied. Pasted fields will be inserted following the selected row. |
| OK | Select to accept the Subfile entries and close the Fields Properties dialog box. |
| Cancel | Select to close the fields dialog box. |
| Apply | Select to accept the Subfile entries and leave the fields dialog box open. |

3. Press the **New** button prior to adding each of the following fields to your Subfile.



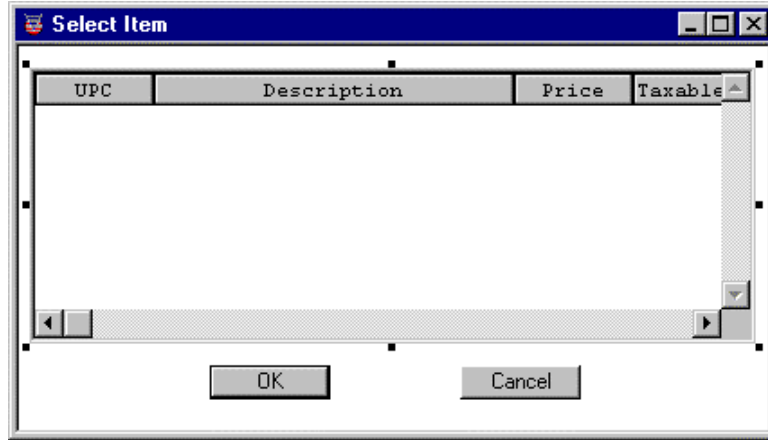
4. Scroll to the right and add the additional information, as shown below, then press the **OK** button when finished.



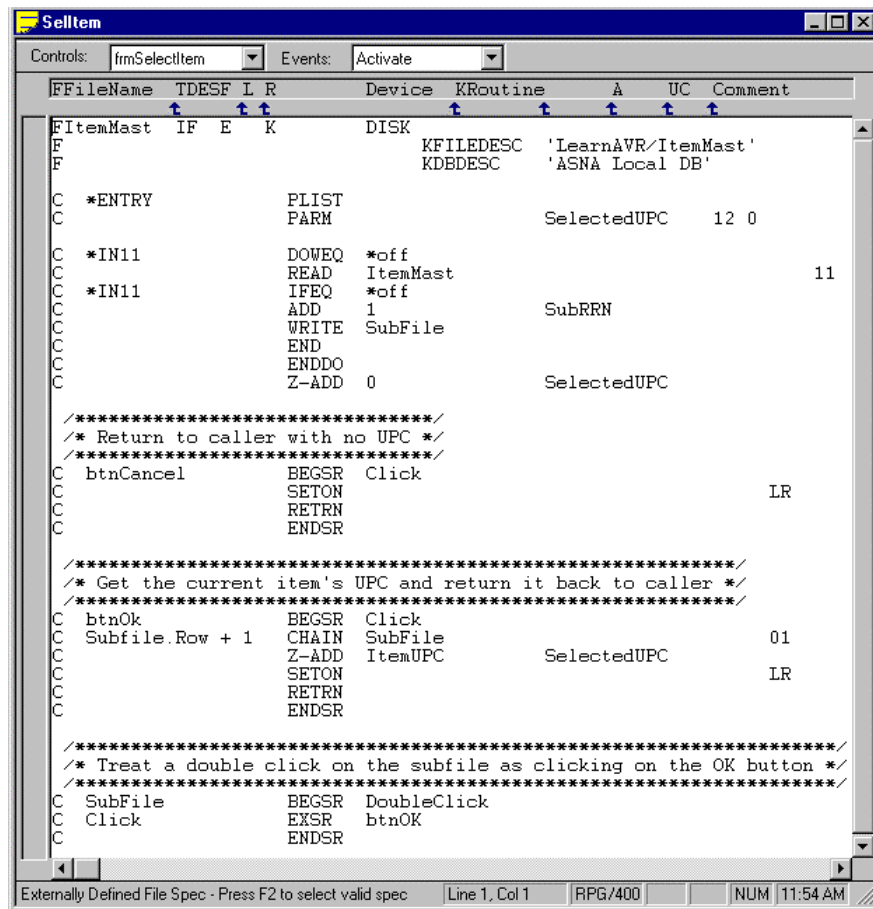
5. Click on the Subfile control and name it **SubFile** using the Name property.
6. Add two CommandButtons to your form, **btnOK** and **btnCancel**.
 For **btnOK** - set its **Default** property to True.
 For **btnCancel** - sets its **Cancel** property to True.

With the above property settings, pressing Enter is equivalent to clicking the OK button, and pressing the Esc key is equivalent to clicking the Cancel button.

The new form should look like the following:



7. Enter the code for the new form, as shown below.



8. Save this new form with the name **SellItem.vrf**.


```

Cashier *
Controls: mnuCashierQuery Events: Click
CCndInd Factor1 OpCode Factor2 Result LenDEHILOEQC
/*****
/* Find item using UPC, update ioFields and compute Amount */
*****/
C ComputeAmount BEGSR
C Z-ADD ioItem ItemUPC
C ItemUPC IFEQ 0
C MSGBOX 'Invalid UPC'
C RETRN
C ENDIF
C ItemUPC SETLL ItemMast
C READ ItemMast 91
C
C /* Update ioFields */
C Z-ADD ioQuantity Quantity 7 2
C ItemPrice MULT Quantity Amount 7 2
C MOVE ItemPrice ioPrice
C MOVE ItemUPC ioItem
C MOVE Amount ioAmount
C ENDSR

```

Calculation Spec Line 126, Col 1 NUM 04:48 PM

12. The following is the new version of **btnBuy-Click**.

```

Cashier *
Controls: mnuCashierQuery Events: Click
Free format code and comments
/*****
/* Handle the event Click for the button Buy */
*****/
C btnBuy BEGSR Click
C ERSR ComputeAmount
C ADD Amount Total 9 2
C MOVE Total ioTotal
C ERSR EnablePay
C
C /* Add Item to the list */
C MOVE ioQuantity LineQuantity P
C MOVE ItemDesc LineDesc P
C MOVE ioAmount.Text LineAmount P
C EVAL lstReceipt.AddItem(LineItem)
C SUB 1 BottomItem 5 0
C Z-ADD BottomItem lstReceipt.TopIndex
C
C /* Clear input fields */
C Z-ADD 1 ioQuantity
C MOVE *BLANKS ioItem
C ENDSR

```

Enter free format code and/or comments here Line 15, Col 2 NUM 04:03 PM



If you have any problems compiling or running your program, consult the enclosed code in the \LearnAVR3\Step7 directory.

The application has taken a very simple approach for populating the Subfile with records. It is using what is usually known as a '**Load-All**' technique. The other two common techniques are: '**Expanding**' and '**Single-Page**'.

- The **Load-All** technique is the simplest, but for files with more than a few dozen records, it is impractical because it takes a long time to load, especially if the source of the records is a Server across a slow network.
- The **Expanding** and **Single-Page** techniques traditionally have needed to know the number of records fitting on a 'page'. The Subfile control does not have a predefined page size or a maximum number of records. Both of these values are dynamic and are controlled at run-time by the programmer. If you want to implement a Subfile with a 'page size' of 10, then just add 10 records to it. If you attempt to add an eleventh record, you will not get an error.

For a **Single-Page** Subfile, use the **ClearObj** method to delete all the records in the Subfile. When the user attempts to scroll beyond the end of the records in the Subfile, an event will be triggered, allowing your program to respond by adding more records to the Subfile. Similarly, if the user tries to scroll above the first record in the Subfile, an event will be triggered.

Subfile Properties

The following is a list of some of the more interesting properties and events of the Subfile control.

To get the complete list of all the properties available for the Subfile, either click on the Subfile control in the control palette or a Subfile on the form, and press **F1**. The Subfile Control help topic will display. Click on the **Properties** link. (Remember that there are some properties which are available only at run-time and others available only at design-time).

AutoExtend

The AutoExtend property allows the Subfile to function in a data-entry mode, with a blank record at the bottom. When this property is set to True, the user can add additional records to the Subfile. When blank records are added, the RecNew event is generated. **Only** when this property is True is the KbdDelIns property valid, which allows the user to Insert and Delete records to the Subfile. Care must be taken to maintain internal control of the number of records in the Subfile or to use the RowCount property when processing in "Page" mode. Loading the Subfile in "Expanding" mode does not require this property to be True.

Browse

The Browse property allows data to be displayed, but does not allow editing in any of the fields.

FirstRow

The FirstRow property sets the specified row to be displayed at the top of the Subfile at run-time only. Setting the FirstRow property causes the Subfile to scroll so that the specified row is the topmost row in the Subfile. FirstRow can also be used to query the row currently at the top of the Subfile. Rows in the Subfile are numbered relative to zero; i.e., the first row is row number zero.

FldHidden

The FldHidden property allows the program to set and get a hidden field. The FldHidden property is available at run-time only, but can be set at design-time in the Fields Properties dialog box. FldHidden is an array property to turn On and Off the visibility of a field to a user.

FrozenCols

The FrozenCols property specifies the number of columns on the left side of the Subfile that do not scroll horizontally. This property is useful when it is desirable to maintain a customer name, for example, into the user's view when scrolling horizontally.

KbdDelIns

The KbdDelIns property allows the user to use the Delete and Insert keys at run-time to delete and insert records. If the user deletes a record, the RecDelete event will be received prior to the record actually being deleted from the Subfile. If the user inserts a new record, the RecNew event is received by the program. The new record will also have the RecNew array property set to True. The AutoExtend property must be set to True for the KbdDelIns property to be True.

NoLines

The NoLines property displays the Subfile without lines separating the cells. The lines created when NoLines = False will be gray. Note that if your Subfile background is also gray, you will not be able to see the lines.

RecChanged

The RecChanged property is an array property to set and get a flag when the contents of a record have changed.

RecBrowse

The RecBrowse property gets or sets the browse state of data in a Subfile to be controlled at the record level.

Row

The Row property sets or gets temporary Row and Column coordinates maintained by the Subfile for the purpose of communicating cell locations between the application and the control. The Row property is set by the Subfile to the new coordinates prior to a SelChanging or SelExtending event. Changing the Row property in response to these notifications changes the resulting selection. The Row property must be set by the application prior to setting or getting the CellData property in order to specify which cell's data to set.

Subfile Events**Bottom**

The Bottom event occurs when the user attempts to scroll off the bottom of the Subfile. When loading a Subfile in "Page" or "Expand" mode, this event can be used to get more records to load into the Subfile.

RecChanged

The RecChanged event is issued when the contents of a record have changed. Care must be taken when updating your database files in response to this event if the Subfile contains the numeric fields with NumEditCode (other than 0 - none). In the case of invalid data entered into such a numeric field, this event is generated even though the user is not allowed to exit the cell. It is more practical to use the SelChanging event to reference the RecChanged property of the exited cell.

RecDelete

The RecDelete event occurs when a record is about to be deleted by the user. The user can only delete records from the Subfile if the KbdDelIns property is set to True. This event will be received prior to the record actually being deleted from the Subfile.

RecNew

The RecNew event indicates that a new record was added by the user. The RecNew event is only valid when the KbdDelIns property is set to True.


Top

The Top event occurs when a user has attempted to scroll off the top of the Subfile. When loading a Subfile in "Page" mode, this event can be used to fetch previous records to load into the Subfile.

VScroll

The VScroll event occurs when a user uses the Vertical Scroll Bar to scroll up or down.

Step 7 Summary

| To | Do This | Button/Keys |
|--|--|---|
| Specify a program as the Startup program | Select Project Settings from the Projects menu, then click on the desired form in Startup Program in the Project tab. | |
| Create a Subfile control | Click on the Subfile control in the control palette, and draw onto the form. |  |
| Enter fields in a Subfile control | Click on the Subfile control, then select the Fields property in the property window. | |

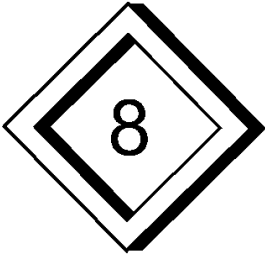
More Information

To find more information on the topics introduced in Step 7, refer to the following help file topics.

Help File Topics:

- Subfile control
- Overview, programs
- Main program
- Startup program

This page Intentionally Left Blank



Graphing Data

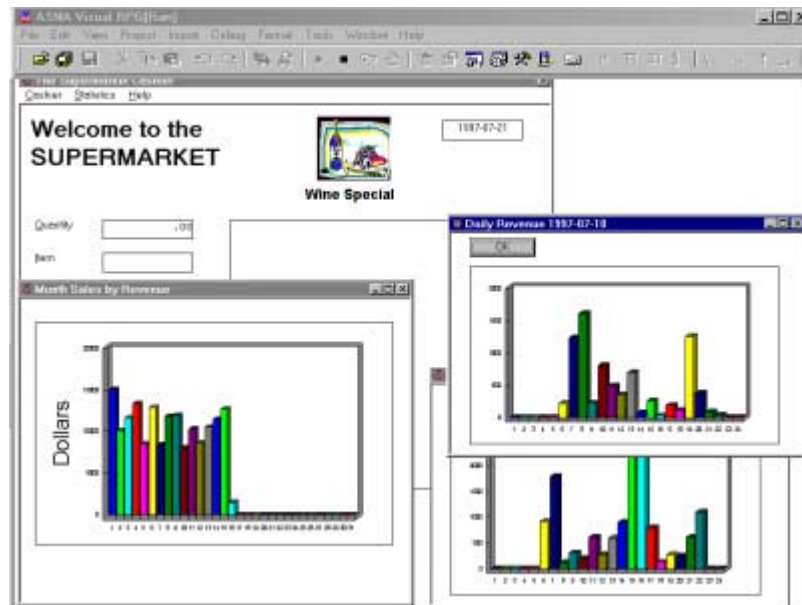
What you will learn in Step 8:

- How to use the graph control.
- How to receive parameters in an event handler.
- How to instantiate multiple copies of the same form.

Approximate Time to Complete Step 8:

30 minutes.

What the Application will look like after completing Step 8:



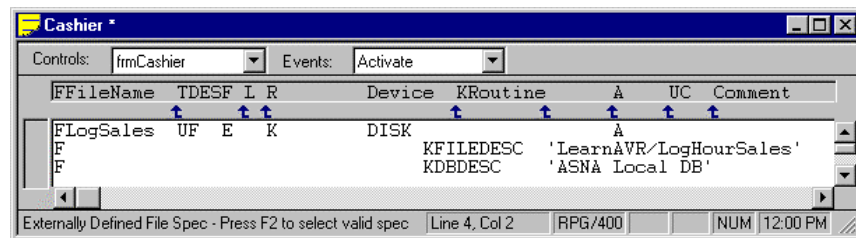
Adding a Graph

We are now going to enhance the application to keep track of all the sales occurring every hour. Once these statistics have been accumulated over time, the application will be able to display a graph showing the sales in dollars for every day of the month. Furthermore, the user will be able to click on any of the days in the graph and a new graph will display the sales per hour for that day.

The first thing we will do is accumulate the sales.

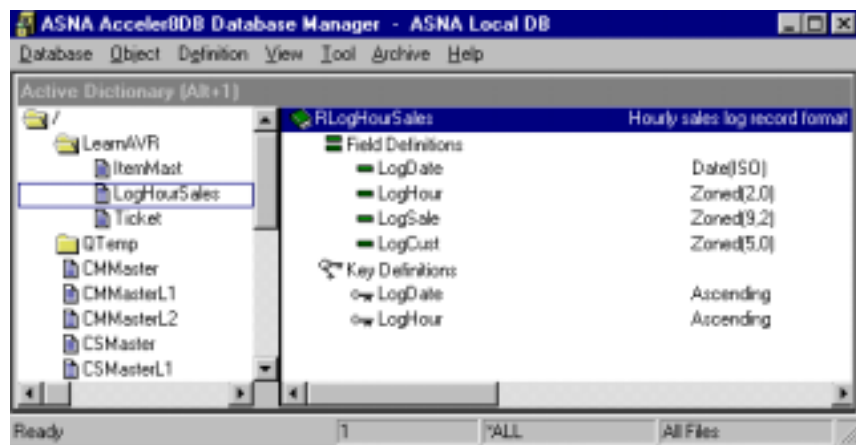
✓ Let's Add a new F-Spec for the file **LogHourSales**

1. Add a new F-Spec for the file **LogHourSales** contained in the directory LearnAVR, containing the ItemMast file, as shown below.



Notice how we have done an *override* of the file name from LogSales to **LogHourSales**. You can do this override at compile-time to select a different file's description, using the continuation line keyword FILEDESC, or at run-time using the keyword FILE. The same applies to the database containing the targeted file. The DBDESC keyword is used at compile time, and the keyword DB at run-time. Both FILE and DB keywords accept either a literal or a variable, while FILEDESC and DBDESC require a literal.

Here is the layout of LogHourSales:



Every time the user clicks on the **Pay** button, we can accumulate in the record corresponding to the day and hour of the total sale for the client. We will put this code in the subroutine **LogTotal** and execute as the last step in the **btnPay-Click** subroutine.

2. Add a new IOField **ioCurrentDate** to display the date in **frmCashier**. Set its **Text** to nothing and its **Alignment** to 2 - Center.
3. Add the following **LogTotal** subroutine:

```

Cashier
Controls: frmCashier Events: Activate
CCndFactor1 OpCode Factor2 Result LenDEHILOEQCon
*****
/* Log total sale on the record corresponding to now */
*****
LogTotal BEGSR
LogKey KLIST
KFLD KFLD LogDate
KFLD KFLD LogHour
TIME TIME Now
MOVE NowDate LogDate
MOVE NowHour LogHour
MOVE LogDate ioCurrentDate
LogKey CHAIN LogSales 01
/* If this is the first customer of the day/hour, create a new log record */
*IN01 IFEQ '1'
Z-ADD 0 LogSale
Z-ADD 0 LogCust
LogKey CHAIN LogSales 02
*IN01 IFEQ '1'
MSGBOX *App.DbErrorText
ENDIF
ENDIF
/* Accumulate sale and count customer */
ADD Total LogSale
ADD 1 LogCust
UPDAT LogSales
ENDSR

```

Notice the subroutine uses the **TIME** op code to obtain the current time and date into the structure **NowDS**, which divides it into **NowTime** and **NowDate**. **NowDate** is being moved into **LogDate**, which is defined in the file as a date field of ISO format.

4. Initialize the **ioCurrentDate** in the main C-Specs by using the **TIME** op code and enter the code as shown below.

```

Cashier
Controls: frmCashier Events: Activate
IName EODSExtFileName OccrLen
INowDS DS 14
I 1 14 Now
I 1 6 ONowTime
I 1 2 ONowHour
I 7 14 ONowDate
E ImageName 5 20
C EXSR LoadFood
C TIME Now
C MOVE NowDate LogDate
C MOVE LogDate ioCurrentDate

```

5. Insert a new form to the project, **Name** it **frmMonthRevenue**, and set its **Caption** to **Month Sales by Revenue**.

- Select the **Graph** control from the control palette, add to the new form, and set the following properties:

| <u>Property</u> | <u>Value</u> |
|-----------------|--------------|
| LeftTitle | Dollars |
| LeftTitleStyle | 1-Vertical |
| Name | gphMonth |
| NumPoints | 31 |
| True3D | 0-Off |

The **GraphData** property holds the actual data values you want to graph. The values are stored as a two-dimensional array indexed by the **ThisSet** and **ThisPoint** properties.

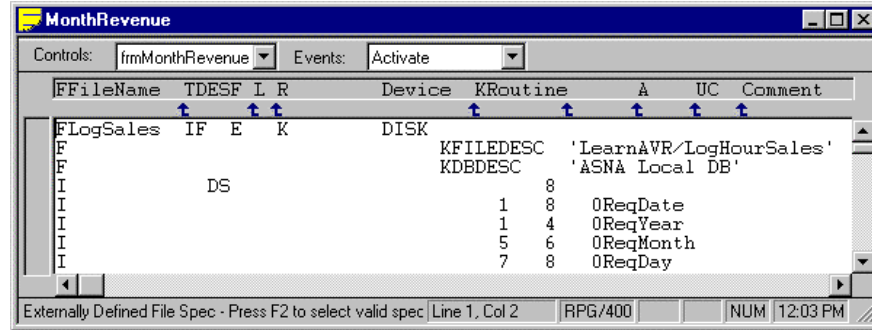
You can use the **AutoInc** property to automatically increment the **ThisSet** and **ThisPoint** values as you enter data.

- The main C-Specs of the **frmMonthRevenue** will compute the total sales for each day by reading the **LogSales** file. Enter the code as shown below:

```

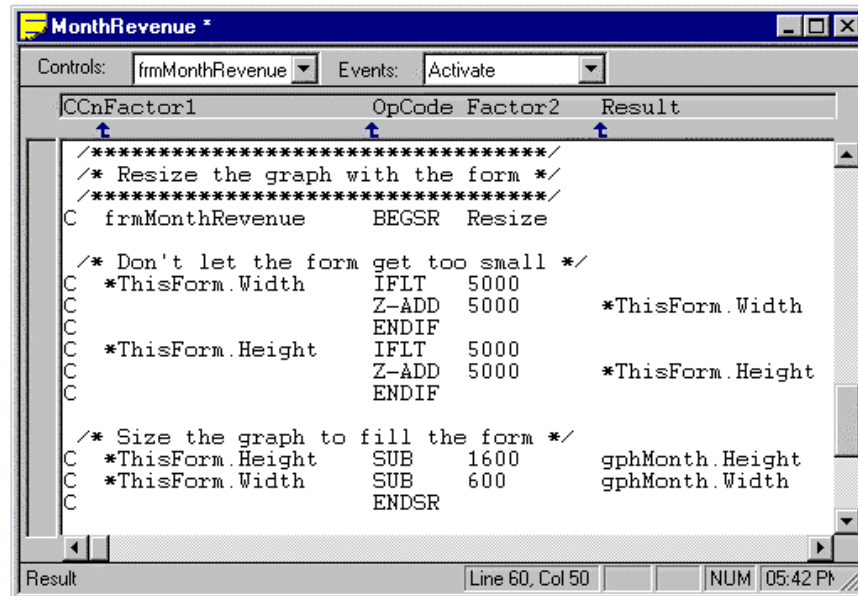
Code - C:\Program Files\ASNA\VisualRPG\StepToMonthRevenue.rtf
Details: frmMonthRevenue *  Events: Activate *  Show Form
-----
R001nsd  Factor1      OpCode  Factor2      Result      LenDENTLREQ
-----
C      *CHTRF      PLIST
C      PRRH
C      H00E      MonthStatsDate      ReqDate      8  8
C      2-800  1      CurrentDay      ReqDate      2  8
C      H00E      CurrentDay      ReqDate
C      H00E      ReqDate      ReqDate
C      2-800  8      DaySales      11  2
/* read first day of the requested month in the log */
C      LogKey      RLST
C      RFLD
C      LogKey      SETLL  LogSales      LogDate
C      READ      LogSales      LogDate
C      EXTRACT  LogDate:=#8      LogMonth      2  8
C      EXTRACT  LogDate:=#0      LogDay      2  8
C      *CHNS      IFED      '1'
C      LogMonth      GRNE      ReqMonth
C      H0000      'No sales in this month'
C      SETON
C      RETURN      LR
C      ENBIF
/* populate the first days of the month with 0 till we get to the first day with data */
C      1
C      00      LogDay - 1
C      2-800  8      gphMonth.GraphData
C      ENBDO
C      2-800  LogDay      CurrentDay
/* loop thru the days of the month computing the GraphData */
C      LogMonth      GRNEQ      ReqMonth
C      LogDay      IFNE      CurrentDay
C      2-800  8      gphMonth.GraphData
C      2-800  8      DaySales      DaySales
C      ADD      1      CurrentDay
C      ENBIF
C      ADD      LogDate      DaySales
C      READ      LogSales
C      LEAVE
C      EXTRACT  LogDate:=#8      LogMonth      2  8
C      EXTRACT  LogDate:=#0      LogDay      2  8
C      ENBDO
C      2-800  DaySales      gphMonth.GraphData
C      2-800  1      gphMonth.Visible
    
```

- Enter the F and I specs to look like the following:



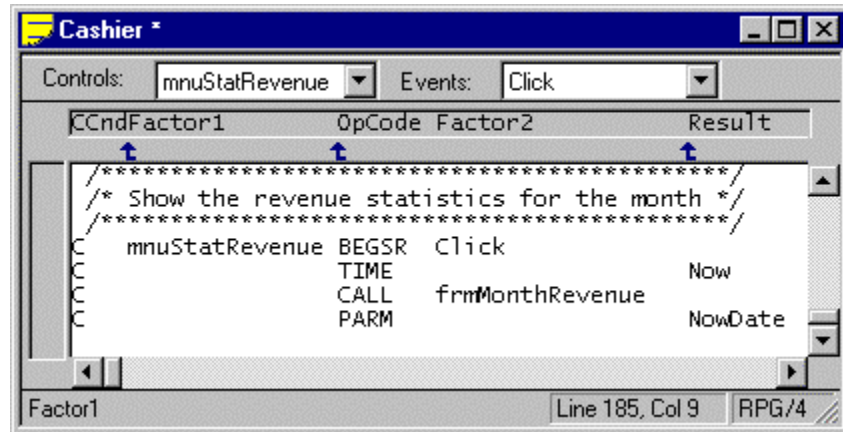
We have left the form's **BorderStyle** as sizeable, so the user will be able to stretch the form by pulling one of its borders. If the user stretches the form, he will probably want the graph to grow/shrink accordingly.

9. Add the following event handler to grow/shrink the graph accordingly.

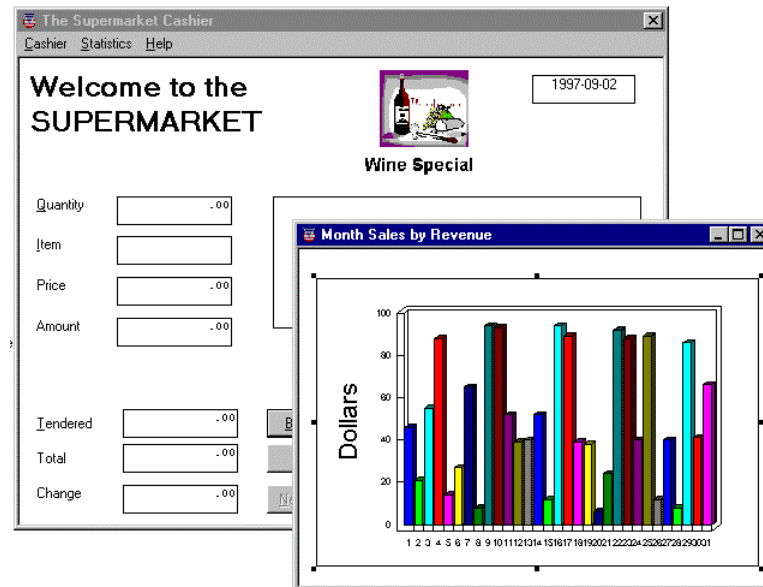


The variable name ***ThisForm** refers to the form that the code belongs to.

10. Save the **frmMonthRevenue** as **MonthRevenue.vrf**.
11. Finally, let's enter the code to respond to the click of **mnuStatRevenue** in **frmCashier** by calling **frmMonthRevenue** like the following:



12. Run the program and request the Revenue Statistics. You should see the following:



To close the graph form, click on the **Close** button  in the top-right corner.

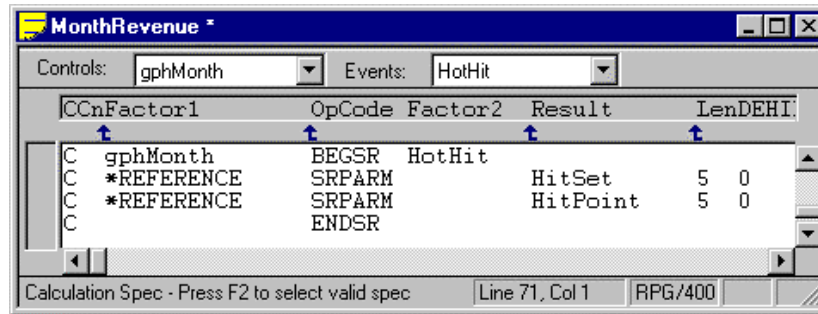
Receiving Parameters in a Subroutine

There are some events that can make some information available to the subroutine handling them. This information is passed in special subroutine parameters. The AVR op code **SRPARM** (SubRoutine PARaMeter), is similar to the traditional PARM, except that is only available to the subroutines where it has been defined.

When the editor generates the subroutine, if the event provides any information to the handler besides the BEGSR and ENDSR, it will generate one SRPARM of the proper type for each parameter.

The Graph control provides one of these events, called **HotHit**. HotHit is triggered whenever the user clicks on one of the data bars in the graph. When your event handler gets called, two parameters are passed to it, the **Set** and the **Point** within the set that was clicked. Our application is only using one set, the sales of the current month, and up to 31 points, one per day. In order for the Graph control to trigger the HotHit event, the **Hot** property must be set to 1 - On.

- ✓ Request the editor to start an event handler for the **HotHit** event for the **gphMonth** control. The following will display.



Notice that the editor has generated **two** SRPARM lines, one for HitSet and one for HitPoint. In this case, both parameters are of the same numeric type.

Events can pass parameters to the handler in one of two ways, by *Reference* or by *Value*.

Factor 1 of SRPARM specifies which convention is being used to pass parameters: *REFERENCE or *VALUE.

- When a parameter is passed by **Reference**, any changes done to it by the handler subroutine will be **'seen'** by the control. This is the way in which traditional RPG passes parameters between programs.
- When a parameter is passed by **Value**, the subroutine gets a copy of the value of the parameter if there are any changes that only affect the subroutine and not the control.

Overlapping Windows

We have already seen one of the new capabilities introduced with Graphical User Interfaces: **Overlapping Windows**.

Overlapping windows allow the programmer to maintain many different contexts available to the user by providing multiple windows which partially cover each other on the screen. When the user clicks any area of a particular window, it comes to the 'foreground' showing all the information it contains. The Windows Operating System handles all necessary 'repainting' to achieve this effect.

In our application, we can take advantage of this capability to allow the user to display multiple graphs showing detailed information about the sales of particular days.

✓ Let's Display Multiple Graphs

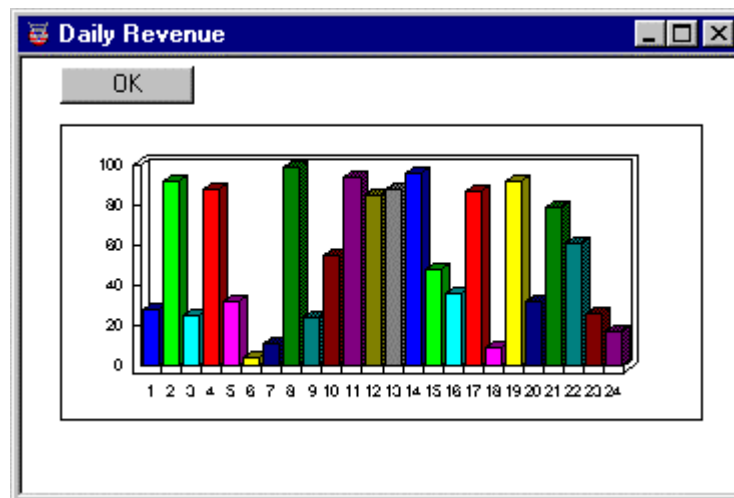
1. Create a new form called **frmDayRevenue**.

This form will contain the logic to display, by hour, the sales for a particular day. Then, whenever a user clicks on a day on the frmMonthRevenue graph, we will respond by creating a new instance of the daily sales graph **frmDayRevenue** and showing it on the screen using the **Modeless** technique. Then, if the user clicks on a **different** day of the month, without closing the first day, he will get a **second** instance of frmDayRevenue showing that second day, and so on.



Graph Control

2. Add a Graph called **gphDay** to the new form, with **NumPoints** set to 24 and **True3D** set to 0.
3. Add an **OK** button named **btnOK**.
4. The form **frmDayRevenue** should now look like the following:



We will use a new op code called **NEWFORM** to instantiate a copy of the **frmDayRevenue** every time we handle the HotHit event. **NEWFORM** takes two parameters in Factor 2; the type of form to instantiate, and where to 'store' the newly-created form.

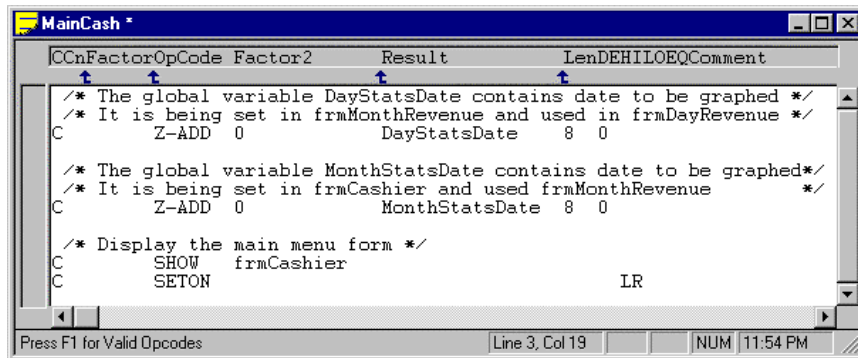
When you add a form to your project, AVR defines a new **type** for that form and a **variable** of that type with the same name. We can use this type and variable in our handler.

Since we want to display the daily graph in a modeless way, we cannot use the op code `CALL`, because it operates in a *modal* fashion. We will have to use the op code `SHOW`. Unfortunately, `SHOW` cannot take parameters, so we will have to pass to the newly instantiated form the date of the day to graph in a global variable. Remember, variables defined in the Main program are *global* to the whole application.

We will define the variables `DayStatsDate` and `MonthStatsDate` in the Procedural Program `MainCash` so that both `frmMonthRevenue` and `frmDayRevenue` can see it.

There is one more issue to address. You can not show a modeless form from a modal form. When an application displays a modal form, it enters into a modal mode, allowing you to **only** show other modal forms.

5. Enter the new code for `MainCash`, as shown below:

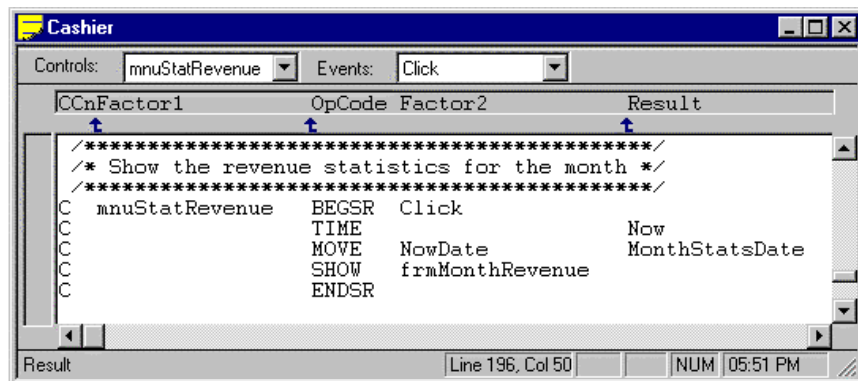


```

MainCash *
|CCnFactorOpCode Factor2      Result      LenDEHILOEQComment
|-----|-----|-----|-----|
|/* The global variable DayStatsDate contains date to be graphed */
|/* It is being set in frmMonthRevenue and used in frmDayRevenue */
C      Z-ADD 0      DayStatsDate 8 0
|
|/* The global variable MonthStatsDate contains date to be graphed*/
|/* It is being set in frmCashier and used frmMonthRevenue */
C      Z-ADD 0      MonthStatsDate 8 0
|
|/* Display the main menu form */
C      SHOW frmCashier
C      SETON
|
|-----|-----|-----|-----|
|Press F1 for Valid OpCodes      Line 3, Col 19      NUM 11:54 PM

```

6. Replace the `CALL` from `frmCashier` to `frmMonthRevenue` with a `SHOW`, and pass the month date to display on the global variable `MonthStatsDate`. Also, remove or comment out the `PLIST` and `PARM` from `frmMonthRevenue`.

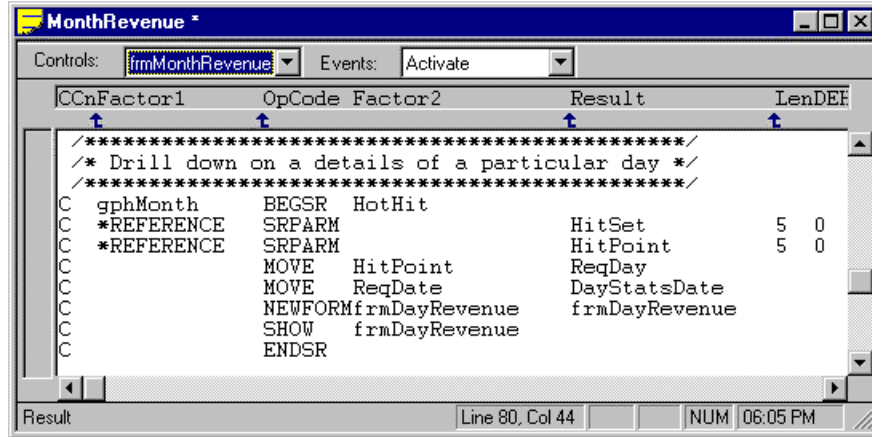


```

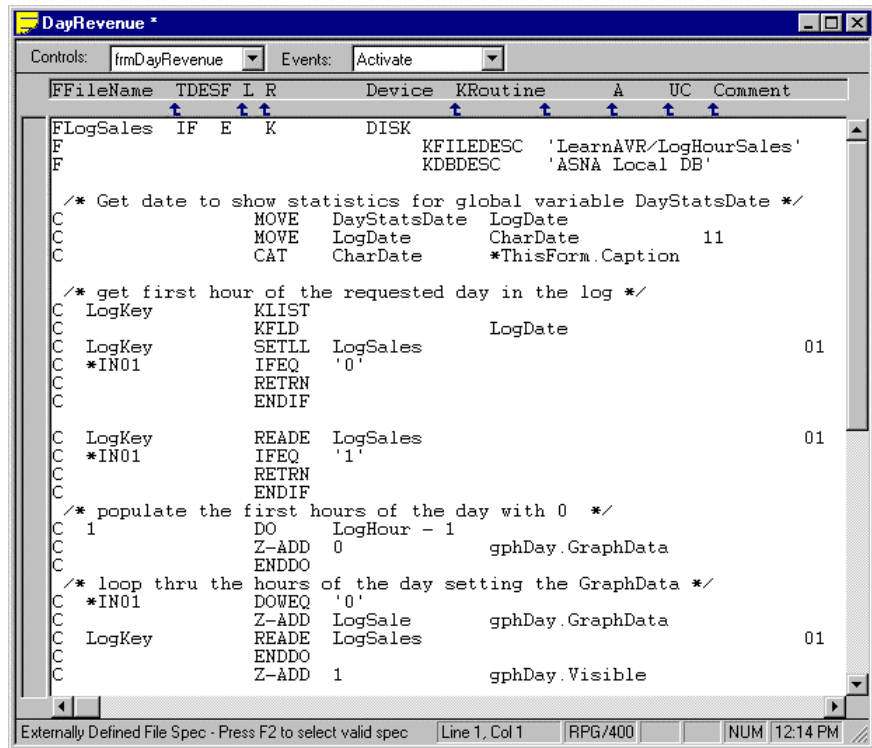
Cashier
Controls: mnuStatRevenue Events: Click
|CCnFactor1      OpCode Factor2      Result
|-----|-----|-----|-----|
|*****
|/* Show the revenue statistics for the month */
|*****
C      mnuStatRevenue BEGSR Click
C      TIME
C      MOVE NowDate      MonthStatsDate
C      SHOW frmMonthRevenue
C      ENDSR
|
|-----|-----|-----|-----|
|Result      Line 196, Col 50      NUM 05:51 PM

```

7. Enter the new code for the `HotHit` handler of `frmMonthRevenue`, as shown below:



8. Enter the code for **frmDayRevenue**, as shown below:



9. Add a resize handler for **frmDayRevenue** and an OK button with its **Default** property set to True and its **TabIndex** set to 0.

The screenshot shows the DayRevenue form editor with the following code:

```

C CndFactor1      OpCode Factor2      Result      LenDEHIL
C
C /******  

C /* Resize the graph with the form */  

C /******  

C frmDayRevenue      BEGSR      Resize
C
C /* Don't let the form get to small */  

C *ThisForm.Width      IFLT      5000
C                       Z-ADD      5000      *ThisForm.Width
C                       ENDIF
C *ThisForm.Height      IFLT      3000
C                       Z-ADD      3000      *ThisForm.Height
C                       ENDIF
C
C /* Size the graph to fill the form */  

C *ThisForm.Height      SUB      1000      gphDay.Height
C *ThisForm.Width      SUB      800      gphDay.Width
C                       ENDSR
C
C /******  

C /* Close the form when OK gets Clicked */  


C /******  

C btnOK              BEGSR      Click
C                       SETON
C                       RETRN
C                       ENDSR
  
```

At the bottom of the editor, the status bar shows: Conditional Indicators | Line 57, Col 2 | RPG/400 | NUM

10. Save your form as **DayRevenue.vrf**.

Step 8 Summary

| To | Do This | Button/Keys |
|--|--|---|
| Create a graph on a form | Select the Graph control from the control palette and draw onto the form. |  |
| Store graph values | Use the ThisSet and ThisPoint properties. | |
| Automatically increment the ThisSet and ThisPoint values as you enter data | Use the AutoInc property. | |
| Pass event subroutine parameters | Use the SRPARM op code. | |
| Request the graph control to trigger the HotHit event | Set the Hot property to 1 = On. | |
| Pass a parameter by reference | Enter *REFERENCE in the SRPARM op code. | |
| Pass a parameter by value | Enter *VALUE in the SRPARM op code. | |
| Display multiple graphs | Use the NEWFORM op code. | |
| Show a form in a modeless fashion | Use the SHOW op code. | |
| Show a form in a modal fashion | Use the CALL op code. | |

More Information

To find more information on the topics introduced in Step 8, refer to the following help file topics.

Help File Topics:

- Event Subroutine parameters
- Graph control
- SRPARM
- SHOW
- CALL
- *REFERENCE
- *VALUE



Printing

What you will learn in Step 9:

- How to define a print file.
- How to print within RPG.

Approximate Time to Complete Step 9:

1 hour, 15 minutes.

What the Application will look like after completing Step 9:

| | | |
|--|-----------------------|-----------------|
| The Supermarket | | |
| 100 Main Street | | |
| Anytown, USA | | |
| Date : 06/27/97 | | Time : 12:13:04 |
| 1.00 | Maxwell Coffee 12 oz. | 4.32 |
| 7.35 | Watermelon | 2.34 |
| 6.00 | Lightbulb | 14.04 |
| 2.5 | Pistachios | 3.07 |
| 1.00 | Orange Jam | 0.74 |
| | | 25.38 |
| Thank you for buying at The Supermarket | | |

In this step, you are going to add to the Cashier application a printed receipt for the client. Printing in AVR is similar to printing in traditional RPG by using **Externally Defined Print Files**.

Acceler8DB, the Database Management System supporting AVR, features Print Files as multiple-formatted files. Each format of the print file describes a **portion** of a page. When a report is to be generated, writing these records causes Windows to 'paint' sections of each page on the report.

To create the layout of each format on a print file, you will use the **Acceler8DB Print File Editor**.

Starting and Using Acceler8DB Print File Editor

- ✓ Start Acceler8DB Print File Editor by performing one of the following:
 - Select **Start - Programs - ASNA Product Suite - Acceler8DB 5.0 - Acceler8DB Print File Editor**.
 - Select **Acceler8DB Print File Editor** from the **Tools** menu in Visual RPG IDE.
 - Select **Create Print File** from the **Definition** menu in Acceler8DB Database Manager.


The Print File Editor will open, as shown below.




The Print File Editor serves as a window in which you edit Print Files by adding formats, fields, and setting properties for each item added.

The Print File Editor window contains a Title Bar, Menu Option Bar, Toolbar, an Untitled format, a Field Palette, and a Status Bar.


To Create a New Print File

- Select **New** from the File Menu, press the **Ctrl+N** Keys, or select  from the ToolBar.


To Open an Existing Print File

- Select **Open** from the File Menu, press the **Ctrl+O** Keys, or select  from the ToolBar.


To Save a Print File

- Select **Save** or **Save As** from the File Menu, press the **Ctrl+S** Keys, or select  from the ToolBar.

To Add Formats to Selected Print File

- Select **Insert New** from the Format Menu, press the **Alt+Ins** Keys, or select  from the ToolBar.

To Add Fields to Selected Format

- Select **Insert New** from the Field Menu, press the **Ins** Key, select  from the ToolBar, or select desired field from Field Palette.

Using Print Editor Window

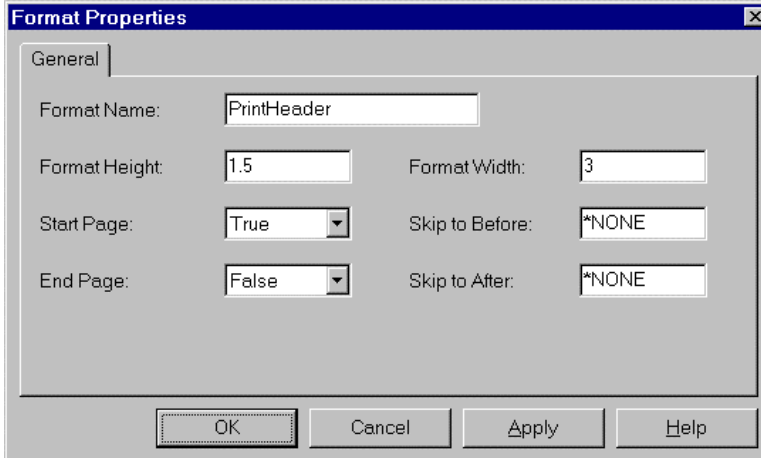
- The Print Editor Window can be closed, minimized, maximized, sized or moved.

 Let's Create a Print File

Defining a print file using Acceler8DB Print File Editor is very similar to defining a form using AVR's IDE. You **add fields** to a format and then **set properties** for the format and fields.

1. Select **Properties...** from the **Format** menu, or press **F4**.

Set the properties of the format according to the following figure:



| Format Properties | | | |
|--|-------------|-----------------|-------|
| General | | | |
| Format Name: | PrintHeader | | |
| Format Height: | 1.5 | Format Width: | 3 |
| Start Page: | True | Skip to Before: | *NONE |
| End Page: | False | Skip to After: | *NONE |
| <input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Apply"/> <input type="button" value="Help"/> | | | |

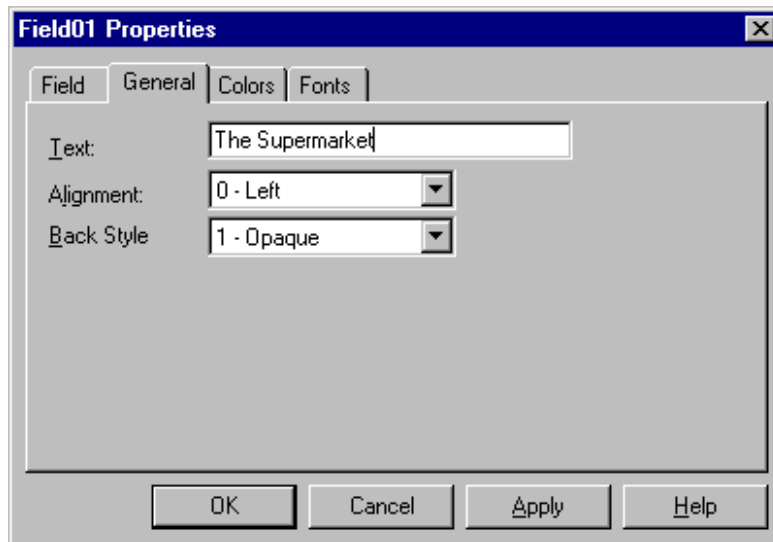
We have requested that the format be named **PrintHeader**, measure 1.5 by 3 inches and that every time it is written, it should force the start of a new page. After you click **OK**, notice how the Editor depicts the format as a white area of the requested size.

To add a field to a format, click on the field control you want on the palette and 'draw' it in the position you want it to appear relative to the format.

2. Add a **Label** field to the format.
3. Once the field is drawn on the format, **double-click** it, or press **F4** while its selected to display a dialog to set its properties.

For fields, the properties dialog is divided into property pages. As shown in the next figure, the Print Label Field has four pages: Field, General, Colors and Fonts.

4. Click the **General** tab and set the **Text** to **The Supermarket**.



5. Select the **Fonts** Tab and set the **Font** to Arial, **Size** to 18 and **Style** to Bold. (*You may need to resize the Label field so the text fits*).
6. Make the following additions to the **PrintHeader** format for the Supermarket receipt, **as shown on the next page**.

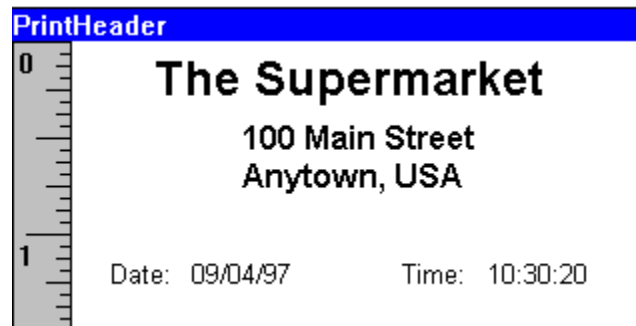


4 Label fields: 100 Main Street
 Anytown, USA
 Date:
 Time:



2 System Value fields: Type - Local date
 Type - Local time

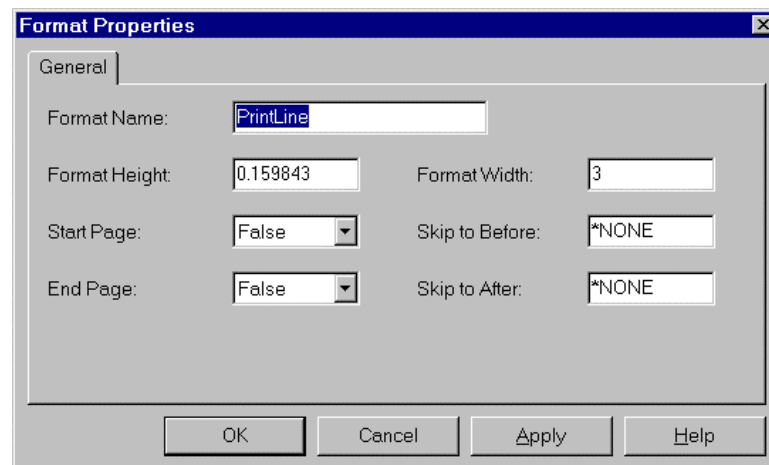
Adjust the font size so that the fields fit comfortably on the format, as described in step 5 above. (Pressing **F4** with a field selected will allow you to change that fields' properties).



In the following steps, you will add two more formats, one for each product bought and one to print the total.

See **page 117** for the completed Print File.

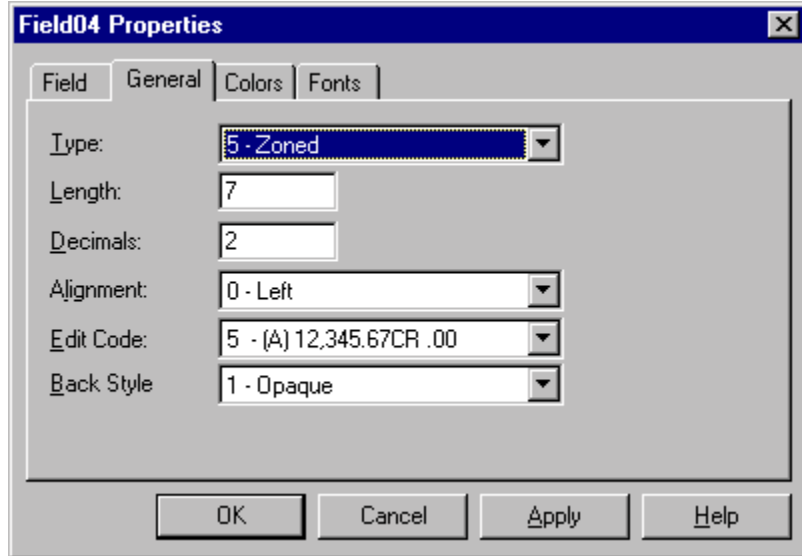
7. Insert a new format by selecting **Insert New** from the **Format** menu. The following is its property page, containing the format name and size.



Next, we will add 3 **Data** fields to the format. A Data field has properties that allow the setting of Data types and Edit codes. It also has a **Name** property, which is the name you will use to reference the field in your RPG program. When you write out a format, the values of the RPG fields that match the names to the Data Fields will be sent as part of the write to the print file.



8. Select Data Field from the field palette and draw 3 data fields side by side on the format. The fields in the PrintLine format are: **Quantity**, **ItemDesc** and **Amount**.
9. Select the Data Field on the left, and press **F4**. The Print Data Field dialog will display. In the Field tab, enter the name **Quantity**.
10. Click the **General** tab and enter the following as shown.



11. Click the **Fonts** Tab and select **Courier New** as the font type with a size of **7.5**.
12. Add the properties for the other two Data Fields, as listed below.

| Name: | ItemDesc | Amount |
|-------------------|-------------------|------------------------|
| Type: | 0 - Character | 5 - Zoned |
| Length: | 20 | 7 |
| Decimals: | 0 | 2 |
| Alignment: | 0 - Left | 0 - Left |
| Edit Code: | 0 (None) | 5 - (A) 12,345,67CR.00 |
| Fonts Tab: | Courier New - 7.5 | Courier New - 7.5 |

When the PrintLine format is complete, it should look like the following:

```
PrintLine
|n_| 99,999.99CR  XXXXXXXXXXXXXXXXXXXXXXXX  99,999.99CR
```

Next, add the **PrintTotal** format by following the steps below.

13. Add a new format by selecting **Insert New** from the **Format** menu, or press the **Alt+Insert** keys.
14. Press **F4** and set the following format properties:

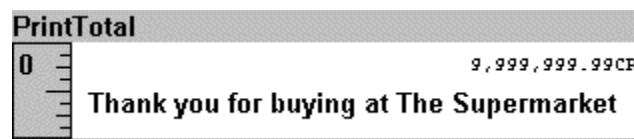
| | |
|-----------------------|------------|
| Format Name: | PrintTotal |
| Format Height: | .5 |
| Format Width: | 3 |

15. Add a **Data Field** in the upper right corner of the format.
16. Press **F4** and set the following properties:

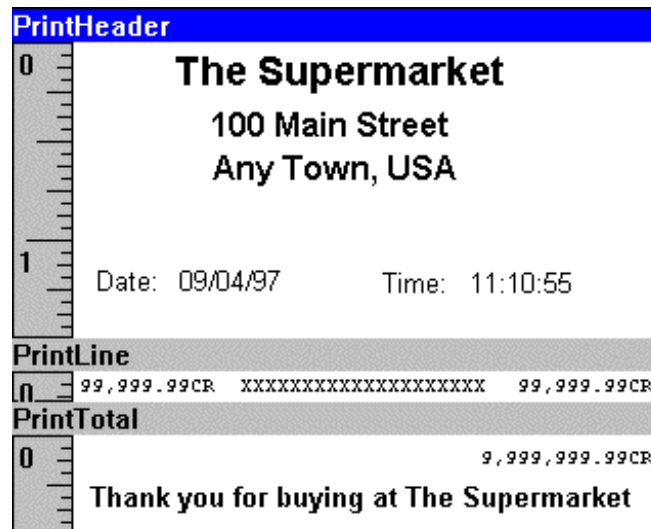
| | |
|-------------------|------------------------|
| Name: | Total |
| Type: | 5 - Zoned |
| Length: | 9 |
| Decimals: | 2 |
| Alignment: | 0 - Left |
| Edit Code: | 5 - (A) 12,345,67CR.00 |
| Fonts Tab: | Courier New - 7.5 |

17. Add a **Label** to the format. Press **F4** and set the **Text** to **Thank you for buying at The Supermarket**, and the **Font** to MS Sans Serif, **bold**, size 10.

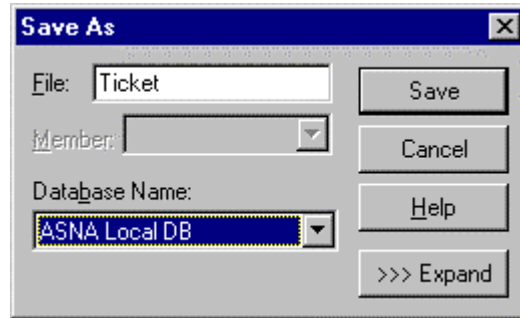
When the PrintTotal format is complete, it should look like the following:



The following is the completed Print File:



18. When you are satisfied with the look of the file, select **Save As** from the **File** menu.
19. The Save As dialog will display, prompting you for the file name and location. Select the Database '**ASNA Local DB**', directory '**LearnAVR**' and name the file '**Ticket**', as shown below.



Printing in RPG

Now that you have created the print file, you can use it in your RPG program.

Let's add the Print File to our AVR Program

1. Add an F-Spec to the **frmCashier** code, as shown below:

```
FTicket  O   E           PRINTER
F                KFILEDESC  'LearnAVR/Ticket'
F                KDBDESC    'ASNA Local DB'
```

This will implicitly open the file when the form gets loaded.

2. Add the following **bolded** line to the **main C-Specs**:

```
...
C                MOVE   LogDate    ioCurrentDate
C                WRITE PrintHeader
```

3. Add the following **bolded** line to the **btnBuy-Click** subroutine:

```

C      btnBuy      BEGSR  Click
...
/* Print line item */
C          WRITE  PrintLine
/* Clear input fields */
C          Z-ADD  1          ioQuantity
C          EXSR   *BLANKS    ioItem
C          ENDSR

```

4. Add the following **bolded** line to the **btnPay-Click** subroutine:

```

C      btnPay      BEGSR  Click
C          WRITE  PrintTotal
C          Z-ADD  ioTendered  Tendered    9  2
...

```

5. Add the following **bolded** line to the **btnNew-Click** subroutine:

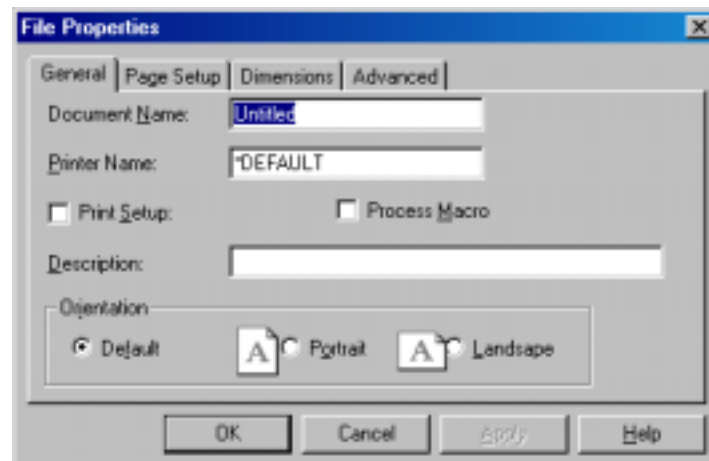
```

C      btnNew      BEGSR  Click
C          MOVE    *OFF
          btnNew.Enabled
C          MOVE    *ON
          btnBuy.Enabled
C          WRITE  PrintHeader
C          EVAL   lstReceipt.ClearObj()
...

```

The print file controls what in Windows is known as a **document**. A document can have a Name which displays when looking at the spooler in Windows. Features like the document name, as well as the name of the printer to send the output to are controlled via **properties** in the file.

6. Invoke the property dialog in the Print File Editor by selecting **Properties** from the **File** menu. The following dialog box will display.











If you enable **Print Setup** by clicking in the box and creating a check mark, then when the file is opened by your RPG program, the regular Windows dialog to select the printer, paper size etc. will be presented to the user.

In Windows, the document is spooled by the operating system until it is completed, then it is sent to the printer.

Each document may have multiple pages. If you want to force the document to be sent to the printer *before* your program ends, you have to close the print file by hand.

If your program needs to continue printing, open it again. You open and close a print file using the normal RPG op codes OPEN and CLOSE.

Step 9 Summary

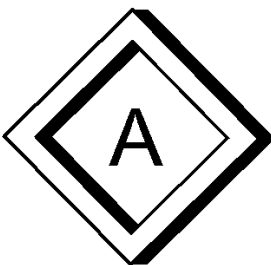
| To | Do This | Button/Keys |
|--|--|---|
| Start Acceler8DB Print File Editor | <ul style="list-style-type: none"> • Select Start - ASNA Product Suite - Acceler8DB 5.0 - Acceler8DB Print File Editor. • Select Acceler8DB from the Tools menu in the Visual RPG IDE. • Select Creat<u>e</u> Print File from the Definition Menu in Acceler8DB Database Manager. | |
| Create a new print file | Select New from the File menu. |  Ctrl+N |
| Open an existing print file | Select Open from the File menu. |  Ctrl+O |
| Save a print file | Select Save or Save As from the File menu. |  Ctrl+S |
| Add formats to the selected print file | Select Insert New from the Format menu. |  Alt+Ins |
| Add fields to the selected format | Select Insert New from the Field menu. |  Ins |
| Add a label field to a format | Select Label from the field palette. |  |
| Add a data field to a format | Select Data Field from the field palette. |  |
| Add a system value to a format | Select System Value from the field palette. |  |

More Information

To find more information on the topics introduced in Step 9, refer to the following help file topics.

Help File Topics:

- Setting fonts
- Inserting a new format
- Data field
- Data field properties
- Label field
- Label field properties
- System Value
- System Value properties
- Format properties, setting
- Field properties, setting
- Saving a file
- F-Spec
- OPEN
- CLOSE



Converting AS/400 Applications

Even though Visual RPG provides tools to import RPG/400 code from the AS/400 and to generate forms from Display Files, it is highly recommended that you become familiar with the new paradigm of event driven programming before you make use of these facilities.

AVR is NOT a migration tool. These importing facilities are more of an aid in converting your application.

AS/400 Display File (Import Menu)

The **AS/400 Display File** menu option allows you to import a Display file (.DDS) from an AS/400 to a Visual RPG form.

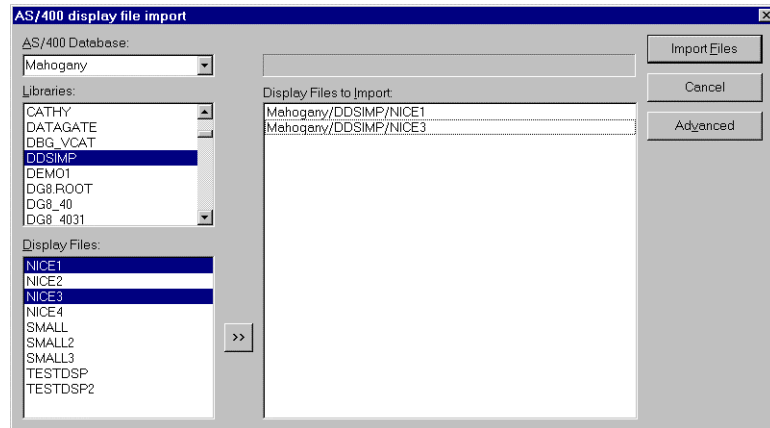
You may be importing a program just to get some general subroutines that are already debugged, or maybe the code you have in a program is just the skeleton of what you want to do in the Visual RPG form program.

The import facility does require DataGate/400 to access data from the AS/400.

Visual RPG will automatically convert AS/400 keywords into associated properties, controls, etc. Refer to Importing and Converting AS/400 Display Files for a listing of AS/400 keywords that are converted by Visual RPG.

To Import Existing AS/400 Display Files to be utilized with Visual RPG

1. Select **AS/400 Display File** from the Import menu.
2. Enter or select the following information to the Import RPG/400 source member dialog box.



AS/400 Database:

The AS/400's currently on the system will display. Click on the arrow to the right to display all the AS/400's currently on the system and select the desired AS/400 database.


Libraries:

The libraries available on the selected AS/400 database will display. Double-click the desired library.

Display Files:

The display files (.DDS) available in the selected library will display. Select the desired display file. Multiple display files can be selected by just clicking on them. (Clicking on an already selected object will unselect it).

Double Arrow Button:

With the desired display file(s) selected, select the  button. The selected files will display in the **Display Files to Import** column on the right.

Display Files to Import:

The selected file(s) in the Display Files section will display once the double arrow button is clicked. These are the files that will import. You can remove one if desired by clicking on them and pressing the Delete Key.

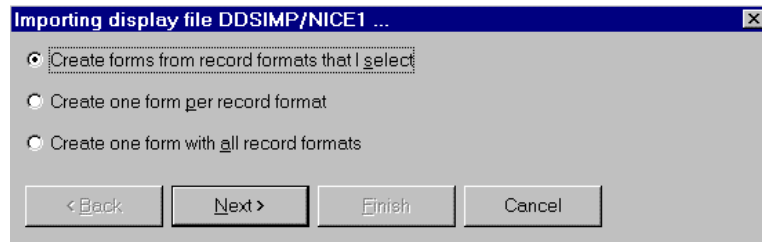
Import Files Button:

Select **Import Files** to import the files displayed in the Display Files to Import column.

Cancel Button:

Select **Cancel** to search for a database containing the file to Import.

3. Once the **Import Files** button is selected, the following dialog box will display to indicate how many formats and forms will be selected/created.



Create forms from record formats that I select

This option is the default, and allows the user to create a form for each record format selected.

Create one form per record format

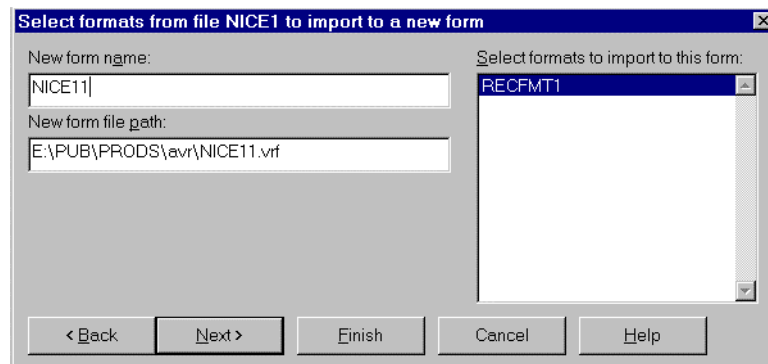
Creates one form for each record format contained in the display file.

Create one form with all record formats

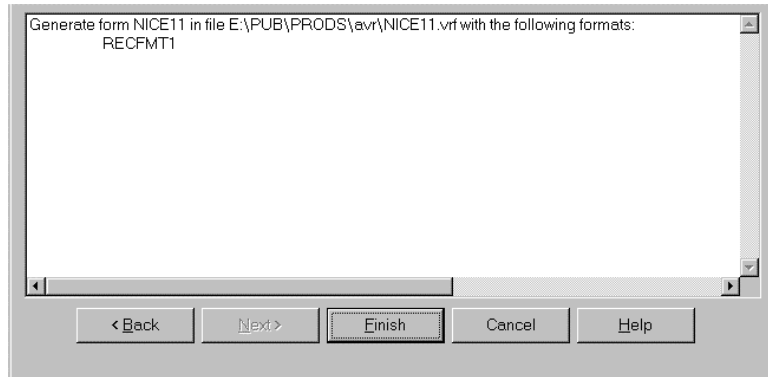
Creates one form that will contain information from all record formats contained in the display file.

Select the desired button to go to the previous screen, next screen, or finish importing the display file(s).

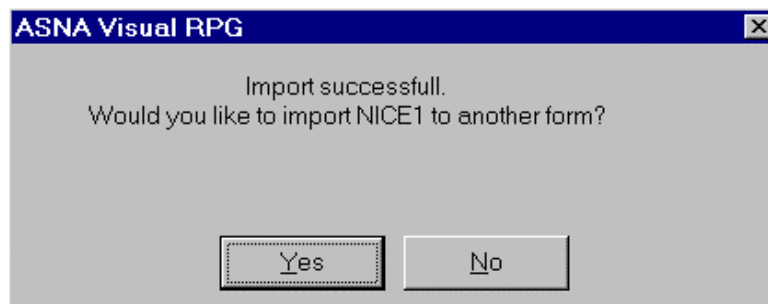
- The following screen will display allowing you to specify or change the name and path of the form to create with the formats highlighted on the right.



- Select the **Next>** or **Finish** button when you are ready to import the display files. When **Next>** is selected, the following screen will appear, displaying which form(s) will be created in what path with what formats.



6. Select **Finish** when you are ready to import the Display file. The display file(s) will be import into the desired form(s). The following message will display, prompting if you would like to import the display file to another form.



7. If **Yes** is selected, you will continue with Step 3.
8. When you are done importing to a form, select Import AS/400 Source Member from the **Import** menu to import the code.

Importing and Converting AS/400 Display Files

Dimensions:

- Form **width** and **height** is found by multiplying display size (either 80x24 or 132x27) by a user-defined font width and height entered in the **Import** Tab of the Tools-Options menu.
- The control's **left** and **top** is found by multiplying position and line, respectively, by the user-defined font width and height.
- Menu items are created from Command Function (**CF**) and Command Attention (**CA**) keys (record or field level) with the Command Key as a shortcut, and menu name created by parsing labels and looking for "Fnn = Something".

IOFields:

- IOFields are created from fields with usage type O, I, or B. (Fields of type H, M, or P are ignored).
- IOFields are uppercase unless the **CHECK(LC)** is present.
- The **CONFLD(n)** keyword is imported as a multi-line IOField with **n** characters in each row and occupying (Length/**n**) number of columns.
- If the **DFT** or **DFTVAL** keywords are present, they are used as the IOField's default text.
- If the **CHECK(FE)** keyword is specified, the IOField's FieldAutoAdvance is set to False.
- NumericLength and Decimals are set for IOFields of type **F, S, N, M, I, D,** or **Y**. MaxLength is set for fields of type **A** or **X**.

Labels:

- Labels are created from constant fields and given a generic name of **Label#**.
- **MSGCON** keyword is imported as a Label with the associated message as its Caption.

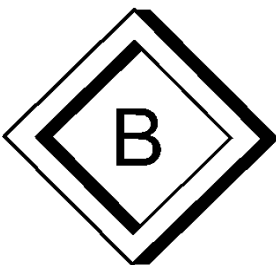
Subfiles:

- Subfiles are created from files with the **SFL** keywords from the file with the **SFLCTL** keyword.
- Subfile **top** is calculated from the first (in row-column order) field's line in the record with the **SFL** keyword.
- Subfile **width** is calculated from the longest field's width in the record with the **SFL** keyword.
- Subfile **height** is given the generic value of 2500 (twips).
- Subfile **left** is calculated from the first (in row-column order) field's position in the record with the **SFL** keyword.

Miscellaneous:

- A generic tab order is assigned in the order that they were read from the DDS file.
- Records with the **WINDOW** keyword specified are ignored.
- The field's **COLOR** is imported as the control's BackColor.

This Page Intentionally Left Blank



Caviar Free Format

AVR allows you to program in either RPG Style Fixed Format or Caviar Free Format. Throughout this tutorial, all examples have been given in RPG Style Fixed Format. In this appendix, you will get a little taste of Caviar Free Format.

The Caviar syntax of AVR is very similar to that of Control Languages. It consists of a command, followed by Keyworded parameters. Parameters can also be given positionally. If a parameter is not given, its default value is used.

Commands can be split into multiple lines by providing the continuation line character (+) as the **last** character of the line.

The following is a listing of the **frmMonthRevenue** code written in Free Format RPG.

Listing for frmMonthRevenue

```
DCLDISKFILE LogSales *UPDATE DSG(*FULL) ORG(*INDEXED) +
  DBDESC('ASNA Local DB') FILEDESC('LearnAVR/LogHourSales') +
  ADDREC (*YES)
```

```
DCLDS LEN(8)
  DCLDSFLD ReqDate START(1) LEN(8, 0)
  DCLDSFLD ReqYear START(1) LEN(4, 0)
  DCLDSFLD ReqMonth START(5) LEN(2, 0)
  DCLDSFLD ReqDay START(7) LEN(2, 0)
```

```
DCLFLD CurrentDay *ZONED LEN(2, 0)
DCLFLD DaySales *ZONED LEN(11, 2)
```

```
ReqDate = MonthStatsDate
CurrentDay = 1
ReqDay = CurrentDay
LogDate = ReqDate
DaySales = 0
```

```

/* read first day of the requested month in the log */
SETLL FILE(LogSales) KEY(LogDate)
READ FROM(LogSales) EOF(*IN01)
EXTRACT F2(LogDate) F2FMT(*M) RESULT(LogMonth) LEN(2, 0)
EXTRACT F2(LogDate) F2FMT(*D) RESULT(LogDay) LEN(2, 0)
IF (*IN01 == '1' *OR LogMonth <> ReqMonth)
    MSGBOX MSG('No sales in this month')
    SETON *INLR
    RETURN
ENDIF

/* populate the first days of the month with 0 */
/* till we get to the first day with data */
DO FROMVAL(1) TOVAL(LogDay - 1)
    gphMonth.GraphData = 0
ENDDO
CurrentDay = LogDay

/* loop through the days of month computing the GraphData */
DOWHILE (LogMonth = ReqMonth)
    IF (LogDay <> CurrentDay)
        gphMonth.GraphData = DaySales
        DaySales = 0
        CurrentDay = CurrentDay + 1
    ENDIF
    DaySales = DaySales + LogSale
    READ FROM(LogSales) EOF(*IN01)
    IF (*IN01)
        LEAVE
    ENDIF
    EXTRACT F2(LogDate) F2FMT(*M) RESULT(LogMonth) LEN(2, 0)
    EXTRACT F2(LogDate) F2FMT(*D) RESULT(LogDay) LEN(2, 0)
ENDDO
gphMonth.GraphData = DaySales

/*****/
/* Resize the graph with the form */
/*****/
BEGSR NAME(frmMonthRevenue) EVENT(Resize)

/* Don't let the form get too small */
IF (*ThisForm.Width < 5000)
    *ThisForm.Width = 5000
ENDIF
IF (*ThisForm.Height < 5000)
    *ThisForm.Height = 5000
ENDIF

/* Size the graph to fill the form */
gphMonth.Height = *ThisForm.Height - 1600
gphMonth.Width = *ThisForm.Width - 600
ENDSR

```

```

/*****
/* Drill down on a details of a particular day */
*****/

BEGSR  NAME(gphMonth) EVENT(HotHit)
DCLSRPARG HitSet  LEN(5, 0) BY(*REFERENCE)
DCLSRPARG HitPoint LEN(5, 0) BY(*REFERENCE)
      ReqDay      = HitPoint
      DayStatsDate = ReqDate
      NEWFORM FORM(frmHourRevenue) FORMTYPE(frmHourRevenue)
      SHOWfrmHourRevenue
ENDSR
```

This Page Intentionally Left Blank

Index

*

*Reference, 105

*Value, 105

▪

.VRM, 84

.VRM file, 28

.VRP, 84

A

About box

displaying, 77

Acceler8-DB

database management system, 51

Acceler8-DB File Viewer

browsing, 51

Access keys

defining, 38

Accessing

DB2/400 files, 51

global variables, 85

menu items, 71

variables, 85

Add Files To Project

file menu, 29

Adding

bitmaps, 64

comments, 39

c-spec, 25

event subroutine, 36

field to a format, 114

files to project, 29

form, 77

f-spec, 100

graphics, 64

items, 61

new item, 59, 66, 81

new row, 59, 66, 81

procedural program, 85

shortcut key, 72

subfile control, 87

subfile record, 96

AddItem method, 59, 66, 81

Alignment property, 41

Ampersand

adding, 38

Appendix b

free format, 129

Application

saving, 29

Array property, 95

AutoExtend property, 94, 95

AutoInc property, 102

B

Background

windows desktop, 5

Beginning

subroutine, 36

BEGSR, 36

Bitmaps

adding, 64

Blank form, 10

Blank-spec

comments, 39

BorderStyle settings

fixed dialog, 14

fixed single, 14

fixed tool window, 14

none, 14

sizable tool window, 14

sizeable, 14

Bottom event, 96

Browse property, 94

Browsing

data, 51

fields, 94

B-Spec

data structure subfield, 26

- Button
 - close, 14
 - maximize, 14
 - minimize, 14
 - run, 6
- C**
- Calc Spec, 25, 35
- CALL, 107
- Caption
 - changing, 11
 - menu editor, 71
- Caption property, 12
- CAT, 65
- CellData property, 96
- CHAIN, 87, 92
- Change event
 - coding, 58
- Changing
 - caption, 11
 - focus order, 62
 - form properties, 15
 - form's title bar, 11
 - name property, 29
 - properties at run-time, 24
 - property at run-time, 24
 - subfile record, 96
- Character fields
 - display password, 41
 - force uppercase characters, 41
 - limit characters, 41
- Checking
 - menu items, 72
- Clearing
 - data, 59
- ClearObj method, 59, 94
- Click event, 38
 - coding, 59
- Close
 - button, 14
- CLOSE, 120
- Closing
 - print file, 120
- Code
 - adding comments, 39
 - referring to a property, 26
- Coding
 - change event, 58
 - click event, 59, 76
 - continuation line, 53
 - e-spec, 65
 - f-spec, 52, 53, 118
 - k-spec, 53
 - message box, 53
 - print file, 118, 119
- Columns
 - spec format, 25
 - stretching, 27
- Command button
 - default property, 22
- Comments
 - adding, 39
- Compiling
 - program, 13
- Components
 - IDE, 4
- Concatenate, 65
- Continuation line
 - coding, 53
- Continuation line character
 - free format editor, 129
- Control
 - copying, 46
 - deleting, 46
 - disabling, 58
 - enabling, 58
 - events, 34
 - focus, 62
 - moving, 59
 - refresh, 59
 - repaint, 59
 - resizing, 46
 - selecting, 46
 - setting focus, 59, 66
 - subfile, 87
 - type, 28
- Control Box
 - editor, 36
- Control buttons, 20
- Control icons, 20
- Control palette, 20
- Control properties
 - displaying in help, 24
- Controlbox property, 14
- Controlling end of program, 85
- Controls
 - selecting multiple, 46
- Copying controls, 46
- Creating
 - access key, 71
 - menu separator, 71

- menus, 70
- new project, 10
- print file, 112
- C-Specs, 25
 - calculation, 26
 - spec type, 35
- D**
- Data
 - browsing, 51
 - clearing, 59
- Data field, 115
- Data Structure Field specification, 60
- Data Structure specification, 60
- Database Management System
 - Acceler8-DB, 51
- Data-entry mode
 - subfile, 94
- DataGate/400
 - providing access to DB2/400 files, 51
- Date
 - displaying, 101
- DB2/400 files
 - accessing, 51
- DBDESC, 100
 - database description, 52
- Debug menu
 - run, 13, 16
- Decimals property, 40
- Declaring
 - e-spec, 65
- Default form
 - form1, 11
- Default property, 22, 38
- Defining
 - access keys, 38
 - number of digits, 40
 - numeric digits, 40
- Deleting
 - control, 46
 - subfile records, 95
- Design mode, 12, 13
- Disabling
 - control, 58
- Displaying
 - about box, 77
 - current time and date, 101
 - LearnAVR folder, 6
 - multiple lines of text, 41
 - multiple windows, 105
 - number of list items, 61
 - password, 41
 - uppercase characters, 41
- Docking
 - field palette, 112
- Document
 - printing, 120
 - spooling, 120
- Double-clicking
 - title bar, 5
- E**
- EditCode property, 22, 40
- Editor
 - adding comments, 39
 - control box, 36
 - event procedure box, 36
 - header bar, 36
- EditWord property, 41
- Enabling
 - control, 58
 - menu items, 72
- End of program
 - controlling, 85
- Entering
 - subfile fields, 90
- Esc key, 38
- E-Spec
 - coding, 65
 - declaring, 65
 - extension, 26
- EVAL, 60
- Event
 - bottom, 96
 - click, 38
 - hothit, 105
 - recchanged, 96
 - recdelete, 96
 - recnew, 96
 - top, 96
 - vscroll, 96
- Event handling
 - subroutines, 35
- Event procedure box
 - editor, 36
- Event subroutine
 - adding, 36
 - executing, 34
- Event-driving programming, 34
- Events

- control, 34
- form, 34
- EXE, 13
- Executable program, 13
- Executing
 - event subroutine, 34
- EXFMT, 79, 85
- Existing project
 - opening, 6
- Expanding
 - subfile, 94
- Externally Defined
 - print files, 112
- Externally described file, 35

F

- F1
 - pressing, 22, 24
- F4
 - pressing, 23, 29, 114
- F5
 - pressing, 27
- F5 key, 13, 16
- F7
 - pressing, 7, 11
- Field
 - adding to a format, 114
- Field palette
 - docking, 112
- FieldAdvance property, 41
- FieldAutoAdvance property, 41
- Fields property, 87
- File
 - .VRM, 28
 - externally described, 35
- File menu
 - add files to project, 29
 - open project, 30
- File Menu
 - new project, 10
- File name
 - overriding, 100
- FILEDESC, 100
 - database file name, 52
- First program, 84, 85
- FirstRow property, 95
- Fixed format editor
 - tab stops, 27
- Fixed Format editor
 - stretching columns, 27

- FldHidden property, 95
- Focus
 - changing order, 62
 - receiving, 62
 - setting, 62
- Folder
 - display LearnAVR, 6
- Fonts
 - changing, 60
 - monospaced, 60
 - proportional, 60
 - width, 60
- Form
 - adding, 77
 - blank, 10
 - description, 10
 - events, 34
 - moving, 59
 - new, 106
 - refresh, 59
 - repaint, 59
 - setting focus, 59, 66
 - showing, 79
 - type, 106
 - variable, 106
- Form program, 84
- Form properties
 - changing, 15
 - viewing, 11
- Form1
 - default form name, 11
- Format
 - adding a field, 114
 - header, 114
- Format properties
 - setting, 113
- Forms
 - modal, 79
 - modeless, 79
 - sections, 84
- Free Format
 - code, 129
 - continuation line character, 129
- FrozenCols property, 95
- F-Spec
 - adding, 100
 - coding, 52, 53, 100,
 - external file, 26
- Function key
 - F1, 22, 24
 - F4, 23, 29, 114

- F5, 27
 - F7, 7, 11
- G**
- Getting started, 2
 - Global variables
 - accessing, 85
 - Going to
 - design mode, 13
 - Graph control
 - AutoInc property, 102
 - clicking, 105
 - graphdata property, 102
 - hothit event, 105
 - inserting, 101
 - ThisPoint property, 102
 - ThisSet property, 102
 - GraphData property, 102
 - Graphical user interface, 4, 34
 - Graphics
 - adding, 64
 - Grid, 11
 - GUI
 - graphical user interface, 4
- H**
- Header format, 114
 - Height property, 12
 - Help file
 - identify help topic, 72
 - Help topic
 - context id, 71
 - Help window
 - F1, 22
 - HelpContextId property, 71
 - HelpKey property, 72
 - Hiding
 - windows desktop, 5
 - HotHit event, 105
- I**
- IDE
 - components, 4
 - starting, 4
 - Identifying
 - help topic, 72
 - Indicator
 - LR, 39
 - Inputting
 - text, 41
 - Inserting
 - graph, 101
 - new form, 77
 - new format, 115
 - subfile records, 95
 - Integrated Development Environment
 - IDE, 4
 - Invoking
 - method, 60
 - IOField
 - editcode property, 22
 - properties, 40, 41
 - Items
 - adding, 59, 66, 81
 - adding to ListBox, 61
 - removing, 59, 66
- J**
- Justifying text, 41
- K**
- KbdDelIns property, 95
 - Key
 - Esc, 38
 - K-Spec
 - coding, 53
 - continuation line for f-spec, 26
- L**
- LearnAVR folder
 - displaying, 6
 - Left property, 12
 - Lines
 - displaying in subfile, 95
 - ListCount property, 61
 - Loading a picture, 64
 - Loading all
 - subfile, 94
 - LOADPIC, 64
 - Logic section, 84
 - LR
 - indicator, 39
- M**
- Main program, 84, 85
 - Maximize
 - button, 14

- MaxLength property, 41
- Menu controls
 - adding, 70
 - coding click event, 76
 - description, 70
- Menu editor
 - caption, 71
- Menu items
 - accessing, 71
 - checking, 72
 - enabling, 72
 - visible, 72
- Menu tree, 70
- Menus
 - creating, 70
- Message box
 - coding, 53
- Method
 - additem, 59, 66, 81
 - clearobj, 59
 - ClearObj, 94
 - invoking, 60
 - moveobj, 59
 - refresh, 59
 - removeitem, 59, 66
 - setfocus, 59, 66
- Minimize
 - button, 14
- Modal forms, 79
- Mode
 - design, 12
 - run, 12
- Modeless
 - displaying graph, 107
 - forms, 79
- Monospaced
 - fonts, 60
- More Information
 - step 1, 17
 - step 2, 32
 - step 3, 44
 - step 4, 56
 - step 5, 67
 - step 6, 82
 - step 7, 97
 - step 8, 110
 - step 9, 122
- MoveObj method, 59
- Moving
 - control, 59
 - form, 59

- MSGBOX, 53
- Multiline property, 41
- Multiple controls
 - selecting, 46
- Multiple forms
 - organizing, 84
- Multiple lines of text
 - displaying, 41
- Multiple windows
 - displaying, 105

N

- Name property, 22, 24, 115
- New form, 106
 - inserting, 77
- New format
 - inserting, 115
- New project
 - creating, 10
 - file menu, 10
- NEWFORM, 106
- NoLines property, 95
- Numeric fields
 - add symbols, 41
 - define number of digits, 40
 - punctuate, 40
- NumericLength property, 40

O

- OPEN, 120
- Open Project
 - file menu, 30
- Opening
 - existing project, 6
 - print file, 120
 - project, 30
 - property window, 11, 23
- Organizing
 - multiple forms, 84
- OutOnly property, 41
- Overlapping windows, 105
- Overriding
 - file name, 100
- Overwrite property, 41
- Overwriting
 - text, 41

P

- Parameters

- passing, 85
 - subroutine, 104
- Passing parameters, 85
 - by reference, 105
 - by value, 105
- PasswordChar property, 41
- Physical file
 - browsing, 51
- Picture
 - loading, 64
- Picture property, 64
- Point parameter, 105
- Positioning
 - text, 41
- Pressing
 - F1, 22, 24
 - F4, 23, 29, 114
 - F7, 7, 11
- Print file
 - closing, 120
 - coding, 118, 119
 - creating, 112
 - opening, 120
- Print File Editor
 - adding a field, 114
 - data field, 115
 - field palette, 112
 - inserting new format, 115
 - name property, 115
 - setting format properties, 113
- Print files
 - externally defined, 112
- Printer
 - setup, 120
 - specifying, 120
- Printing
 - document, 120
- Procedural program, 84
 - adding, 85
- Procedural subroutines, 35
- Program
 - compiling, 13
 - executable, 13
 - first, 84, 85
 - form, 84
 - main, 84, 85
 - procedural, 84
 - running, 12
 - saving, 29
 - startup, 84, 85
- Programming
 - event-driven, 34
- Project
 - adding files to, 29
 - creating, 10
 - opening, 6, 30
- Properties
 - alignment, 41
 - autoextend, 94, 95
 - autoinc, 102
 - borderstyle, 14, 103
 - browse, 94
 - caption, 12
 - celldata, 96
 - changing, 29
 - changing at run-time, 24
 - controlbox, 14
 - decimals, 40
 - default, 22, 38
 - editcode, 22, 40
 - editword, 41
 - fieldadvance, 41
 - fieldautoadvance, 41
 - fields, 87
 - firstrow, 95
 - FldHidden, 95
 - frozencols, 95
 - graphdata, 102
 - height, 12
 - helpcontextid, 71
 - helpkey, 72
 - IOField, 40, 41
 - KbdDelIns, 95
 - left, 12
 - listcount, 61
 - MaxLength, 41
 - multiline, 41
 - name, 22, 24
 - nolines, 95
 - NumericLength, 40
 - outonly, 41
 - overwrite, 41
 - passwordchar, 41
 - picture, 64
 - recchange, 95
 - referring to in code, 26
 - row, 92, 96
 - RRNFldName, 92
 - setting, 22
 - shortcut, 72
 - tabindex, 62
 - tabstop, 62

- thispoint, 102
- thisset, 102
- top, 12
- topindex, 61
- uppercaseonly, 41
- value, 24, 28
- width, 12
- Property window
 - description, 11
 - opening, 11, 23
- Proportional fonts, 60
- Punctuating
 - numeric fields, 40, 41
- UpperCaseOnly property, 41
- R**
- READC, 87
- RecChange property, 95
- RecChanged event, 96
- RecDelete event, 96
- Receiving focus, 62
- RecNew event, 96
- Record has changed, 95
- Referring to a property in code, 26
- Refreshing
 - control, 59
 - form, 59
- Refresh method, 59
- Relative record number
 - specifying, 92
- RemoveItem method, 59, 66
- Removing
 - item, 59, 66
 - row, 59, 66
- Repainting
 - control, 59
 - form, 59
- Resizing
 - control, 46
- RETRN, 85
- Root menu, 70
- Row
 - adding, 59, 66, 81
 - removing, 59, 66
 - selected, 92
- Row property, 92, 96
- RRNFldName property, 92
- Run
 - button, 6
 - debug menu, 13, 16
 - mode, 12
- Running
 - program, 12
- Run-time
 - changing properties, 24
- S**
- Saving
 - application, 29
 - program, 29
- Scrolling
 - subfile, 96
- Second form
 - adding, 77
- Selected
 - row, 92
- Selecting
 - control, 46
 - multiple controls, 46
 - run button, 6
- Set parameter, 105
- SetFocus method, 59, 66
- Setting
 - focus, 62
 - format properties, 113
 - properties, 22
 - property value, 22
 - subfile coordinates, 96
- Setting Focus
 - control, 59, 66
 - form, 59, 66
- Settings Box, 22, 31
- Shortcut key
 - adding, 72
- ShortCut property, 72
- SHOW, 80, 107
- Showing
 - form, 79
- Single-page
 - subfile, 94
- Source files
 - directory, 2
- Spec column, 25
- Spec formats, 25
- Spec type
 - blank spec, 39
 - c-spec, 35
- Specifying
 - printer, 120
 - top subfile row, 95

- Spooling
 - document, 120
 - SRPARM, 104
 - S-Spec
 - data structure, 26
 - Starting the IDE, 4
 - Startup program, 84, 85
 - Step 1
 - more information, 17
 - Step 2
 - more information, 32
 - Step 3
 - more information, 44
 - Step 4
 - more information, 56
 - Step 5
 - more information, 67
 - Step 6
 - more information, 82
 - Step 7
 - more information, 97
 - Step 8
 - more information, 110
 - Step 9
 - more information, 122
 - printing, 111
 - Stretching
 - columns, 27
 - Subfile
 - autoextend property, 94
 - browse property, 94
 - browsing fields, 94
 - data entry mode, 94
 - displaying without lines, 95
 - expanding, 94
 - fields property, 87
 - firstrow property, 95
 - FldHidden property, 95
 - frozencols property, 95
 - get hidden field, 95
 - KbdDelIns property, 95
 - loading all, 94
 - nolines property, 95
 - recchange property, 95
 - recchanged event, 96
 - recdelete event, 96
 - recnew event, 96
 - row property, 92, 96
 - rrnflname property, 92
 - scrolling, 96
 - single-page, 94
 - specifying non-scrolling columns, 95
 - specifying relative record number, 92
 - specifying top row, 95
 - top event, 96
 - vscroll event, 96
 - Subfile control, 87
 - Subfile coordinates
 - setting, 96
 - Subfile event
 - bottom, 96
 - Subfile field
 - visibility, 95
 - Subfile fields
 - entering, 90
 - Subfile record
 - adding, 96
 - changing, 96
 - deleting, 96
 - Subfile records
 - deleting, 95
 - inserting, 95
 - Subroutine parameters, 104
 - Subroutines
 - beginning, 36
 - event handling, 35
 - procedural, 35
- ## T
- Tab key
 - on numeric pad, 41
 - Tab sequence
 - automatically advance, 41
 - Tab stops
 - fixed format editor, 27
 - TabIndex property, 62
 - TabStop property, 62
 - Text
 - inputting, 41
 - justifying, 41
 - overwriting, 41
 - positioning, 41
 - ThisPoint property, 102
 - ThisSet property, 102
 - Time
 - displaying, 101
 - TIME, 101
 - Title bar
 - changing, 11
 - changing modes, 13
 - double-clicking, 5

Title Bar properties
borderstyle, 14
controlbox, 14
Top event, 96
Top property, 12
TopIndex property, 61
Tutorial
appendix b, 129
getting started, 2
source files, 2
step 9, 111
Type
of form, 106
of control, 28

U

UPDATE, 87, 92
Using
menu editor, 70

V

Value property, 28

Variables
defining numeric digits, 40
form, 106
global, 85
Viewing
form properties, 11
Visibility
menu items, 72
subfile field, 95
Visual section, 84
VScroll event, 96

W

Width property, 12
Windows
overlapping, 105
printer dialog, 120
Windows Desktop
background, 5
WRITE, 87, 92