C L A R I O N 5

Interactive Self-Study Essentials of Clarion Lab Exercises

TopSpeed Corporation
150 East Sample Road
Pompano Beach, Florida 33064
(954) 785-4555

Printed in the United States of America (0400)

TABLE OF CONTENTS

Introduction

Welcome to the Essentials of Clarion Lab exercises.

The Lab exercises take you step by step through various topics. You will develop an Orders application starting with the wizards. A completed dictionary is provided You should have covered the material about the wizard options in the dictionary. You will then move on to enhance this application, some *without writing a single line of code*, others use very little coding to accomplish a seemingly complex task. The main objective is to expose you to the ABC classes and templates in a smooth gradient.

As you progress through the material in the exercises, you'll have the chance to use the topics you study. A CD-ROM is included with this material and it contains all of the sample code and lab projects you will be using. As you work through the laboratory exercises in this manual, you can follow along with the provided applications.

Here is a summary of what you will find in each Laboratory Lesson:

- In Lab 1, you run two applications that show you events and properties. The source code is exposed to you so that you can see how the two relate. This helps you understand and locate where to find embed points in a later lesson.
- In Lab 2, you start with a partial Orders application and break it up into two DLLs. You learn how to make DLLs in your own projects. This lab doubles as a reference and checklist when you need to do this in your own applications.
- In Lab 3, you learn how to construct reports and view the features in the formatter that many seasoned Clarion developers overlook. You create two reports, one very simple (while showing you a nice shortcut) and the other which shows you how to make a complex report with multiple files and group breaks.
- In Lab 4, you learn how to find an ABC embed point, what it means, and how little code you need to write to accomplish a not so trival task.
- In Lab 5, you learn how to use the debugger effectively and how to find problems in a problem application without wading though many lines of code. This lab finds a nasty bug in ABC quickly.
- In Lab 6, you learn how to add an OCX control to an application.
- In Lab 7, you need to use an API function. This walks you through it step-by-step.
- In Lab 8, you add drag and drop to the Orders application so it prints a report based on the selected item in a list box. This lab uses the report you made in Lab 3.

Documentation Conventions

Typeface Conventions:

Italics	Indicates what to type at the keyboard, such as *Enter This*.
SMALL CAPS	Indicates keystrokes to enter at the keyboard, such as ENTER or ESCAPE, or to CLICK the mouse.
Boldface	Indicates commands or options from a menu, or text in a dialog window. Note: this style can also utilize a different typeface to match the helvetica bold face which Windows uses as the system font.
LETTER GOTHIC	Used for diagrams, source code listings, to annotate examples, and for examples of the usage of source statements.

Keyboard Conventions:

F1	Indicates a single keystroke. In this case, press and release the F1 key.
ALT+X	Indicates a combination of keystrokes. In this case, hold down the ALT key and press the X key, then release both keys.

About the Companion CD-ROM

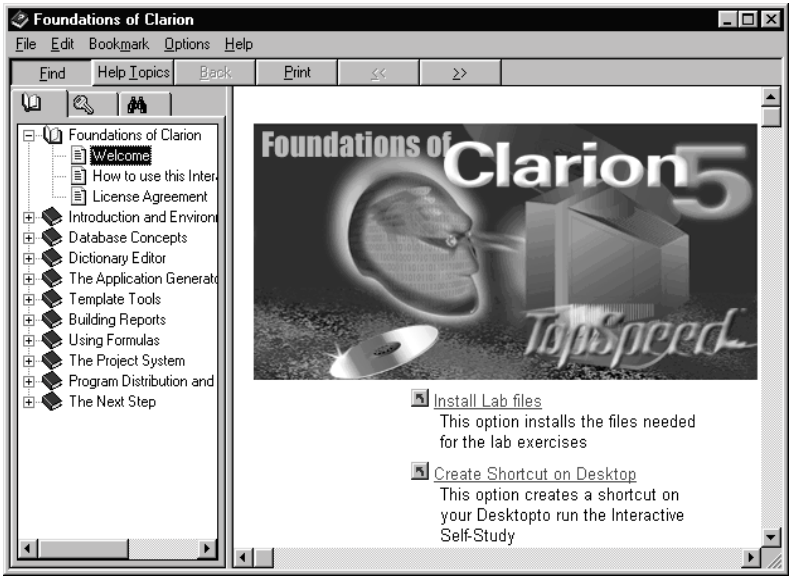
A CD-ROM is included with this material. In addition to the on-line tutorials, the CD includes all of the sample projects and applications required for the Laboratory sessions.

Installing the Lab Applications

The install program has a user option that copies the sample project folders and files to a folder named X:\Clarion5\CBT\Essentials on your hard drive. (Drive *X* is the drive that you chose to install the lab exercises and examples to. Normally, this would be '*C:*').

If you have not copied the sample projects and applications to your hard drive, do so now.

1. **Close** any currently running programs**.**
2. **Insert** the *Essentials of Clarion* CD into your CD-ROM drive**.**
3. **Launch** the setup program, and follow the instructions on the window to install the sample applications.



Lab Applications on CD

Each Laboratory(lab) session is designed to reinforce the concepts presented in the lessons. As you perform each lab, you are directed to start Clarion and open the appropriate files.

The CD-ROM includes example applications, lab projects, and solutions. The files follow this naming convention:

Sample Type	Filename Convention
Lab	X:\Clarion5\CBT\Essentials\lab02\start
Solution	X:\Clarion5\CBT\Essentials\lab02\solution

The lab applications are the starting point for each exercise. They give you a quick start by providing the necessary data that you will modify using the lab exercise. This offers an easy way for you to try selected topics without having to always start with creating an application from scratch. Some labs do not follow the above convention. However, all labs give you precise instructions on how to proceed.

Solution

These are solutions to the laboratory exercises. Use the solution projects to check your work on an exercise, or use them before you start the exercise, or at any time, to get a feel for what you are about to accomplish. Every lab exercise has a solution application.

Most of the Solution Application folders contain an executable file (.exe) that you can run at any time. All you have to do is navigate to the folder of interest, and launch the application.

All of the Solution Application folders contain Application, Dictionary, and data files. Once again you can load these into Clarion at anytime for review.

When you are ready, roll up your sleeves, flex your mouse finger, and proceed to Lab Exercise 1!

Lab 1

Events and Properties

Introduction

What We Are Going to Accomplish

This example project shows different events as you interact with the program. The Lab guides you through various controls and windows, displaying events as they happen. To assist you with this lab, open the *Language Reference Manual* to *Appendix B*. This is the list of events, by event type.

TestEvent

Exercise Starting Point

You are now going to launch the TestEvent.EXE application, and explore how it behaves.

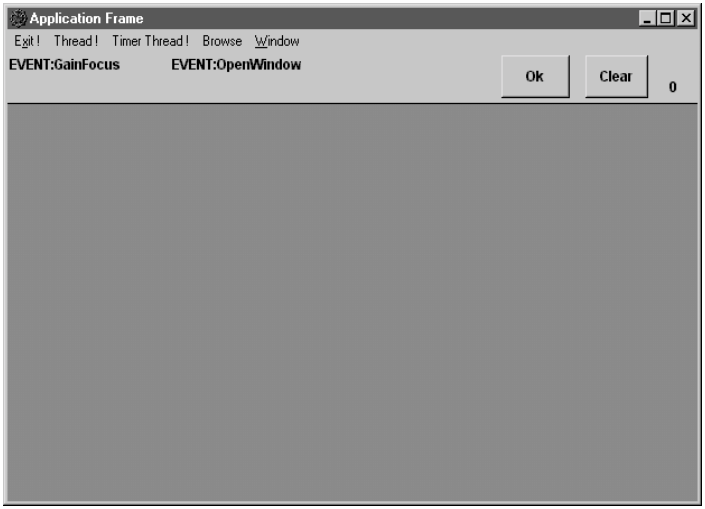
Starting Point:

1. From Windows, **navigate** to the *X:\Clarion5\CBT\Essentials\Lab01\Events* folder.
2. DOUBLE-CLICK **testevnt.exe**.

   The Test Event Application is ready for review.

Now let's explore the application, taking note of the events displayed as you interact with it.

1. **Notice** the events displayed when the program starts. What *type* of events are they?



Both of these events are *window* or *field-independant* events. A Field-independent event does not relate to any one control but requires some program action (for example, to close a window, quit the program, or change execution threads). Most of these events cause the system to become modal for the period during which they are processing, since they require a response before the program may continue. It is important to recognize the *type* of event as this assists you in locating an embed point later. Becoming familiar with the events helps you determine which embed to use.

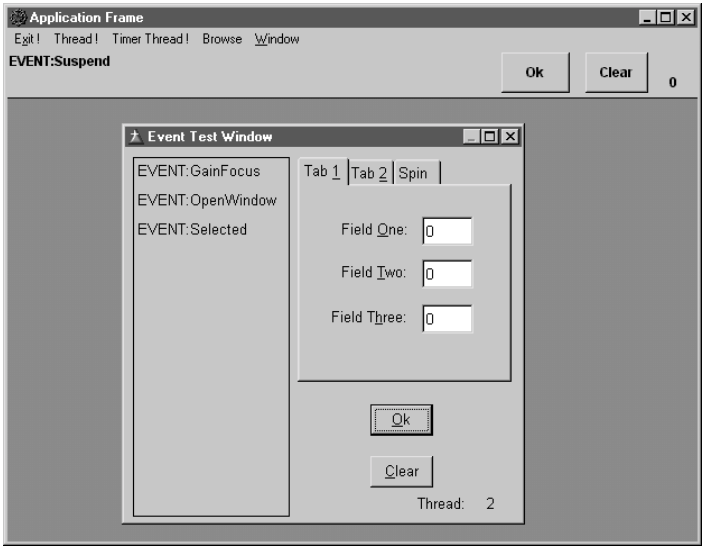
Entry controls

1. **Choose** the **Thread!** menu item on the menu bar.

   A new window opens and more events display. What type of events are they? Notice the control on the window that has focus.

2. **Press** TAB.

   Which control has focus now? What event did this action trigger? For more information, see *Appendix B* in the *Language Reference Manual*.



3. **Click** on the **Field One** entry control.

   Notice the event(s) displayed.

4. **Press** TAB to move to the next entry control.

   Notice the new event displayed on the list to the left? Is it the same or different than before?

5. **Press** SHIFT-TAB to move back to the previous entry control. Type any character and **press** TAB.

   Which event(s) displayed when you did this?

6. **Press** SHIFT-TAB again to move back to the previous entry control. With the contents of the entry field highlighted, type any character to change the value. Without leaving the entry control, change it back to the original value.

   Which event(s) do you think will display? Remember, you are leaving the entry control with the same value as you started with. **Press** TAB after you make your prediction and see if it was accurate.

   Refer to *Appendix B* in the *Language Reference Manual* and look up the event triggered. Why did it this event happen when you changed the value, even though you changed it back to the original value?

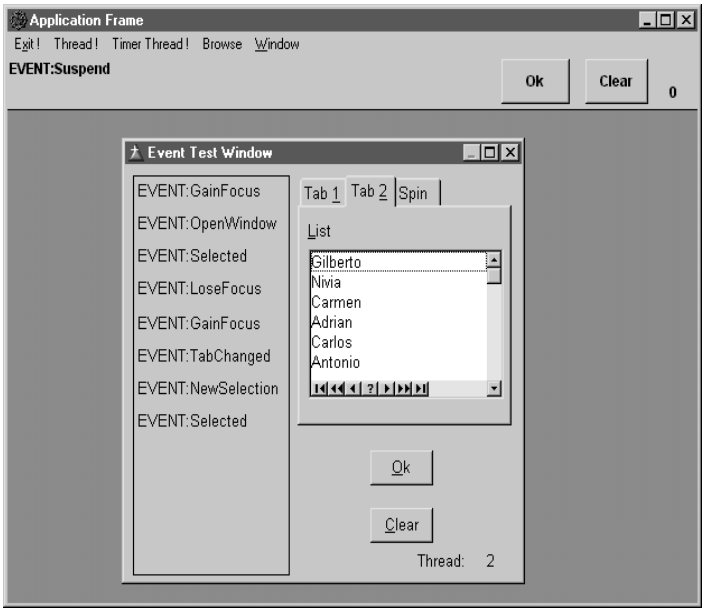
7. If you **select** the **Tab2** control, which event(s) do you think will display?

   Select it and see if your prediction is accurate. Refer to *Appendix B* in the *Language Reference Manual* for the definition of the event(s).

List controls

List controls generate events. This exercise exposes these events. The purpose is to become familiar with the events that you can detect and act upon in your applications when working with list controls.

From the last step in the previous section, not only did you discover the event a tab generated, but notice that you got another event for selecting the list control itself. This is in addition to any list-specific events. List controls generate many events, thus it is important to become familiar with them. Your application should look like this:



1. **Select** a name in the list and notice the event(s) generated.
2. **Press** the up and down arrow keys to select a new item.

   Notice the event(s) generated by this action. Is it the same as selecting a record with the mouse?
3. Notice the VCR controls at the bottom of the list control. **Press** each button and notice the different events for each. Refer to *Appendix B* in the *Language Reference Manual* for the definition of each of these events.
4. Using the mouse, **drag** the thumb button on the scroll bar up and down.

   Notice the event(s) for this action.
5. **Release** the mouse button and notice the new event generated. Leave the thumb about halfway in the vertical scroll bar.

6. **Click** on the vertical scroll bar above and below the thumb button.

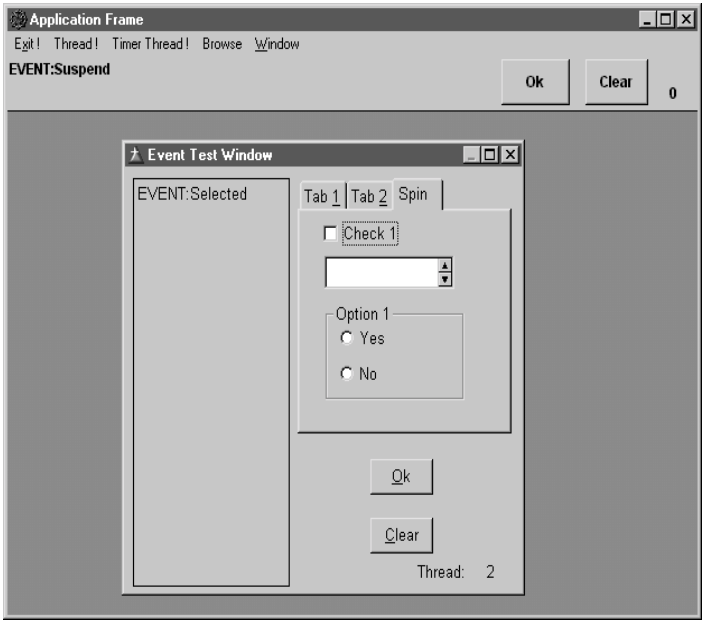
   Which event(s) generate when you do this?

Other controls

This lab explores events generated by a checkbox, spin and radio buttons.

1. **Select** the **Spin** tab.

   Your application should look similiar to the following image:



2. **Check** the check box.

   Which event(s) generated? What if you click on it again? Any difference? How is it different than an entry control?

3. **Press** TAB to select the spin control.

   This control is set to adjust a time.

4. **Enter** *13:34*.

5. **Press** either of the spin buttons and notice the event generated.

   Have you seen this event before? What other control can generate this event?

6. **Press** TAB to select the first radio button.

What event(s) did leaving the control generate? What control generates this event in a similar fashion?

7. **Choose** the **Yes** radio button.

   What event(s) generated? Click on this button again? Any new events?

8. **Choose** the **No** radio button.

   Which event(s) do you see on the list?

9. **Press** the up and down arrow keys to select the radio buttons.

   Notice any events?

10. **Press** SPACEBAR to activate a radio button. Which event(s) appear in the list on the left?

11. **Close** this window when you are done.

Timer events

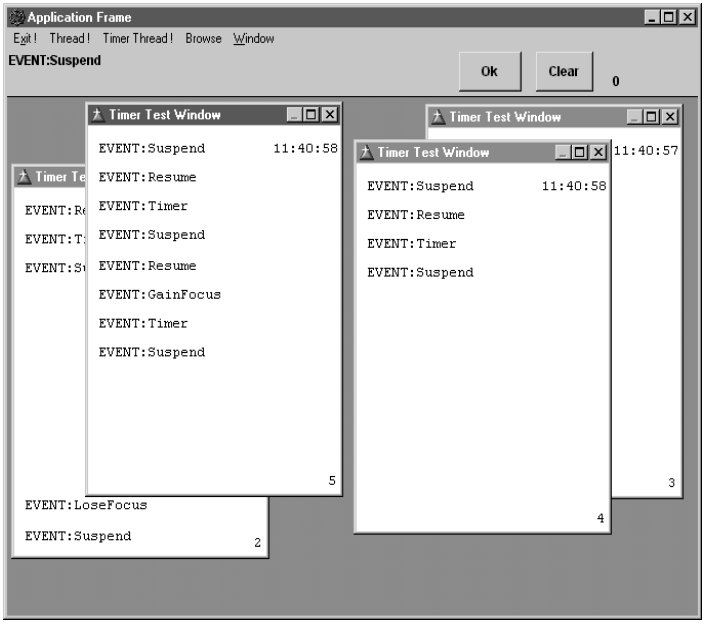
This exercise illustrates timer events. This is defined as an event that generates at regular intervals.

1. **Choose** **Timer Thread!** about four or five times.

   You don't see this, but there are multiple windows open in your application.

2. **Choose** the **Window ➤ Cascade** or **Window ➤ Tile** to arrange the windows.

   You can drag the windows around on the application's desktop so that you see all of the open windows. You should have a display similiar to this:



3. **Watch** the windows.

   Notice the events that are generated at regular intervals for each window.

> **Note:** **The timer interval is set in the code. We will examine the code later in this lab.**

4. **Close** the TestEvent application when you are done.

Examing the source code

This section of the lab shows you the Clarion source code for the TestEvent program. Since this application was hand-coded, it is a project.

You should have just started the Clarion development environment. The Pick dialog should be open.

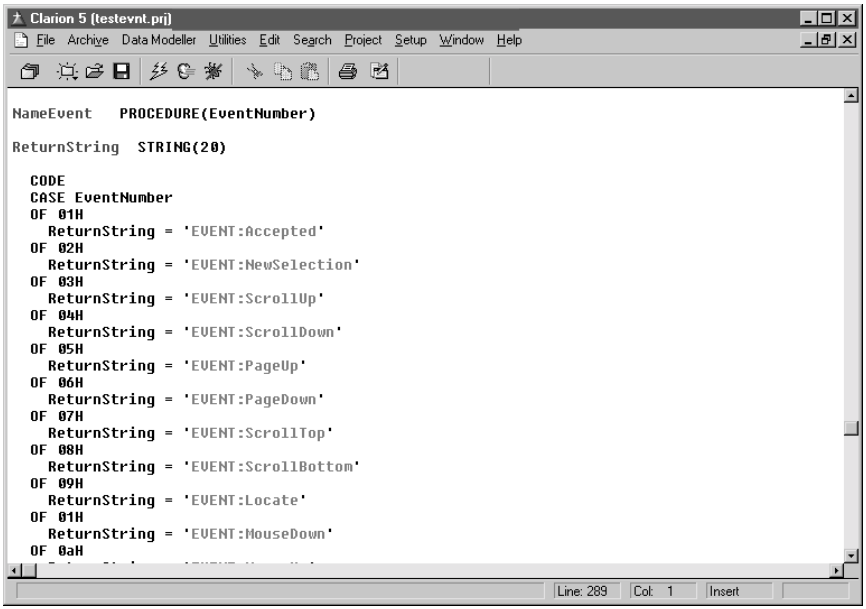
Exercise Starting Point

1. **Select** the **Project** tab. **Press** the **Open** button and navigate to the *X:\Clarion5\CBT\Essentials\Lab01\Events* folder.
2. **Highlight testevnt.prj** and **press** the **Open** button.
3. **Highlight testevnt.clw** under the **External source files** node of the project tree.

4. **Press** the **Edit** button.

   This opens the source code in the Clarion text editor.

5. **Press** PAGE-DOWN about 7 times. You should see a procedure labeled *NameEvent*.



   This is the source for the procedure that returns the name of the event detected. This simply receives the event number from another procedure and looks for a matching number in the CASE statement. Once it finds a match, it copies some text to a string variable and returns that data to the calling procedure. You see this as the event name on the window.

6. **Press** PAGE-DOWN until you get to the bottom of the procedure.

   Notice the statement looking for a range of values. These are the user defined events discussed in the lesson text.

7. **Close** the source window and the project tree.

   You are returned to the Clarion desktop.

Properties

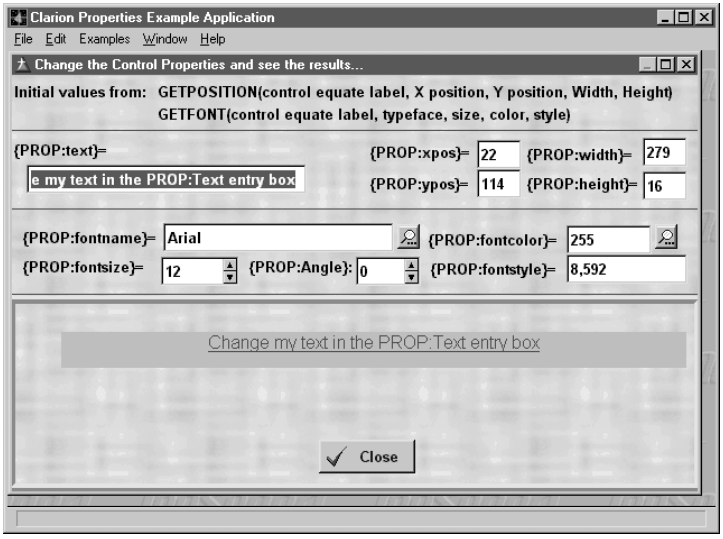
This lab exercise shows you property statements in action. This lab uses an application to demonstrate some of the common properties you can use in your applications.

Load the Clarion development environment if it is not already open. The Pick dialog should be open.

Starting point

1. **From the Application tab on the Pick dialog**, **press** the **Open** button. Navigate to the *X:\Clarion5\CBT\Essentials\Lab01\Property* folder. **Highlight props.app** and **press** the **Open** button.
2. **Once the application is open, press** the **make and run** toolbar button to compile and run the application.
3. **Choose Examples ➤ Control properties** from the menu.

   You see this display:



4. **Change** the *PROP:Text* entry by entering anything you wish, then **press** TAB.

   What happened in the display area?

5. **Change** the values in the *PROP:xpos*, *PROP:yos*, *PROP:width*, and *PROP:height* entry fields and observe the changes in the display area.
6. **Press** the lookup button next to the *PROP:font* entry and chose a different font.

   Did the font change as expected? Lets try some of the other font settings.

7. **Enter** a new value for the *PROP:fontcolor* entry or **press** the lookup button to choose a new color.

   Did the color change as expected?

8. **Spin** the *PROP:fontsize* by **pressing** the up and down buttons.

You may either **press** on them or hold the button down and watch the effect on the display area.

9. **Spin** the *PROP:Angle* entry.

Note: You must use a True Type font for this feature to work correctly.

10. **Enter** *800* in the PROP:fontstyle entry and press TAB.

This is defined as an integer constant or constant expression or EQUATE specifying the strike weight and style of the font. See the *Help* under *FONT (set default font)* for details.

11. **Press** the **Close** button when you are finished.

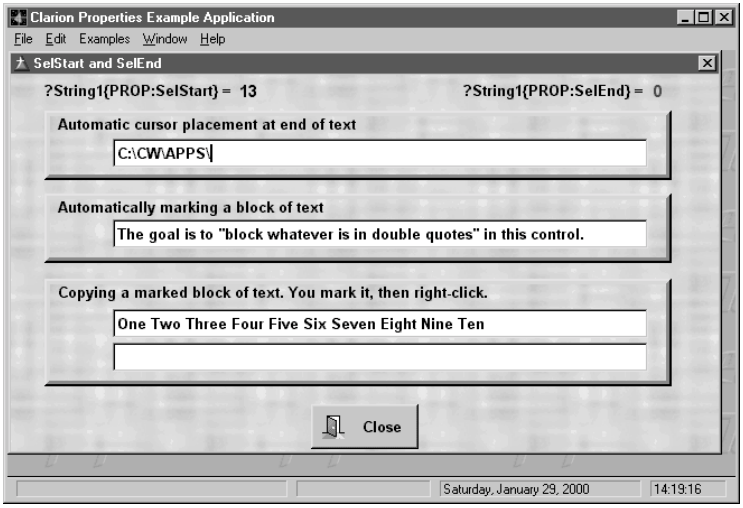
PROP:SelStart and PROP:SelEnd

PROP:SelStart sets or retrieves the beginning (inclusive) character to mark as a block in an ENTRY or TEXT control. It positions the data entry cursor to the left of the character, and sets PROP:SelEnd to zero (0) to indicate no block is marked. It also identifies the currently highlighted entry in a LIST or COMBO control.

PROP:SelEnd sets or retrieves the ending (inclusive) character to mark as a block in an ENTRY or TEXT control.

1. **Choose Examples ➤ SelStart SelEnd** from the menu.

The following window displays:



At the top of this window, you see the values of PROP:SelStart and PROP:SelEnd. PROP:SelStart is set to a value of *13* when the window opens. Notice that the number of characters displayed in the *Automatic placement of cursor in text* entry has only 12 characters. This means that you can "overshoot" whatever text is there and always get the cursor to the end of the string.

2. **Press** TAB.

   You will notice that the selection is between the two quote marks and that PROP:SelStart and PROP:SelEnd show their values at the top of the window.

3. **Press** TAB to advance to the next entry.

   The goal is to select any text in this control and paste it to the control underneath it.

4. **Press** the left mouse button in the middle of the word *Two*.

   This puts the cursor on it.

5. While holding the left mouse button, move the mouse to the right until you are in somewhere in the middle of the word *Seven*. Release the mouse button.

   This action selects half of the word *Two* and somewhere in the middle of the word *Seven* and all text between them.

6. **Press** the right mouse button.

   What happened with your selection?

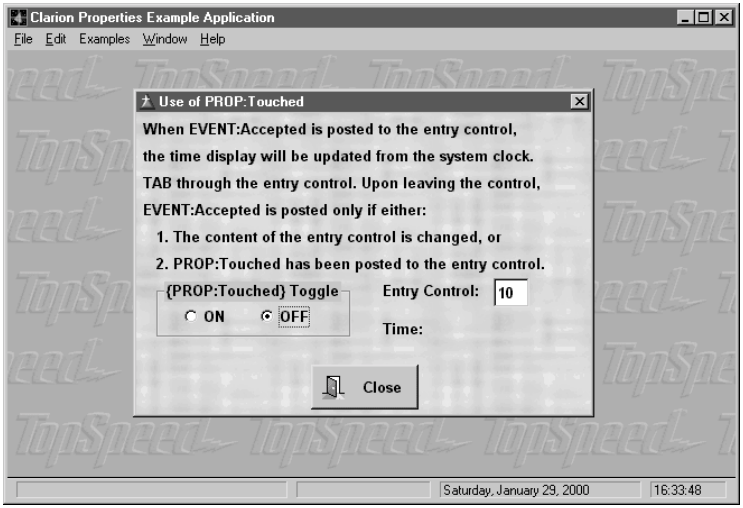
7. Press the Close button when you are finished.

PROP:Touched

This exercise explores how you can use this property in your applications.

1. **Choose Examples ➤ Event:Accepted** from the menu.

   You see this window:



2. **Press** TAB a few times.

   You should note that nothing happens. This is the default behavior for entry controls. This window displays the current time when an EVENT:Accepted occurs.

3. **Select** the entry control.

4. Change the default numeric value and **press** TAB.

   You see that the current time is placed on the window.

5. Now hold down TAB.

   You see the focus of the controls moving to the next until you release **tab.** Notice that the time display does not change.

6. **Choose** the **ON** radio button.

7. **Press** TAB a few times.

   Notice a change in the behavior. Every time you leave the entry control, the current time is displayed. You can see that on entry controls, if a user did not change anything, they can leave the entry control without generating an EVENT:Accepted. See *Appendix B* in the *Language Reference Manual*. Sometimes a design requires an entry control to perform some action, such as data validation, even if the user did not actually do anything.

   Setting PROP:Touched to True when a user selects a control (EVENT:Selected which always occurs), will force the EVENT:Accepted event to generate. This applies even when a user has not performed any edits.

8. **Press** the **close** button.

9. **Exit** the application.

Source Code

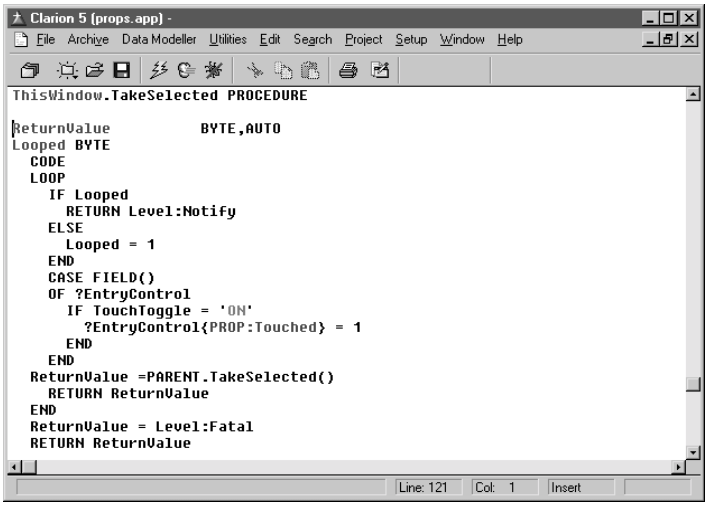
You have seen the effects of property changes during runtime. Now lets examine how this is done in Clarion code.

Starting point

1. The PROPS.APP application is open showing the procedure tree.
2. **Highlight** the *Touched* procedure.
3. RIGHT-CLICK to display the popup menu then CLICK **Module**.

   This opens the generated source file for this procedure.
4. Go to the bottom of this source module.

   You will see the following code:



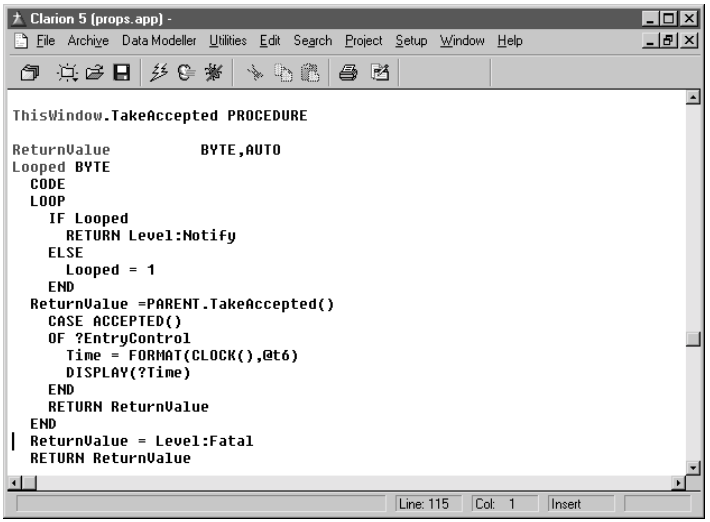
The code that is of interest is the CASE FIELD() structure. If the TouchToggle radio button is set to the value ‘ON’, then set the PROP:Touched value to True (or 1).

The next executable statement calls the bit of code for the event, “EVENT:Selected”. This code is contained elsewhere and we will cover this in later lessons.

The main point to consider is that when a user lands on or selects this control, PROP:Touched = True says that even if a user did nothing but do some action to leave this entry, then Clarion thinks that the user did change the data. This causes any code placed in EVENT:Accepted to execute.

5. **Scroll** up to the previous procedure.

   You see this code:



   This is the code that executes when an EVENT:Accepted generates.

6. Look at the CASE Accepted() structure.

   The Accepted() statement returns the field number of the control on which an EVENT:Accepted event occurred. The next line of code checks if it is the EntryControl. If it is, get the current time from the system and display it.

   If the EVENT:Accepted event does not generate, then this code quits as the condition is false.

   It is worth noting that this code does execute if the user changes the data in the entry control and then moves off the control. In this case, it does not matter what value PROP:Touched is set to.

7. **Close** the source window.

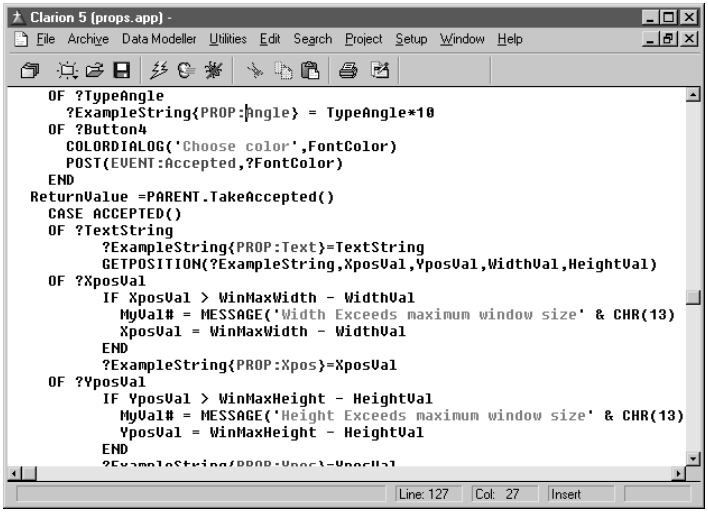
   You are returned to the procedure tree.

8. **Open** the generated source module for the *ControlProperties* procedure. RIGHT-CLICK on the procedure and **choose module**.

   This opens the editor and loads the source module.

9. Find the line of code that determines the angle of a string.

This is about line 127 (see the status box at the bottom of the editor). The code you see looks similar to the following:



10. **Position** your mouse cursor on PROP:Angle and **press** F1.

This brings up the Help and goes to the help topic, *ANGLE (set control display or print angle)*. Since PROP:Angle sets the angle of a string control, this is the correct topic. Notice that PROP:Angle is mentioned here. You may do this for any property in Clarion code as they all have links to the help topic that you may need more information.

Close the help file.

11. Look down a few lines and notice the PROP:Text and PROP:Xpos property expressions.

12. **Position** your mouse cursor on each and find their help topics.

Close the Help when you are done.

13. **Inspect** the rest of the code in this section and that all of the property expressions change some attribute of the ?ExampleString field equate.

Do you see how many attributes or properties you can affect on one control?

14. **Close** the source editor and close the PROPS.APP.

Summary

During this lab exercise, we:

- Explored two different types of events, field-independant and field dependant events and what they are.
- Used an example application to show these events in action.
- Different controls can generate different and similar events.
- How an event can be detected in Clarion code.
- What a property is.
- Used an example application to explore some common property statements.
- Inspected Clarion source code to see how to use events to change properties of a control.
- The quick and fast way to look up an event or look up a property in the Help system.

Lab 2

Dynamic Link Libraries

Introduction

What We Are Going to Accomplish

We are going to take the Orders application that you created in the lesson text and configure it to use two DLLs.

What we will accomplish during Lab 2:

- Create a DLL with no procedures.
- Create another DLL that maintains only one data file.
- Explore the Global switches that make this possible.
- Link these two DLLs into our executable.
- Change one of the DLLs and demonstrate a deployment strategy.

Exercise Starting Point

Note: Every Lab session has a corresponding solutions folder. This folder contains the Application files, Dictionary File, Data Files, DLLs and an Executable File. You can launch and view a working application that represents the solution of this Lab.

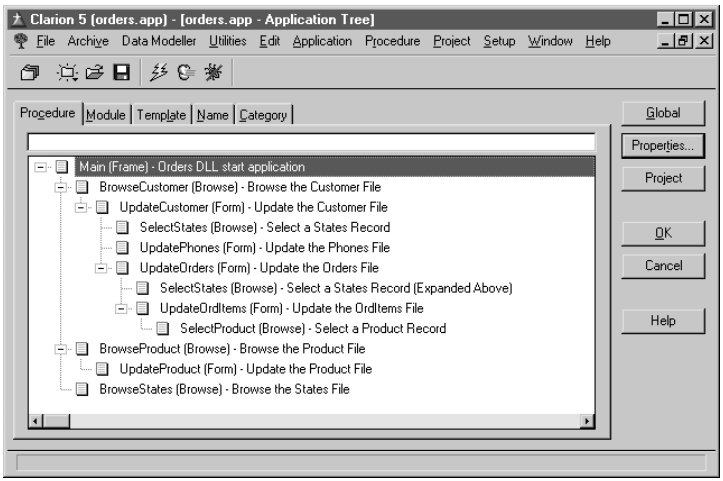
To launch the solution application for this Lab session, from Windows navigate to the *X:\Clarion5\CBT\Essentials\Lab02\Solution* folder and DOUBLE-CLICK on orders.exe.

You should have just started the Clarion development environment. The Pick dialog should be open.

Exercise Starting Point

1. **N**avigate to the *X:\Clarion5\CBT\Essentials\Lab02\Start* folder.
2. **Highlight orders.app** and **press** the **Open** button.

Your application should look like the following:



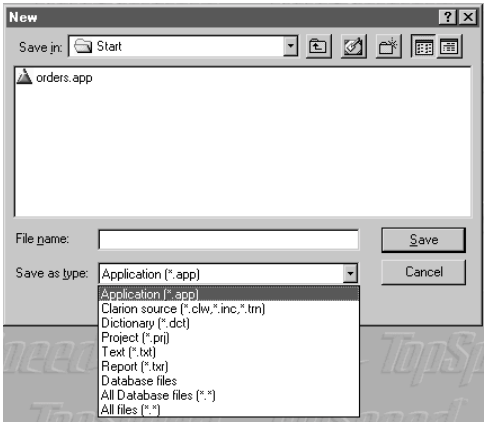
We would like to create a DLL that contains all procedures that maintain the States file, *States.DLL*. The global variables, dictionary variables, and internal variables are placed in the *Global.DLL*. The Orders application is open so it is easier to visualize what we need to accomplish.

2. **Press** the **OK** button to close the Orders application.

   We will come back to this later once we make the DLLs it needs.

3. **Open** the pick dialog and **press** the **New** button.

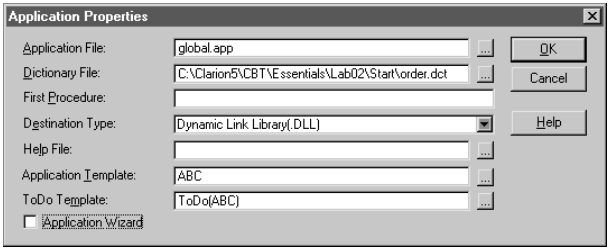
4. **Navigate** to the *X:\Clarion5\CBT\Essentials\Lab02\Start* folder and **select** *Application(*.app)* from the **Save as Type** drop down list box if this is not already shown.



5. **Type** *Global* in the **File name** entry box, and **press** the **Save** button. Make sure that the **Use Quick Start** box is not checked.

   The Application Properties dialog appears.

Global application



1. Since the Dictionary File has already been created, **press** the ellipsis(...) button (just to the right of the **Dictionary File** entry field). **Highlight** *order.dct* and **press** the **Open** button.

2. **Clear** the **First Procedure** name.

   Do not to use the space bar to type over the name as this gives the first procedure a label of space. This is an illegal character for labels!

3. **Select** *Dynamic Link Library (.DLL)* from the Destination type drop list.

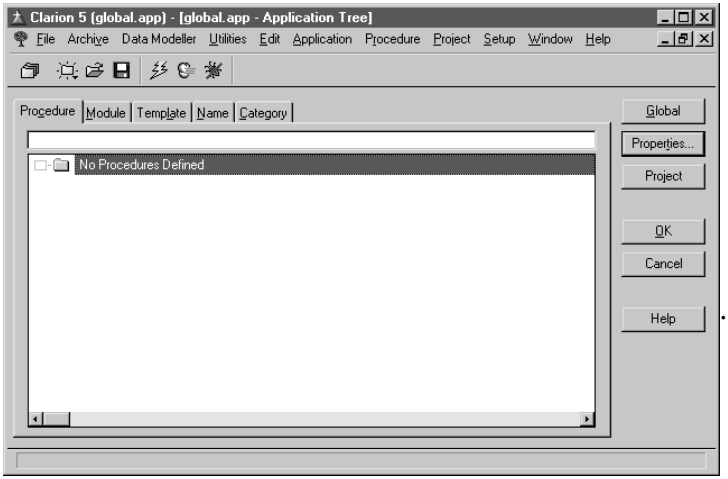
   This means that we are going to start right away with a DLL instead of an executable. This makes sense as we are not going to create any procedures in this DLL.

   Application Template should be set to ABC.

4. Make sure the **Application Wizard** box is *not* checked (this is located on the lower left hand corner of the **Application Properties** dialog).

5. **Press** the **OK** button.

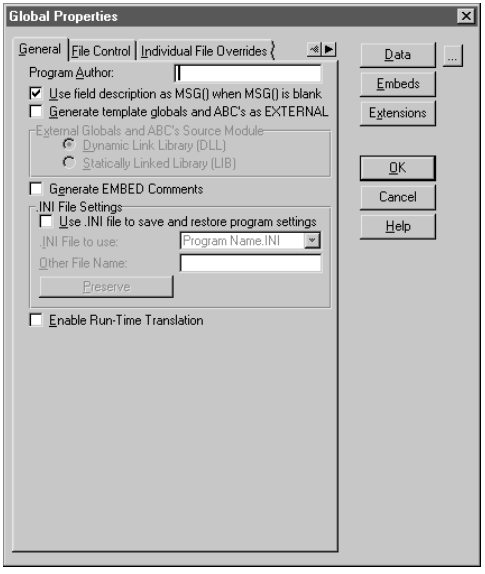
   You should now have a new application that looks like this:



You now have an application that could be successfully compiled. However, it would be quite useless as there is no way into the DLL. It is missing 'entry points', or areas where other DLLs and executables gain access to data. Anything contained in this DLL is completely closed off to any other application. What we need to do now is make some of these 'entry points'.

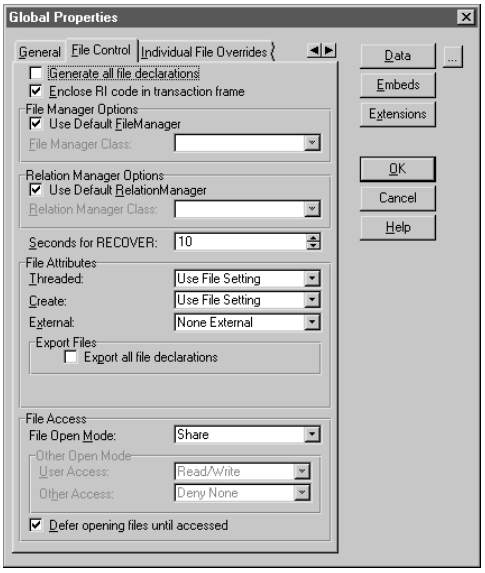
6. Press the **Global** button.

This dialog is displayed:



This dialog is divided into serveral tabs. For the global DLL, we do not need to do anything on the *General* tab. This DLL is the first DLL in our project or collection of DLLs and executables. Recall the purpose of this DLL is simply to provide access to all global data variables and this includes the internal variables used by the ABC templates as well as any variables declared in the data dictionary. This means that you do need to change some setting in all other *later* DLLs and executables. We will cover this in the next section.

7. **Select** the **File Control** tab.



This tab establishes default behavior for each file declared in the dictionary (including global ‘files’). If you wish to override individual files, the **Individual File Overrides** tab allows you to set different default behaviors on a file by file basis. It has the same dialog as shown above, but for a chosen file.

8. **Check** the **Generate all file declarations** box.

This means that the ABC templates generate all file declarations from your dictionary. The default behavior is to generate file declarations for each file referenced by a procedure. Since there are no procedures in this application, this means that no file declarations generate. This violates the design of having all dictionary variables stored in this application.

9. **Check** the **Export all file declarations** box.

This check is available only when the target type is not an executable (EXE). The word ‘export’ means ‘a DLL entry point’. So this allows all dictionary variables to be seen outside of this DLL. Notice the **External** drop list is set to *None external*. This means that all file and field declarations are the original definitions. All dialogs are from the viewpoint of the current application.

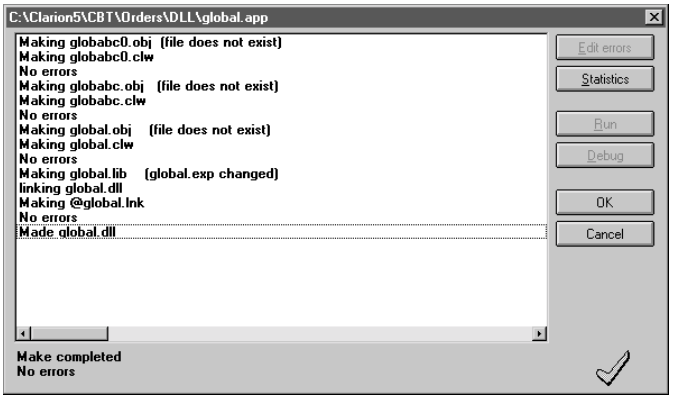
10. **Press** the **OK** button to close this dialog.

11. We are now going to make this a 32-bit application. **Press** the **Project** button. This will display the **Project Editor** window. **Press** the **Properties** button. Select **Windows - 32-bit** from drop down listbox for the **Target OS**. **Press** the **OK** button. When you are back at the **Project Editor** window, **press** the **OK** button to save this setting. You should now be back to the application tree.

12. **Press** the **Save** button on the toolbar to save your work.

13. **Press** the **Make** button to compile your application.

If all has gone well, you should see a similar dialog to this:



You have made your first DLL. Congratulations!

14. **Press** the **OK** button and close the Global application.

States application.

1. **Open** the pick dialog and **press** the **New** button. The file type is Application and the name of the new file is *States*.

Make sure the **Use Quick Start** box is not checked. **Press** the **Save** button.

2. The *Application Properties* dialog is now visible. **Press** the ellipsis (...) button to lookup the Order dictionary.

This dictionary should be visible in the dialog. Notice that you changed to this folder when you made the global DLL in the previous section.

3. You do need a first procedure. The default is named *Main*, but **enter** *MainStates*.

This is not required, but it does help keep procedure names straight.

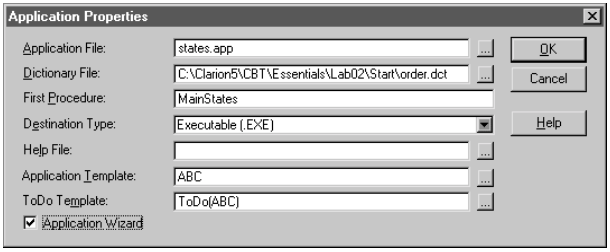
4. The **Destination type** droplist should be set to *Executable (.EXE)* for the moment.

You will change this to DLL later. This is so that you can test the application as an EXE and not have to worry about folding it in another executable for testing purposes. This is covered in more detail later in this lab.

Make sure the **Application Template** is set to ABC.

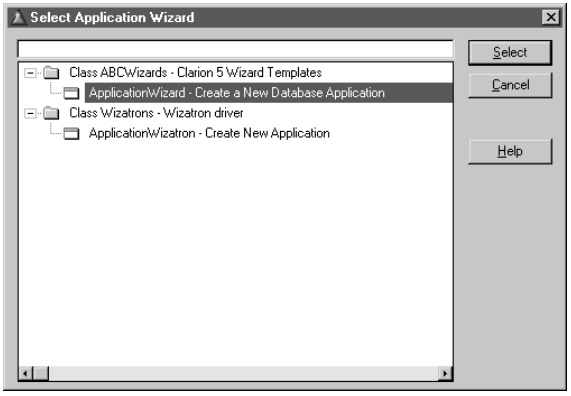
5. **Check** the **Application Wizard** box.

   The dialog now looks like this:



6. **Press** the **OK** button.

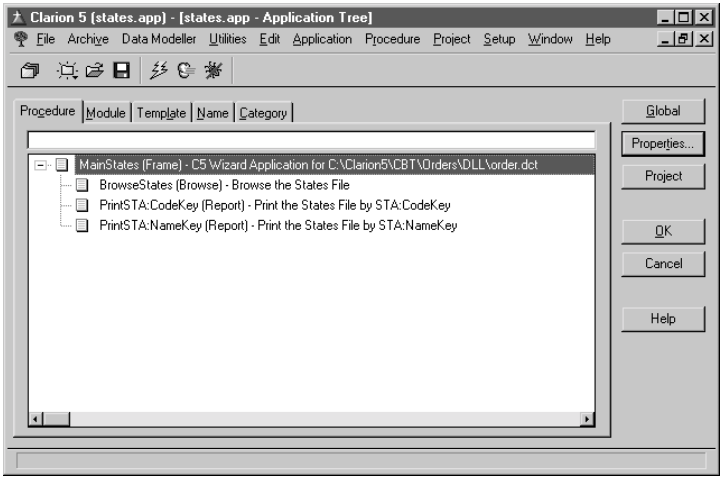
   You are asked which Application Wizard you wish to use. Select the Application Wizard as shown below and press Select. The first wizard is just an intro, **press** the **Next** button.



7. **Clear** the *Generate procedures for all files in my dictionary* box and **press the Next** button.

8. **Highlight** *States* and then **press** the **Finish** button.

   The wizard has all the information it needs. It produced two reports based on the file keys in addition to a browse procedure. Your application looks like this:

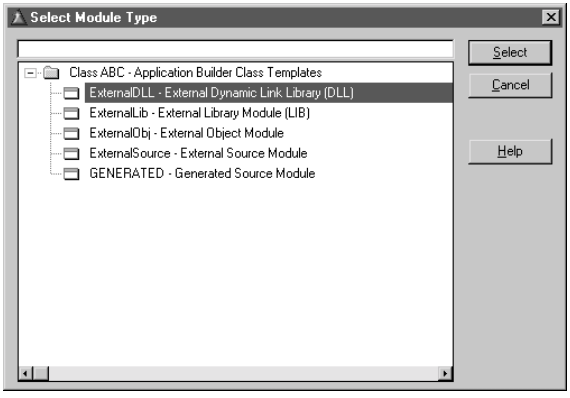


Notice that there is no *UpdateStates* procedure. The *States* file has the User option 'EDITINPLACE' set in the file properties. This means the browse will not only show a list of records, but allow them to be edited directly in the browsebox.

9. **Select Application ➤ View Dictionary** from the menu.

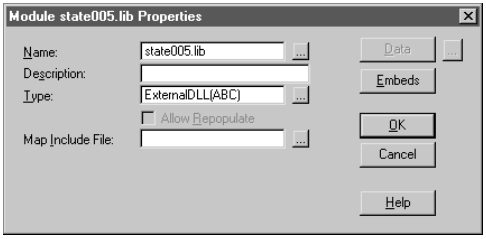
If you DOUBLE-CLICK on the States file and then **select** the **Options** tab, you can see this setting. **Press** the **Close** button when you are done. To move the viewer out of the way, **press** the minimize button. This keeps the dictionary available should you need to refer to it later.

10. You need to make this application aware of the DLL you created in the previous exercise in the lab. **Choose Application ➤ Insert module** from the menu. You see this dialog:



11. **Highlight** *ExternalDLL (External Dynamic Link Library (DLL)* and **press** the **Select** button.

You are presented with this dialog:



12. Press the ellipsis (...) button to lookup the library name. This is always required at this step. Select the file *Global.lib* and **press** the **Save** button.

Clarion generates lib files for all applications and projects, regardless of target type (EXE, DLL or LIB). A LIB file is simply an 'address book' that describes where to find procedures and variables in an external file.

13. Enter an optional description.

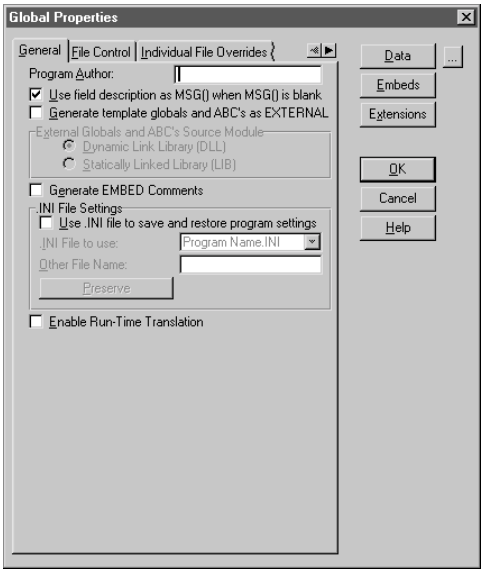
This is always a good idea as it documents what this module is used for. **Enter** *Original data defintions.*

14. Press the **OK** button.

Now this application has the "root" DLL included. However, it does not know anything about it yet. If you compiled the States application, you will get many duplicate warnings from the compiler. This is correct as it sees two different sets of *original* definitions. Since you need to use the dictionary variables and internal variables in *this* application, only one set must be seen as original from the viewpoint of the compiler. We do this with the settings under the Global button.

15. Press the Global button.

This familiar dialog opens:



Remember, we must inform the compiler about the duplicate variable definitions. We need them to get our work done, but the compiler must be happy about it.

16. **Check** the **Generate template globals and ABC's as EXTERNAL** box.

    The group below it becomes enabled allowing you to choose where these variables are located. The default is *Dynamic Link Library (DLL)*. Leave it at this setting. Remember, any future EXE or DLL that requires the Global.DLL *must* have this setting checked.

17. **Select** the **File control** tab.

    Do you need to check the **Generate all file declarations** box? If you recall from the previous section of this lab, the ABC templates generate file declarations *only* when a procedure uses that file. So we know that the *States* file definition is generated. No other file declaration is generated as no other declarations are needed for this application. So the net effect is that there is some added time to the source generation and compile. The bigger the dictionary, the more added time. You would need to have a big dictionary to actually see this.

    Leave this checkbox blank.

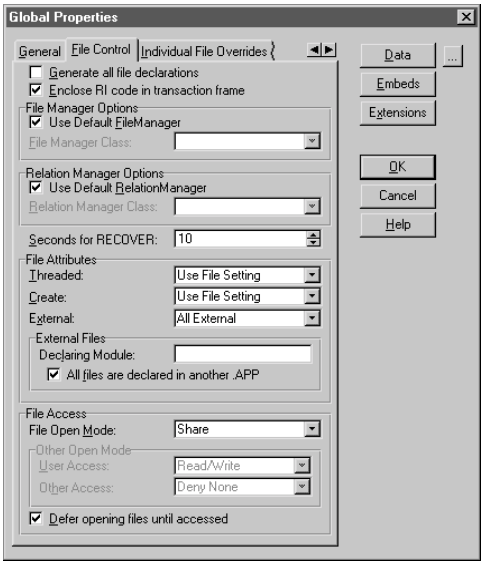
18. Look at the **External** drop list. It is set to *None external*.

    This means is that all the files used in this application are declared *originally* in this application. This is false as the original declarations are in Global.DLL. You need to change this to *All external.*

    When you do this, a new set of prompts displays.

19. Check the **All files declared in another app** box.

This is all you need. The **Declaring module** entry is for the name of the file that has the file declarations. This is provided for hand-coders. Leave this entry blank. Your dialog should look like this:

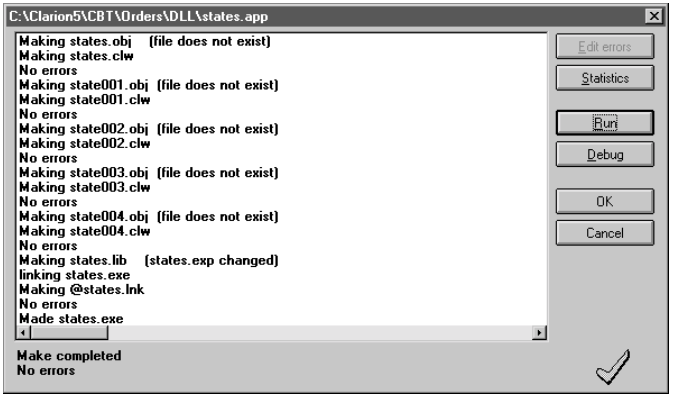


20. Press the OK button.

21. We are now going to make this a 32-bit application. **Press** the **Project** button. This will display the **Project Editor** window. **Press** the **Properties** button. Select **Windows - 32-bit** from drop down listbox for the **Target OS**. **Press** the **OK** button. When you are back at the **Project Editor** window, **press** the **OK** button to save this setting. You should now be back to the application tree.

22. Press the **Save** button on the toolbar and then **press** the **Make** button on the toolbar to compile the States application.

Your compile dialog should look like this if everything is OK:



If you did not get a clean compile, go back and review the previous steps and find out which step was missed. If you did get a clean compile, very well done! Run the application and test it. Add a new state, change it and then delete it. Preview both reports. If you wish, you may print both to your printer.

Let's review what we have accomplished so far:

- We created a new application and inserted a module for the external DLL.
- Changed the global settings so that the compiler will know where the original variable definitions are defined. This eliminates duplicate warning messages.
- Tested the application to ensure it works.

Why did we test the States application as an EXE when it will be a DLL? The reason is that you do not have to fold in the States.DLL into the Orders application to test it. Running as an EXE provides a way to test an application as an EXE first, debugging it as needed. Once everything is in order, you can change the target type to a DLL. This is what we are going to do next.

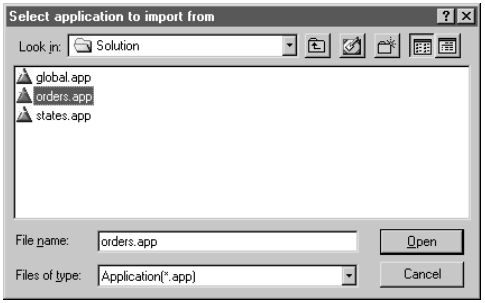
Finishing the States application

There are some small tasks we need to do to complete the States application. Remember, our design calls for any procedure that deals with the States file must reside in this application. Due to the way the wizard works, there is another procedure that needs to be in this application.

However, there is no need to define it as it exists in another application. We need to copy it from another application. You should still have the States application open at this point.

1. **Choose File ➤ Import From Application**.

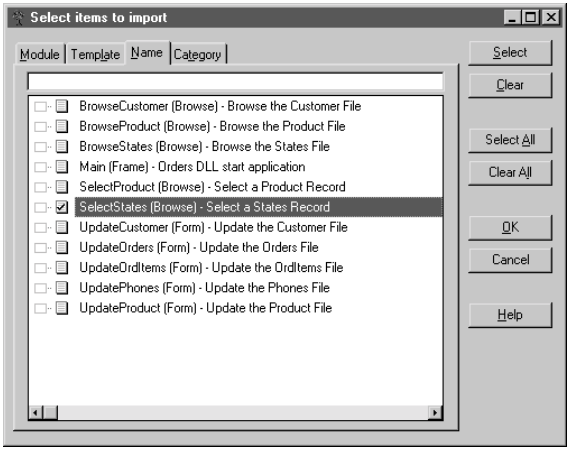
   You see this file dialog:



2. **Select** *orders.app* from the dialog and **press the** **Open** button.
3. **Select** the **Name** tab.

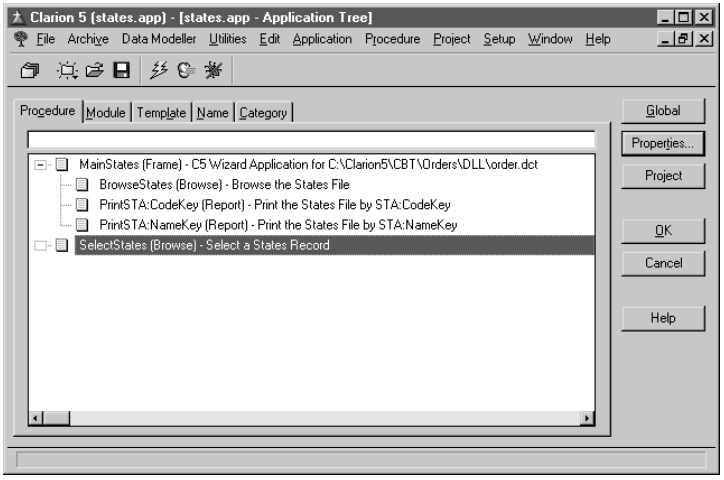
   This makes the list of procedures easier to find.
4. DOUBLE-CLICK *SelectStates* or **highlight** it and **press the** **Select** **button.**

   This adds a small blue checkmark next to the procedure. You may select more than one procedure (which we will do in a later step), but this is the only procedure needed now. Your dialog should look like this:



5. **Press** the **OK** button to import the procedure.

   The States application looks like this:



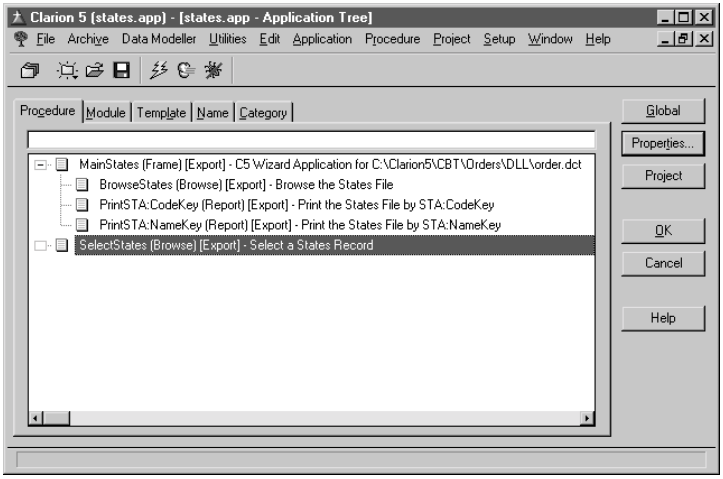
The *SelectStates* procedure is not attached to a parent procedure. This is because no procedure calls it. You could make some extra dummy procedures if you wished to test the functionality of selecting a state. When this application is an EXE, you can test it. However, since this is meant to be a DLL, this is not a problem as calls to this procedure come from outside this application.

The States application needs to be a DLL at this point. All testing proves the application works.

6. **Choose Application ➤ Properties.**

7. **Select** *Dynamic Link Library (DLL)* from the *Destination type* drop list and press the **OK** button.

   Notice the appearance of the tree list changes. You see the word *Export* next to every procedure. This is how the tree should look like when the destination type is a DLL.



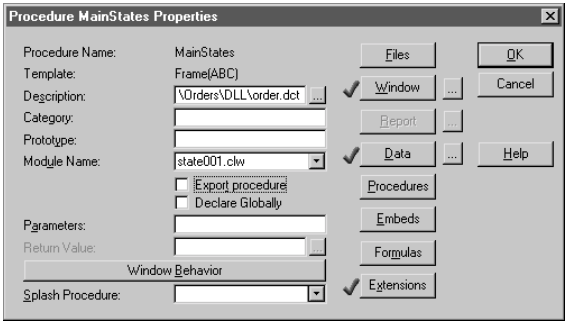
   This means that every procedure in this application is visible or callable from outside the application. Typically, Browse, Report, some Source, and Process procedures are good entry points into a DLL. Menu, Splash, and form procedures generally are not.

   There is only one procedure that should not be exported and that is *MainStates*.

8. **Highlight** *MainStates* and **press** the **Properties button.**

9. **Uncheck** the *Export* box.

   The procedure properties should look like this:



10. **Press** the **OK** button to accept the changes.

    When working on your own applications, you may need to uncheck the export box for more than one procedure.

11. **Press** the **Save** button from the toolbar to save your work so far.

12. **Press** the **Make** button on the toolbar to make the States.DLL.

    You should get a clean compile, meaning that you now made your second DLL! If you did not get a clean compile, review the prior steps and find which one was missed.

The Orders application

In this section, we will incorporate both DLLs into the Orders application. Some of these steps you did in the previous section and are repeated for the Orders application. There are some new steps too.

For this lesson, you should have the starter Orders application open or a wizarded version that built the application based on 3 files; Customer, State, and Products. No reports at this time.

Here's what we will accomplish next:

- Add the States and Global DLL files to the Orders application.
- Set up the Orders application to use external procedure calls and global variables.
- Change the procedures that affect the State file to external procedure calls.
- Import two new procedures from the States application.

1. **Choose Application ➤ Insert Module.**

2. **Highlight** *ExternalDLL - External Dynamic Link Library (DLL)* and **press** the **Select** button.

3. **Press** the ellipsis (...) button to the right of **Name** to lookup the library file.

4. **Highlight** *Global.lib* and **press** the **Open** button.

   The module name is placed in the **Name** entry for you.

5. **Enter** an optional description about this module (recommended).

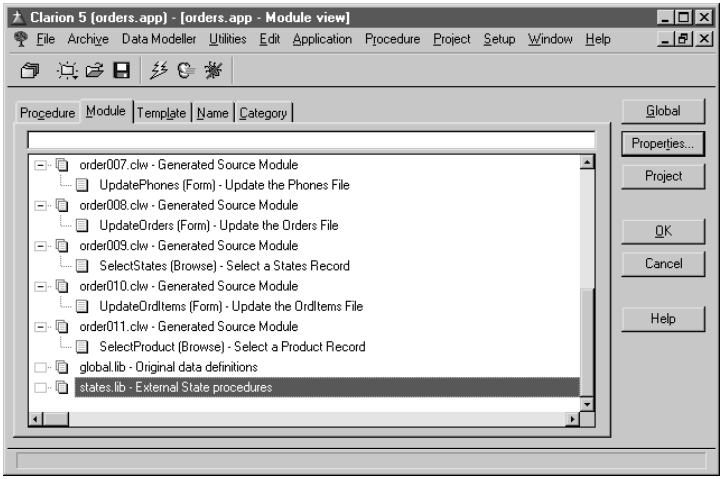
6. **Press** the **OK** button.

7. Repeat steps 1 through 6 for the *States.lib* file.

The Orders application includes two new external modules. But you cannot see them in the current view. There are two ways to verify that everything is in order:

Method One

1. **Select** the **Module** tab and look at the bottom of the list. Your application should look like this:

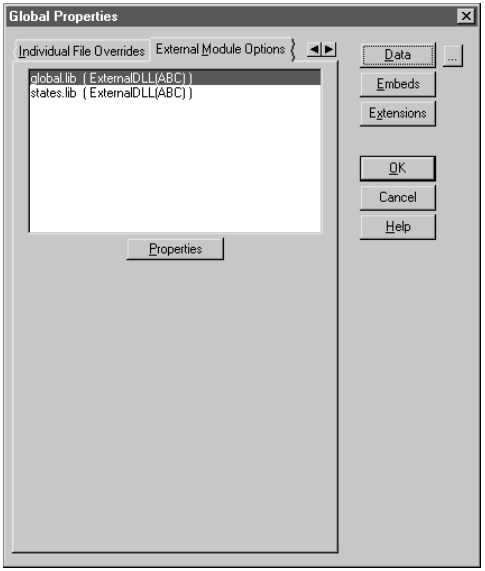


2. **Select** the **Procedure** tab when you are finished.

Method Two

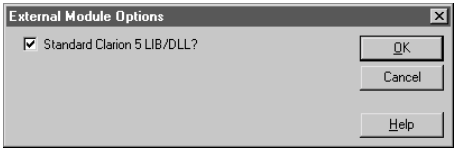
1. **Press** the **Global** button.

   This opens the global dialog. Since inserting the first external module, a new tab is created for you. Just scroll right by clicking on the right scroll button until you see this display:



2. **Press** the **Properties** button.

   This displays a simple dialog with just a checkbox, *Standard Clarion 5 LIB/DLL?*



   Check this box if the LIB or DLL is produced by the ABC Templates or a similar coding scheme. Checking the box generates code to initialize and shut down global objects used by the LIB or DLL. If it is a hand-coded LIB or DLL you should probably clear this box.

3. **Press** the **OK** or **Cancel** button to leave the box checked and close the dialog**.**

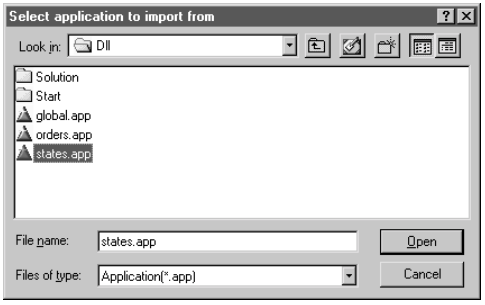
4. **Press** the **OK** button to close the global properties.

Importing two procedures

In order to print a list of states, we need to import the two reports that the wizard created when we made the States application. We will then add the calls to the Main procedure so that a user may run these reports. To start this lesson, make sure the Orders application is open.

1. **Choose File ➤ Import From Application.**

2. **Highlight** *states.app*.

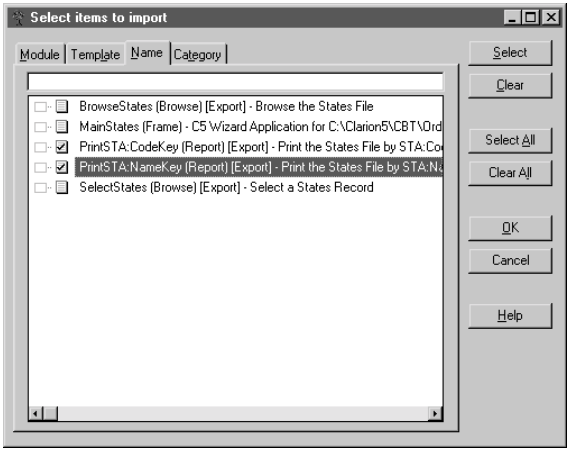


3. **Press Open**.

4. **Choose** the **Name** tab to make the list of procedures you wish to import easier to find.

5. DOUBLE-CLICK on *PrintSTA:CodeKey* and *PrintSTA:NameKey* procedures.

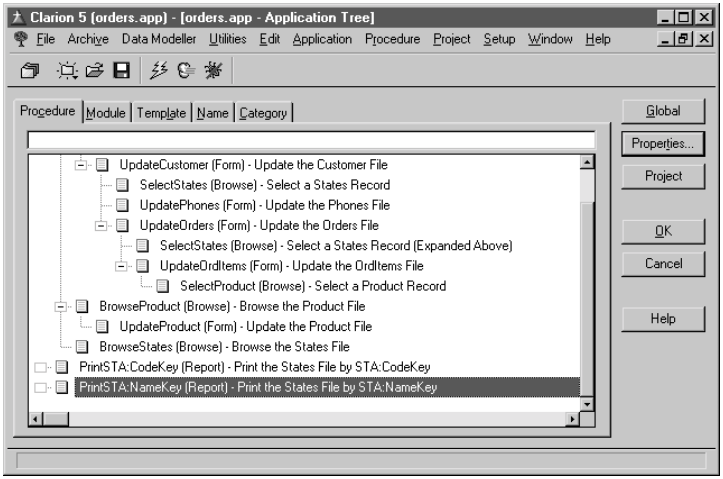
   Remember, if they are selected for import, a small blue checkmark appears next to the names.



Tip: Renaming the procedures generated by the wizard makes it easier for you to remember the procedure names. To rename a procedure, RIGHT-CLICK on any procedure from the application tree and then choose rename from the popup menu.

6. **Press** the **OK** button to import the procedures selected.

   These procedures appear like this after you import them:



They are not attached to any procedure as they are not called in this application. To run these two reports, we need to add some calls to them. This is a straightforward task.

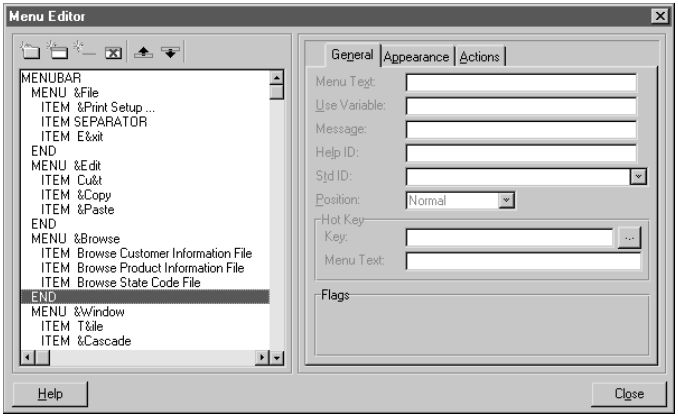
7. RIGHT-CLICK on the *Main* procedure and **select Window** from the popup menu.

This brings up the Window Formatter tool. To add some calls to these report procedures we need to edit the menu.

8. **Choose Menu ➤ Edit menu** or DOUBLE-CLICK anywhere on the white menu bar located on the window.

This loads the menu editor. We need to add another menu.

9. Highlight the END belonging to the Browse menu.



This is the spot where we will insert a new menu.

10. **Press** the **New menu** toolbar button (first button from the left) or **press** SHIFT-INSERT.

    A new menu appears called *Menu&6* in the *Menu Text* entry. This is the text that the end user sees. The default text is not very useful. Since the entire text is highlighted after you add a new menu, it is very easy to change the text to something more meaningful.

11. **Enter** *&Reports* and **press** TAB.

    This not only accepts the text you entered but a Use Variable is created for you. We need to add some menu items now.

12. **Press** the **New item** toolbar button (second button from the left) or **press** INSERT.

    This action enters *Item&16* (or another number depending on how many items exist on the menu) and this text is highlighted, ready for replacement.

13. **Enter** *States by &Code* and **press** TAB.

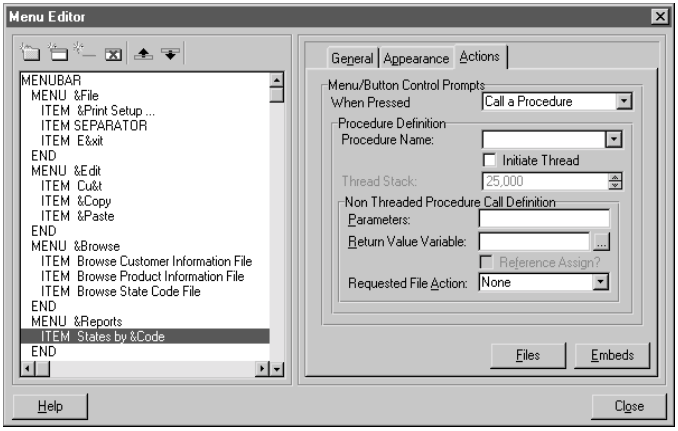
    This not only accepts your entry, but builds a use variable based on the menu name plus the item text (minus the spaces). At this time, all you have is a menu item that does nothing. Its needs an action to perform if a user selects this menu item.

14. **Select** the **Actions** tab.

    The default action is no action.

15. From the **When pressed** drop-down list, choose *Call a procedure*.

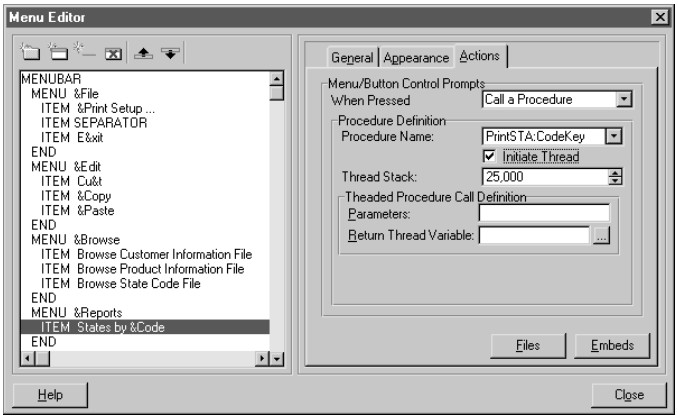
    This dialog displays:



16. **Select** *PrintSTA:CodeKey* from the **Procedure name** droplist.

17. **Check** the **Initiate Thread** box.

This is required as the report is an MDI child procedure. The dialog now appears like this:

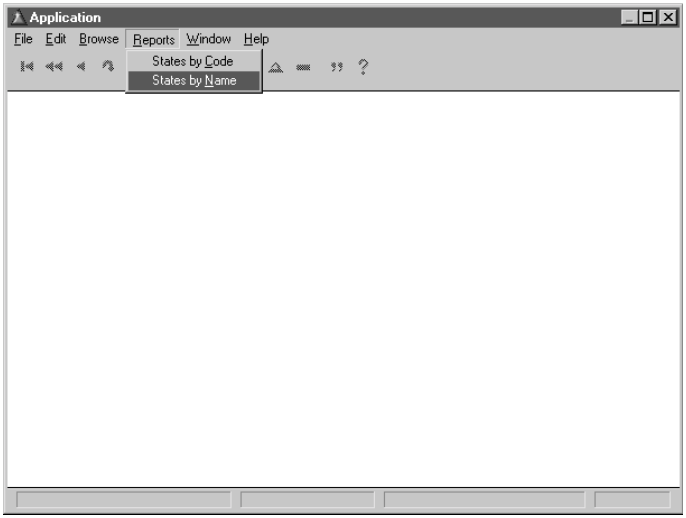


18. **Press** the **New item** toolbar button (second button from the left) or **press** INSERT.
19. **Enter** *States by &Name* and **press** TAB.
20. **Select** the **Actions** tab.
21. From the **When pressed** drop list, choose *Call a procedure*.
22. **Select** *PrintSTA:NameKey* from the **Procedure name** droplist.
23. **Check** the **Initiate Thread** box.
24. **Press** the **Close** button.

    You now see the Reports menu on your window.

25. **Press Preview!** to see how the window will look when the application is running.

It should look like this:



26. **Press E**sc to quit or exit from preview mode.

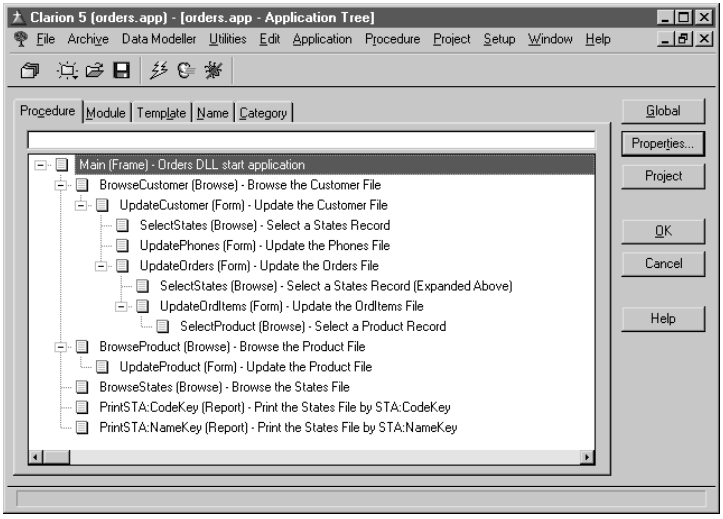
27. **Press Save** (green check button on the toolbar) to save your edits.

    You are returned to the application tree. If you opened the window formatter via the procedure properties dialog, you need to **press** the **OK** button to return to the application tree.

Tip: **While you are in the window formatter for the menu, you may wish to enhance the window. For example, the text for the window could say "Orders Application" instead of just plain "Application".**

Other enhancements could be that the window opens in maximize mode instead of normal (windowed) mode, adding wallpaper, a splash procedure, setting a different font, colors, etc..

The two imported print procedures are attached to the Main procedure. This means that Main calls these two print procedures. The application appears like this:



Tying it all together

There is just few more steps needed to finish this application for use with the other DLLs. As it exists now, it will not compile clean.

Here is what we need to do in this section:

- Make the state procedures external procedures.
- Change the same areas in the global area of the Orders application that we did with the States application.
- Test the executable.

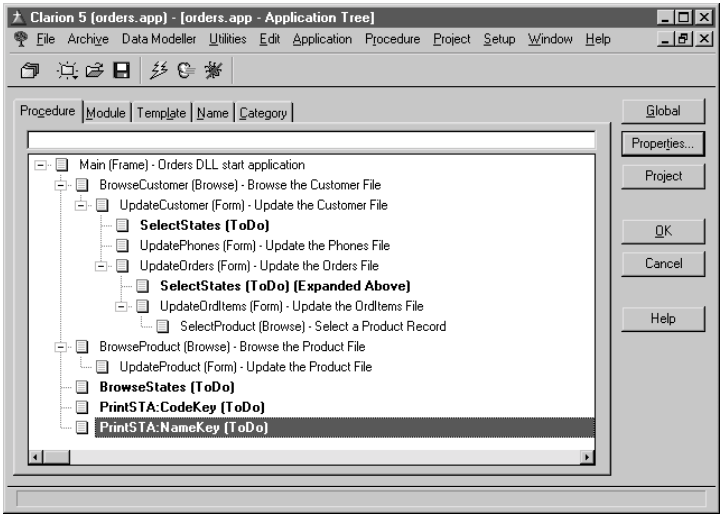
1. **Highlight** each state procedure and delete them.

Once the *BrowseStates*, *SelectStates*, *PrintSTA:CodeKey*, and *PrintSTA:NameKey* procedures are deleted they appear as *ToDo* procedures in the application tree. The reason we went through all the trouble importing the two reports, adding them to the menu is so that we can delete them. The reason is that you cannot have an unattached *ToDo* procedure. If you deleted these procedures before attaching them to a procedure (like we did with the *Main* procedure), you will have to import them again

.

Note: You can add procedure calls to the menu before you import from another application. Just enter the name of the procedure in the Call Procedure drop list instead of selecting it. It will appear as a ToDo item until you import it.

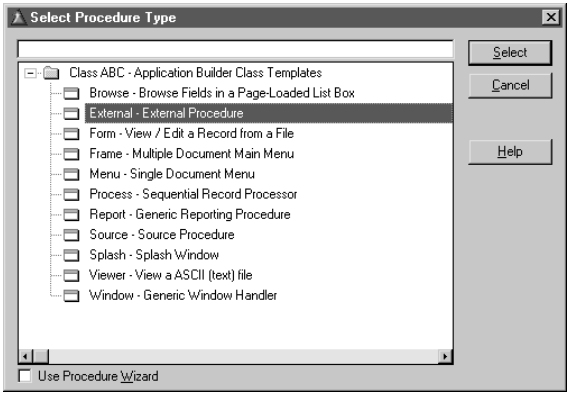
This method is more prone to naming errors, thus in this lab the import is done first. If you misname the procedure and then try to call a procedure in a DLL that does not exist, the application will compile clean. But you get a error at runtime when you try to call the procedure that is mispelled.



The next steps are very easy to do.

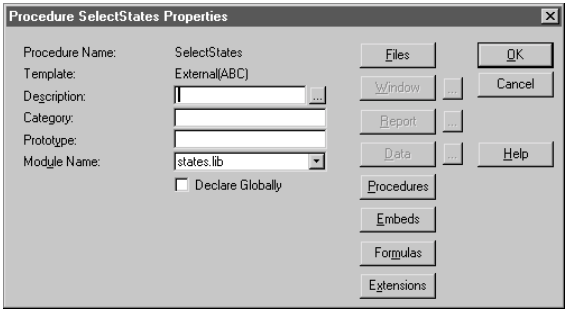
2. DOUBLE-CLICK, **press ENTER** or **press** the **Properties** button on any *ToDo* procedure.

You will see this dialog asking for the procedure type:



3. **Select** *External - External Procedure* from the list. Make sure the **Use Procedure Wizard** box is clear. **Press** the **Select** button.

This dialog displays:

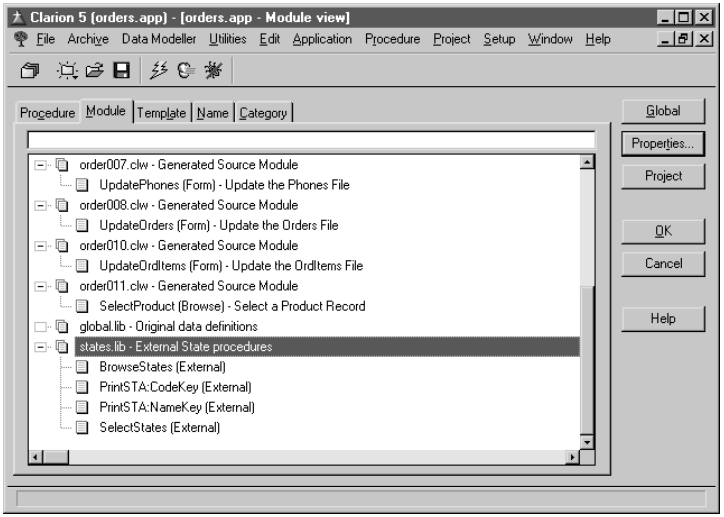


4. **Press** the **OK** button.

The only thing you must be sure of is that the module name is filled in correctly. In this case, *States.lib* is the correct name. Repeat the above steps for each remaining *ToDo* procedure.

Note: **You can use this technique to add multiple library names not just one. Just ensure that the Module Name is correct. You can think of this as the "home" where the procedure lives.**

5. Switch to the module view and look at the States.lib module.



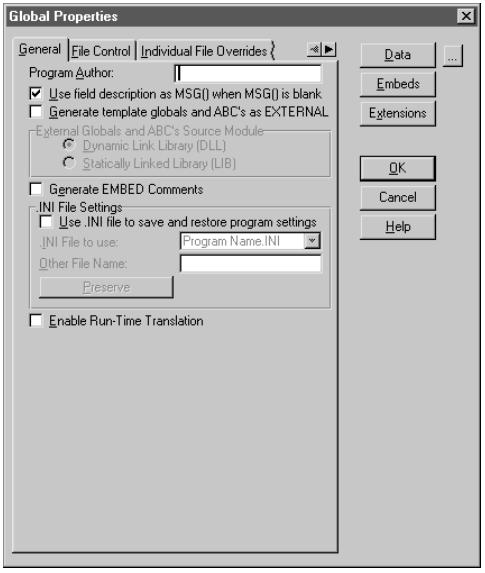
Notice the procedures that are now attached to the *States.lib* module. Also notice there are no procedures attached to the *Global.lib* module as it does not have any procedures.

The final steps

Just a few more steps and we are done. All that remains is setting the global switches correctly. The same steps you previously did in the states application need to be repeated here. This tells the compiler where to find everything it needs to make the executable work. Here are the repeated steps.

1. **Press** the **Global** button.

   This familiar dialog opens:



Remember, we must inform the compiler about the duplicate variable definitions. We need them to get our work done, but the compiler must be happy about it.

2. **Check** the **Generate template globals and ABC's as EXTERNAL** box.

   The group below it becomes enabled allowing you to choose where these variables are located. The default is *Dynamic Link Library (DLL)*. Leave it at this setting. Remember, any future EXE or DLL that requires the Global.DLL *must* have this setting checked.

3. **Select** the **File control** tab.

   Do you need to check the **Generate all file declarations** box? Remember, the ABC templates generate file declarations *only* when a procedure uses that file. In this case, since there are procedures that use each file in our dictionary, it makes no difference whatsoever if you check this box or not. All file definitions are generated.

   Leave this box blank or check it, your preference.

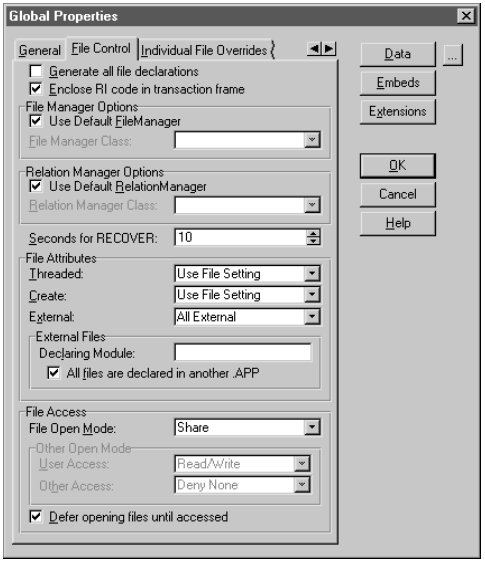
4. Look at **External** drop list. It is set to *None external*.

   What this means is that all the files used in this application are declared *originally* in this application. This is false as the original declarations are in Global.DLL. You need to change this to *All external.*

   When you do this, a new set of prompts displays.

5. **Check** the **All files declared in another .APP** box.

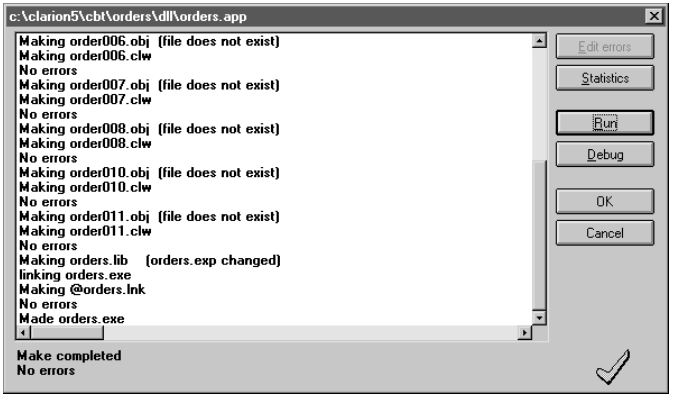
   This is all you need. The **Declaring Module** entry is for the name of the file that has the file declarations. This is provided for hand-coders. Leave this entry blank. Your dialog should look like this:



6. **Press** the **OK** button.

7. **Press** the **Save** button on the toolbar and then **press Make** on the toolbar to compile the States application.

   Your compile dialog should look like this if everything is OK:



   If you did not get a clean compile, go back and review the previous steps and find out which step was missed. If you did get a clean compile, very well done! Run the application an test it. Add a new state, change it and then delete it. Preview both reports. If you wish, you may print both to your printer.

   If it does not look like anything special, that is exactly what is supposed to happen! You should not be able to tell if a procedure is in the current application or not.

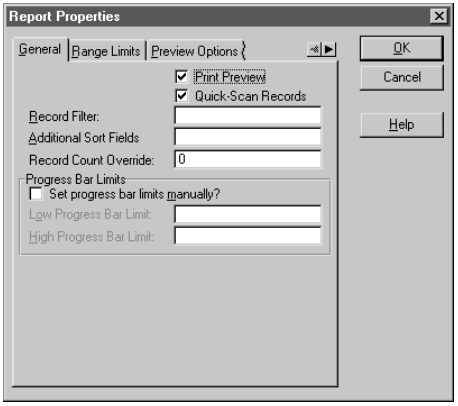
Deployment exercise

This is a fast lab exercise to show how you could deploy a changed DLL. In this lab we make a simple change to one of the reports.

For this exercise, we need to close the Orders application and load the States application.

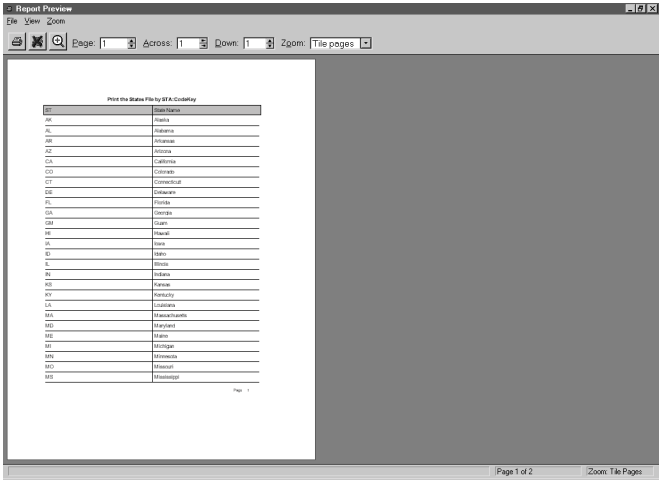
1. **Highlight** the PrintSTA:CodeKey procedure.
2. **Press** the **Properties** button.
3. **Press** the **Report Properties** button.

   This dialog displays:



4. **Select** the **Preview Options** tab.
5. **Check** the **Maximize Preview Window** box.
6. **Press** the **OK** button twice to return to the application tree and save your changes.
7. **Press** the **Make** button. **Press** the **OK** button when you get a clean compile.

   At this point, the *States.DLL* is 'deployed'. Launch Windows Explorer and navigate to the *X:\Clarion5\CBT\Essentials\Lab02\Start* folder where your *Orders.EXE* program is located. DOUBLE-CLICK on *Orders.EXE* and run the *States by Code* procedure. Your report should look like this:



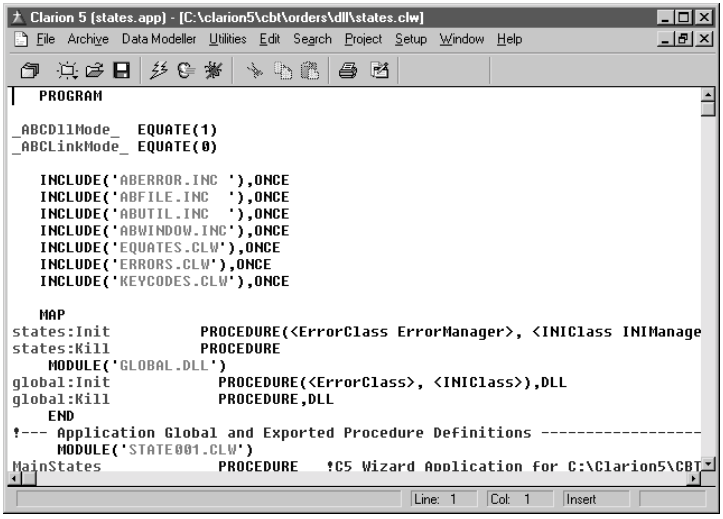
8. When you are finished examining the report quit the running application.

Source code

This last exercise takes a tour of the generated code. There is some Clarion code generated that makes DLLs work. You may have either the Orders or the States application open to start this exercise.

1. **Select** the **Module** tab to switch to module view.
2. **Highlight** *States.clw* (or *Orders.clw* if you have the Orders application loaded).
3. RIGHT-CLICK and **select Module** from the popup menu.

   This opens the editor and loads the source for the module. This is the source you should see:



   There is one equate, *_ABCDllMode_*, that is applicable to DLLs. It is simply a flag used in ABC objects.

   *_ABCDllMode_* is used as the flag in the DLL attribute. The DLL attribute specifies that the declaration (this may any variable dclaration, or a FILE, QUEUE, GROUP, or CLASS structure) on which it is placed is defined in a .DLL.

4. Scroll down until you find the first file declaration.

   Scroll to the right until you can see all the attributes of the file. It will look similar to this:

LAB 3

Reports

Introduction

What We Are Going to Accomplish

In this Lab, we'll make two reports. One very simple and the other which uses multiple files and group breaks. We will cover an easy way to build these reports We will also take a tour of the report formatter features. Many of these features are easily overlooked, but are vital to creating good reports.

The Data Dictionary, Data Files, and Application are supplied as a starting point to this lab. It is strongly recommended that you have studied the Language, OOP and source embed topics before doing this lab. This lab assumes that you have completed these lessons as it builds on them.

In the first report we will:

- Tour the report formatter features.
- Use one data file.
- Make 3-up mailing labels.
- Learn techniques to find unknown ABC embed points.

The second report we will:

- Use multiple data files.
- Add group breaks.
- Add an image.
- Do totalling.

Overview

Clarion's report capabilities are flexible and the report template could not be simpler. It takes advantage of the Report Engine's features.

Exercise Starting Point

You should have just opened the Clarion development environment. The Pick dialog should be open.

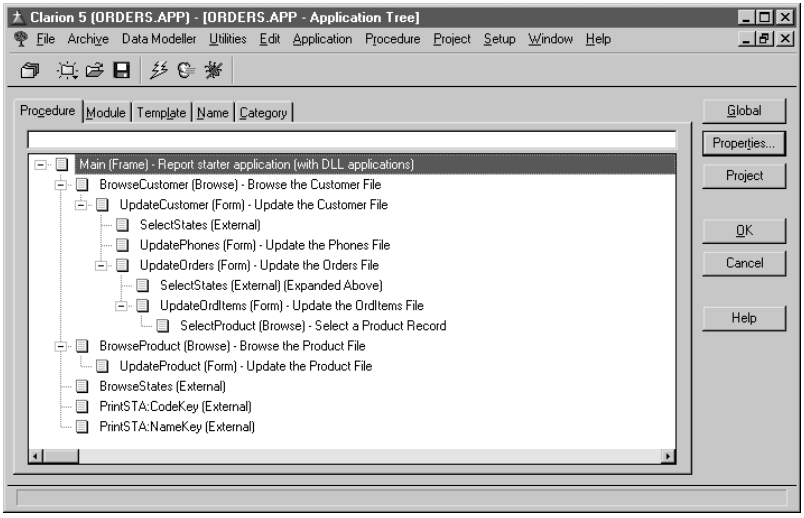
From within Clarion:

1. **Press** the **Open** button.
2. **Navigate** to the *X:\Clarion5\CBT\Essentials\Lab03\Start* folder.

> **Note:** **If at anytime you wish to restart this lesson, copy the contents in the *.\Lab03\Start* folder. If you wish to compare how you did in this lab, look in the *.\Lab03\Solutions* folder. There are two solution folders, *.\Lab03\Solutions\Invoice* and *\Lab03\Solutions\TwoLabels* as there are two reports in this lab.**

3. **Select** *Orders***,** make sure the **File of Type** is *Application (*.app)***.**
4. **Press** the **Open** button.

The Application Tree displays.



A Simple Report

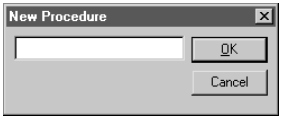
Add a new procedure

In the previous lab, we added new procedures by importing from another application. There are two other ways to add a new procedure. By the time you finish this lab, both methods will be used.

We'll start by inserting a new procedure.

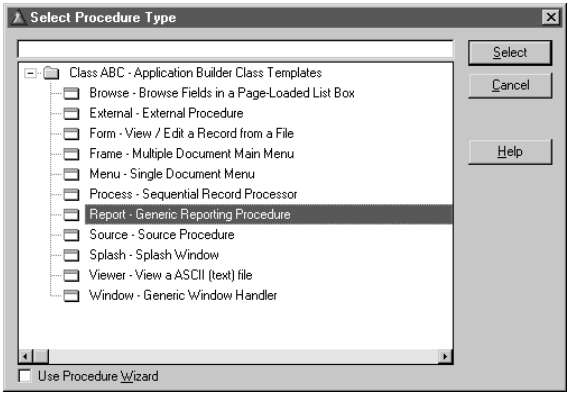
1. **Press** INSERT.

   This displays a dialog prompting you for the procedure name.

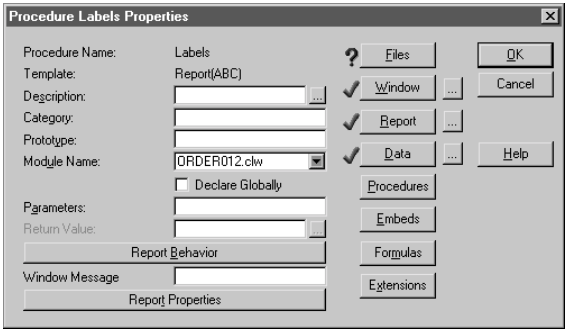


2. **Enter** *Labels* and **press** the **OK** button.

The S*elect Procedure Type* dialog displays.



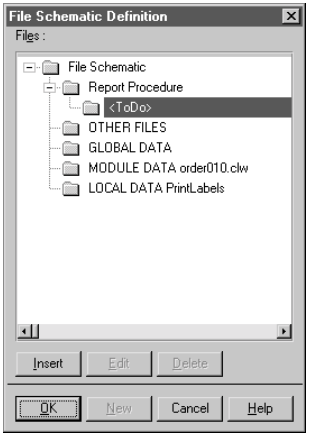
3. **Highlight** *Report - Generic Reporting Procedure* and **press** the **Select** button. Make sure the **Use procedure wizard** box is clear.



4. **Enter** *Mailing Labels* in the **Description** entry field.

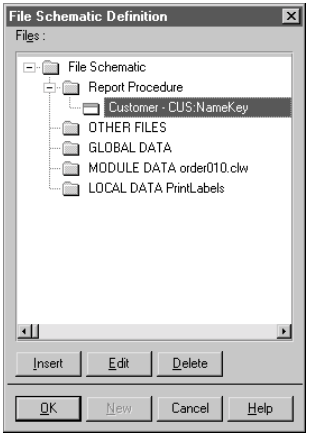
5. **Press** the **Files** button.

   The red question mark means that a required entry is missing. The file schematic dialog is shown.



   DOUBLE-CLICK on the ToDo section or **press** INSERT. From the list of files, **select** *Customer*.

6. **Press** the **Edit** button and select *NameKey*.



7. **Press** the **OK** button to close the dialog and the red question mark becomes a green check**.**

8. **Press** the **OK** button to close the procedure dialog.

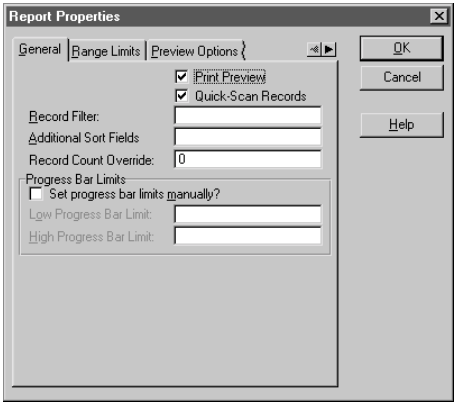
   The Labels procedure is 'orphaned' as no other procedure in our application calls it yet. We will add the call to this later. Before proceeding to the next section, save the work done so far.

Report Properties

1. Open the procedure property dialog for *Labels* again.

2. **Press** the **Report Properties** button.

   The Report Properties dialog displays:

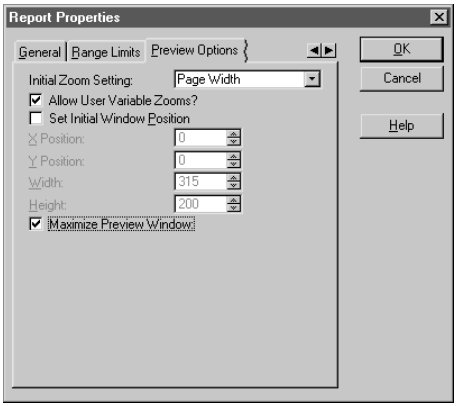


3. **Select** the **Preview Options** tab.

4. **Select** *Page Width* from the **Initial Zoom Setting** droplist.

5. **Check** the **Mazimize Preview Window** box.

These settings make the preview window full screen and the report is zoomed in so that it is easy to read.



6. Press the **OK** button twice.

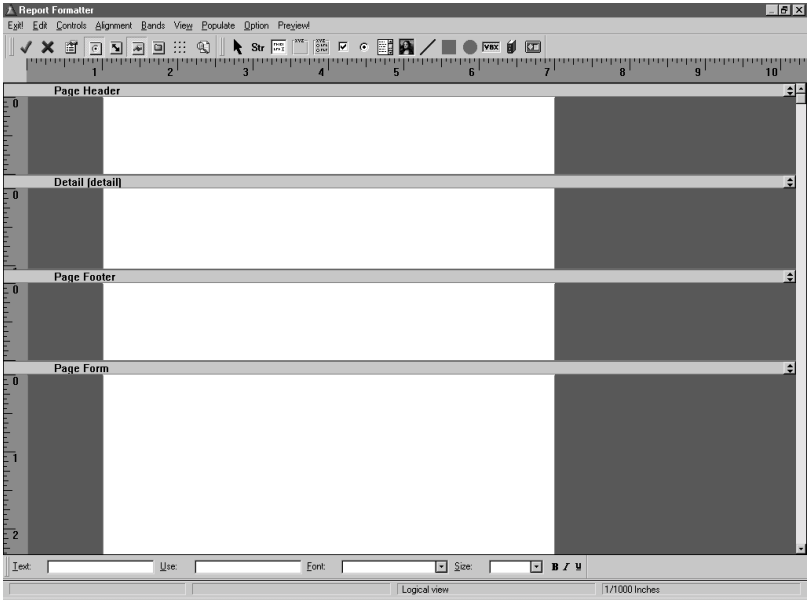
You are returned to the procedure properties dialog, then to the application tree. Save the application.

Setting up the Report

In this section of the lab, we will take a small tour of some features of the report formatter. This section guides you through some seldom used features that make your reports look good. During the course of this lesson, you can see how reports are easy to maintain. We will use some ABC embed points, even if we do not know where they are!

1. The procedure, *Labels* should be highlighted. RIGHT-CLICK and select *Report* from the popup menu.

This opens the Report Formatter. It is here that most Clarion developers start dropping controls on their reports. However, this creates more work for you as the default report is really not set up yet.



2. **Position** your mouse in the *Page Header* band, RIGHT-CLICK and choose **Delete** from the popup menu.

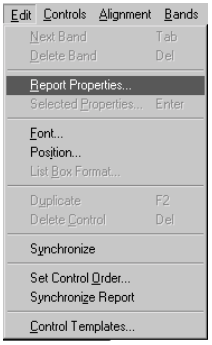
   Since we are making mailing labels, we do not need a Page Header band.

3. **Delete** the *Page Footer* band in the same manner.

   We do not need a page footer either. The only bands left are the *Detail* and *Page Form*. You do not need the Page Form either, but it will not affect our labels. It does seem the help when paying out your report. More on that shortly.

4. **Choose Edit** from the Report Formatter menu bar.

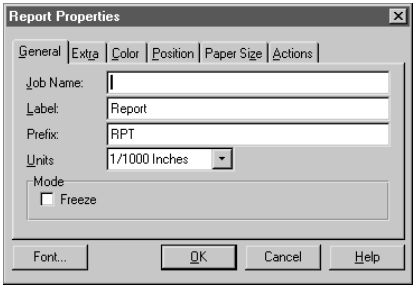
   You see this submenu.



   This is the menu you should become very familiar with. Many useful tools are found here.

5. **Select Report Properties** from the menu.

   This dialog displays:



6. **Enter** *Mailing Labels* in the **Job Name** entry.

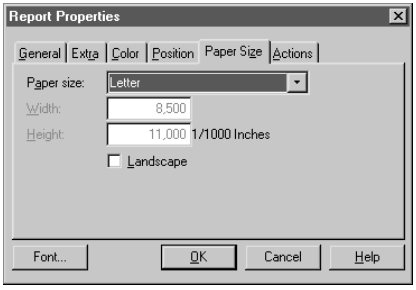
   This is optional, but this the the description of the report when it is in the printer spooler. If this is left blank, the description of the report in the spooler is blank.

7. **Select** the **Paper Size** tab.

   Many reports are special types of reports. Most are not meant to fit on standard paper sizes such as Letter or A4. The default paper size is set for *Other*, but it really means “custom size”. Enter the values of the size of your sheet in the unit of measure, which is displayed here. If you need to change it, go back to the **General** tab and pick the unit of measure for your location.

   If you wish to use a standard US paper size, such as letter, you could enter *8,500* for the width and *11,000* for the height. This corresponds to 8.5 by 11 inches, with each dimension multiplied by the scale show (1/1,000th of an inch).

8. While still in the Paper Size drop list, simply type the letter *L*, which fills in the height and width entries for you.



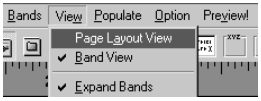
9. **Press** the **Font** button.

This opens the Windows font dialog. **Choose** a *Courier New* **Font**, *Regular* **Font Style**, and a **Size** of *8* in the dialog. You see a visual representation of the font. **Press** the **OK** button to close the font dialog.

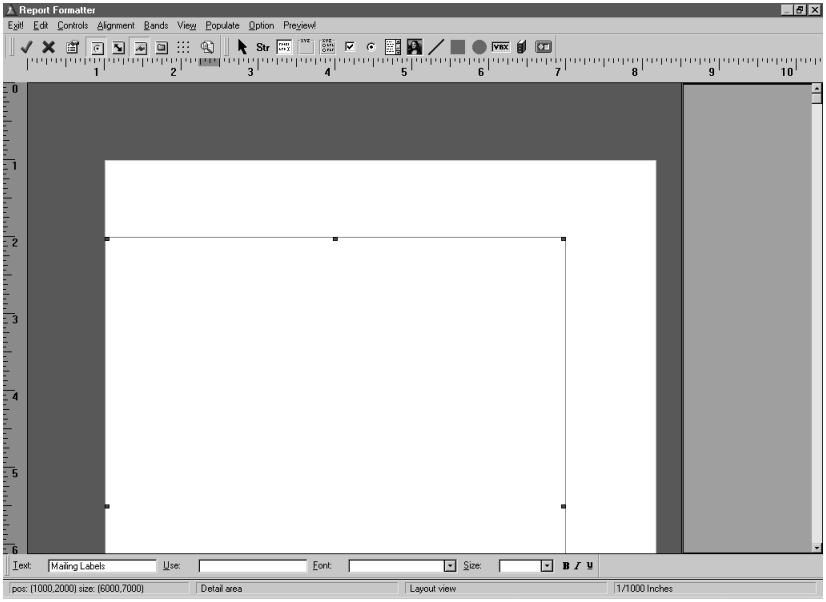
Tip: Always choose a font when defining a new report. When previewed, they look very close to how they print.

10. **Press** the **OK** button to close the *Report Properties* dialog**.** You are returned to Report Formatter.

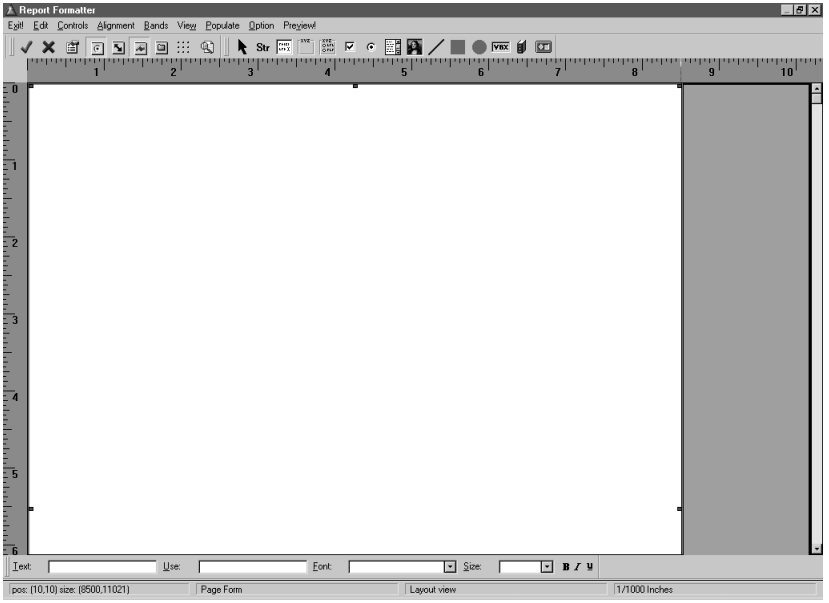
11. **Choose View ➤ PageLayout View** from the Report Formatter Menu.



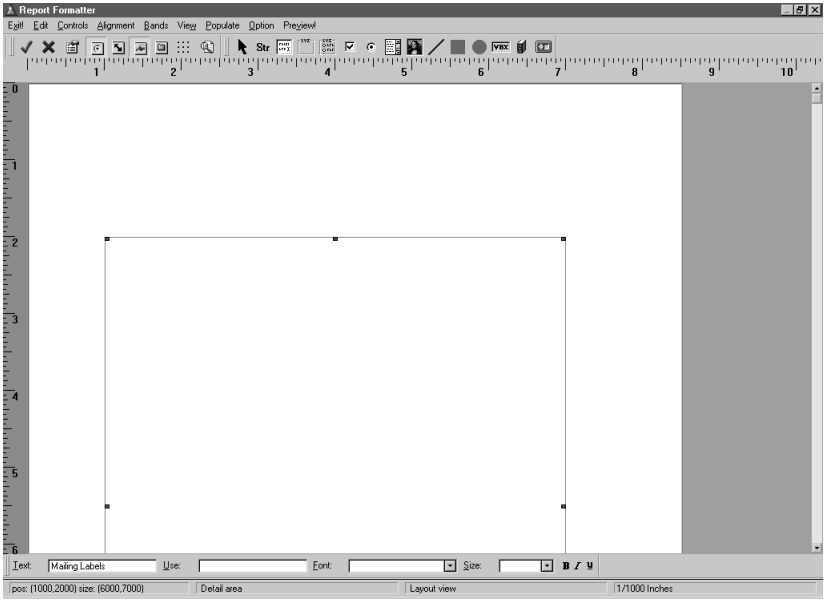
This shows where the data in your report will print.



The default image has the detail area selected (see the status message box at the bottom). Click outside this area to select the Page Form. Using the corner handles, stretch the corners until there is no blue background showing. It should look like a big piece of paper (a good reason to keep it on your reports).



12. Press TAB to select the detail area.



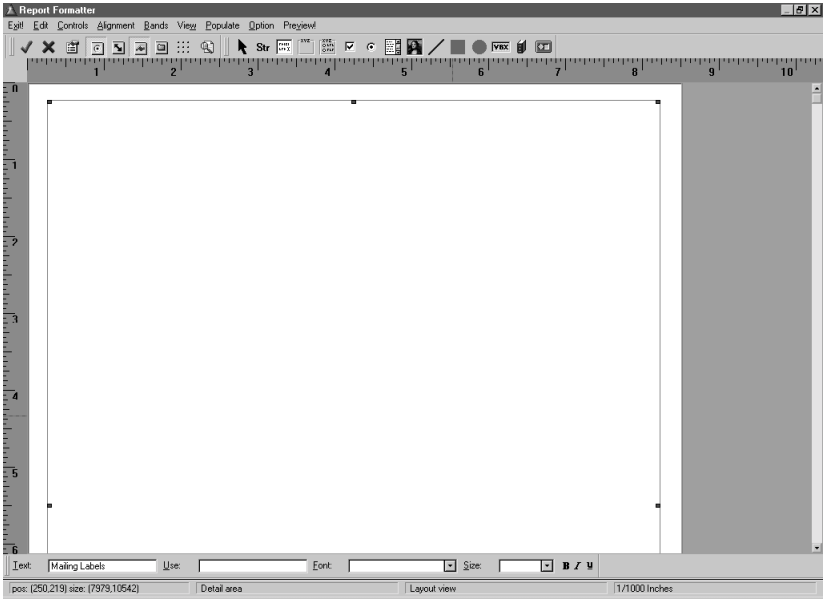
The detail area is where your detail band prints. It never prints outside this area.

Note: Any group headers and footers are parts of the detail band to which they are attached to. They always print in the detail area too.

Since we are making a label type report, there is wasted space. Simply drag the corners of the detail area and leave about a 1/4 inch gap on all sides.

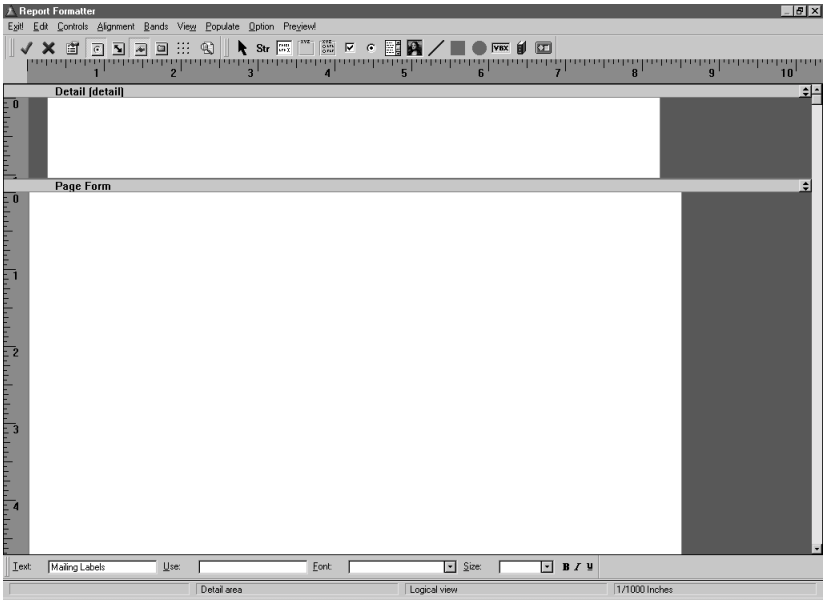
Tip: Use the rulers to measure how tall and wide you wish to make this area.

Your area should look like this:



13. **Choose View ➤ Band View** from the Report Formatter Menu to return to the band view.

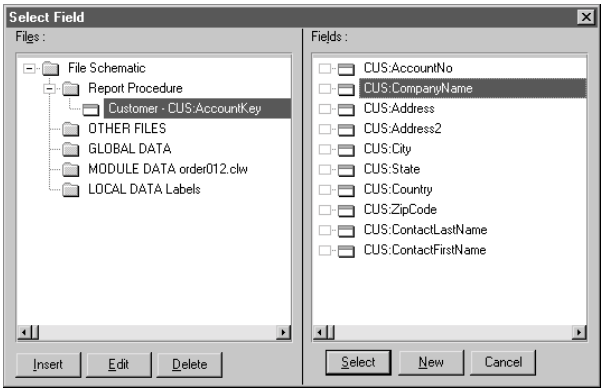
Notice the image aligns with the changes you made.



Populating controls

It is at this point you are ready to drop controls on the report.

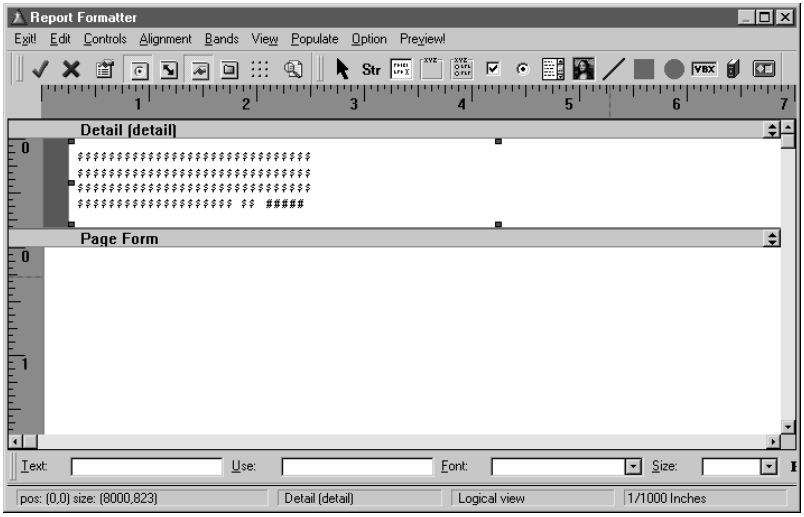
1. **Choose Populate ➤ Multiple fields** from the Report Formatter Menu.
2. **Select** *CUS:CompanyName*.



3. **Move** the mouse cursor over the detail band and you notice it changes to a crosshair. Place it close to the upper left hand corner and LEFT-CLICK to drop the control.

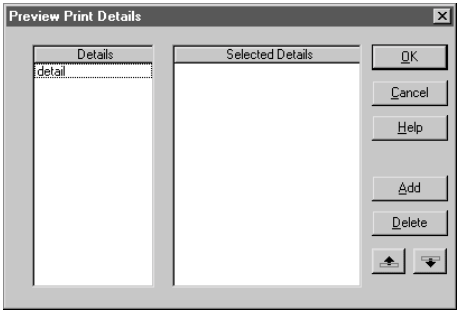
4. The dialog appears again, **select** *CUS:Address* and drop this control under the previous control.
5. **Select** *CUS:Address2* and drop this control under the last.
6. **Select** *CUS:City* and drop this under the *CUS:Address2* control.
7. **Select** *CUS:State* and drop this to the right of *CUS:City*.
8. **Select** *CUS:ZipCode* and drop the to the right of *CUS:State*.
9. **Press** the **Cancel** button.

   You have controls that more or less look like a label. Use the alignment tools and drag controls, where needed, to improve their appearance. You may find that the detail band height is too big. Using the bottom handle, drag it to close it up. Your report should look similiar to this:



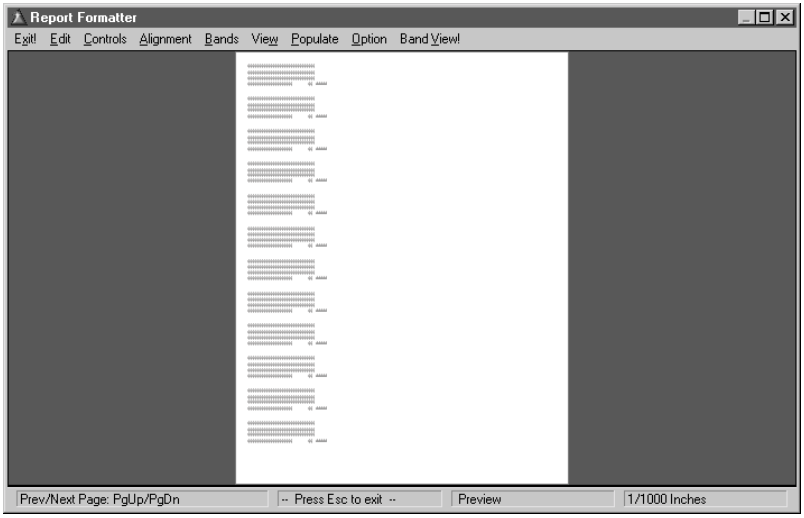
10. **Press Preview!** from the Report Formatter Menu.

    You see the preview dialog.



11. **Highlight** *detail* and **press** the **Add** button about 15 times to add a few detail lines. **Press** the **OK** button.

    The preview should look similar to this:

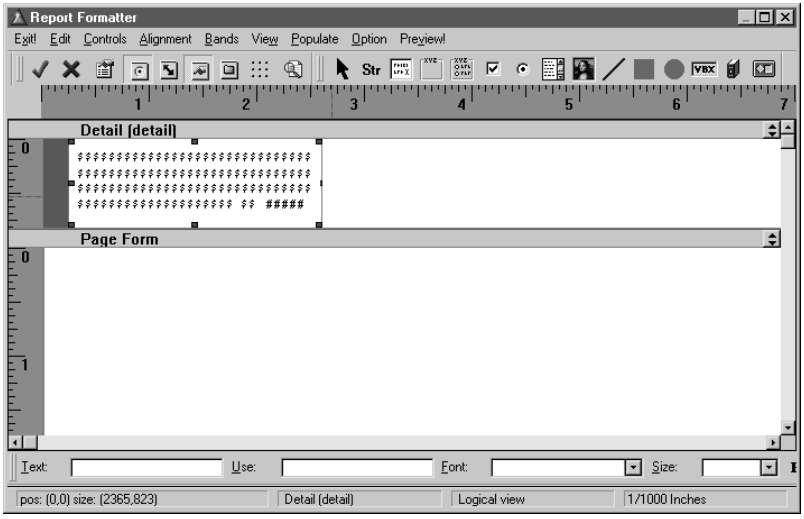


12. When done, **press Band view!** from the Report Formatter Menu.

> **Tip: If you add enough detail lines to overflow a page, you can use the PgDn and PgUp keys to test page overflow.**

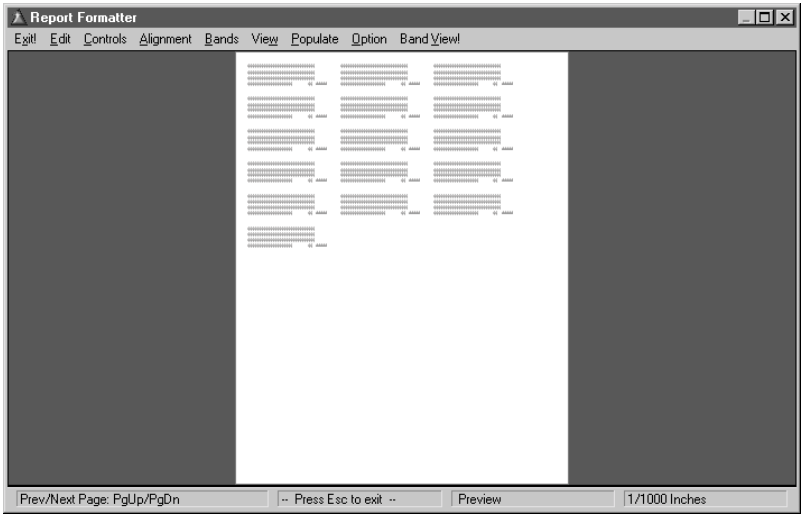
So far, so good. However, there certainly is room for multiple column labels, possibly even 3-up labels. This is easy to do in the report formatter. Remember that you have a detail area (as laid out in the layout view earlier) and there is a width of this detail band. It even printed the white space of the detail band. You may have guessed how to make multi-up labels now.

13. **Select** the detail band. **Grab** the far right middle handle and drag it to the left to make the detail band narrower. Stop the adjustment when the right border of the detail is just to the right of the controls.



14. Press Preview! from the Report Formatter Menu.

Notice the last preview setup is remembered. **Press** the **OK** button.



Do you have similar results?

15. Exit and save your work.

You are returned to the application tree.

16. Save the application.

Covered so far

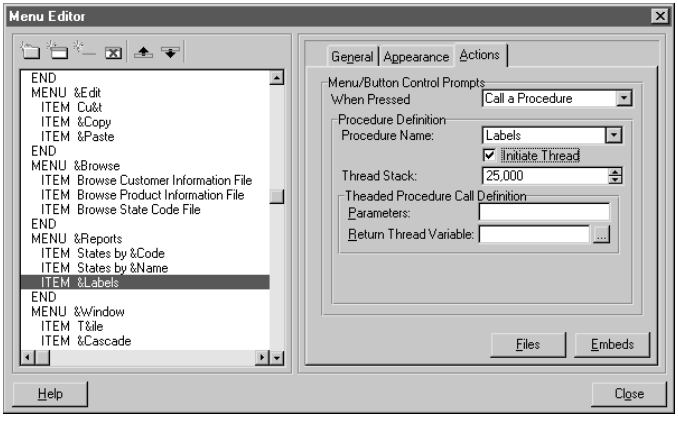
This is a summary of what we covered:

- Adding a new procedure.
- Setting preview properties.
- Setting up the report with fonts, paper size and setting the job name.
- Deleting uneeded bands.
- Adjusting the layout of the report.
- Populating multiple controls at once.
- Using the preview feature.
- Making columnar labels.

Test the label report

In this next section we need to attach the report to the menu so we can run it. We also want to ensure that everything works expected. The Orders application should be open as a starting point.

1. **Highlight** *Main* and RIGHT-CLICK to popup its menu. **Choose** *Window*.
2. **Open** the menu editor and add a new menu item to the report menu. **Enter** *&Labels* for the menu item text. On the **Actions** tab, **select** *Call a procedure* for the **When Pressed** droplist. **Select** *Labels* as the procedure to call and **check** the **Initiate thread** box.

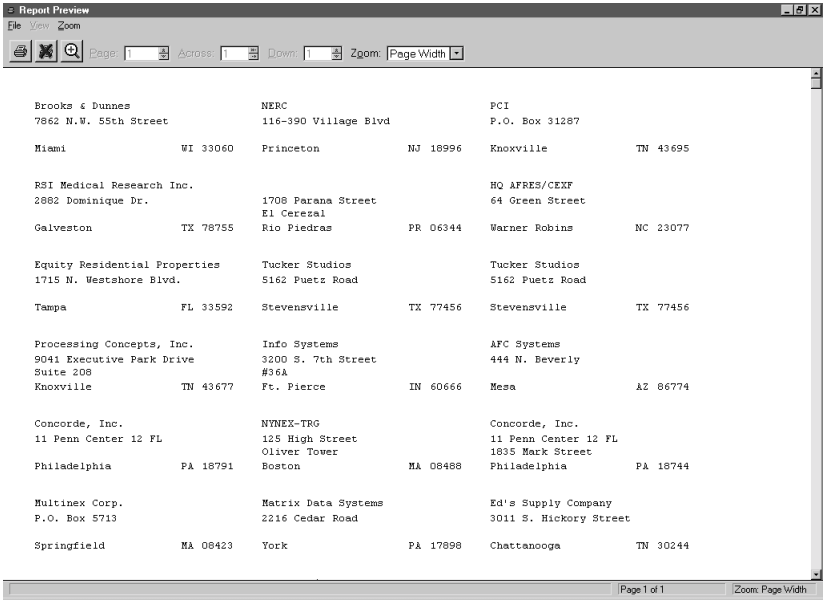


3. **Press** the **Close** button to save the edits in the menu editor and then **press** the **save and exit** button on the window formatter toolbar.

   You are returned to the application tree and the *Labels* procedure is in the tree. Save the application.

4. **Press** the **Run** button from the toolbar to generate, compile and run the application.

5. **Run** the labels report. It should appear similar to this:



These labels are functional, but they do not look pretty. What we need to do to fix this report? Here is a list of problems:

- If there is no company name, it should print the contact first and last name.
- Some labels have a second address, others do not. We need to close the gap without ruining the label alignment.
- The City, State and Zip Code fields do not have to be in fixed positions.

6. When you are done viewing the report, close the print preview window and exit the program.

Embedded Code

Problems such as the above usually mean we need to write some code. And we do. To make matters worse, this is ABC and we do not have any idea where the code should be placed.

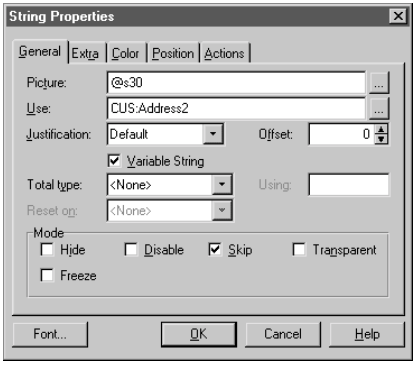
However, one of the above is easily handled without writing any code! Lets do that one first.

1. **Open** the report formatter for the *Labels* procedure.
2. **Highlight** *CUS:Address2*.

> **Tip:** **If you have the property toolbox turned on, you can easily see which control is highlighted. To give yourself more room on the report formatter worksheet, dock this toolbox to the top or bottom of the formatter.**

3. **Press** the **Properties** button.
4. **Check** the **Skip** box.

   This puts the SKIP attribute on this control. The SKIP attribute on a report control specifies the STRING or TEXT control prints only if its USE variable contains data. If the USE variable does not contain data, the STRING or TEXT control does not print and all controls following in the band "move up" to fill in the space. This is most useful for label printing to prevent extra blank lines in addresses.



5. **Save** your work and run the report again.

   Notice the second address issue is solved perfectly. The nice thing is that we did not have to write any code to handle this.

   For the remaing two issues, we do need to write some code. Some can visualize the code needed to handle our other two situations. The real issue is where do we place it? If one has no clue as to where it goes, there is help. All that is really needed is to “ask the IDE” where to place your code. One of the best tools to do this is to use the Formula Editor.

Formula Editor

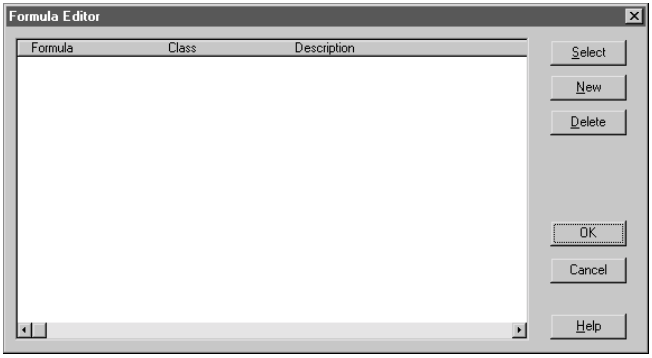
Over time, as Clarion developers increase their skills writing code, the formula editor tends to fall out of use. This is a tool that writes clean code for

you, and a great way to learn how to write Clarion code. The skilled Clarion developers know they can write code faster than the formula editor. Under these conditions, it falls out of use.

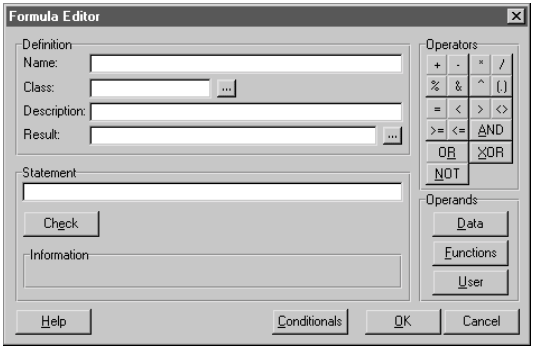
By the time ABC was originally released, many Clarion developers have long stopped using the formula editor, not realizing that it solves one of the biggest hurdles in ABC: “Where does the code go?” Lets find out now.

1. RIGHT-CLICK on the *Labels* procedure and **select Formula** from the popup menu.

   This opens a dialog showing all formulae written. Since we have none, it displays as a blank list.

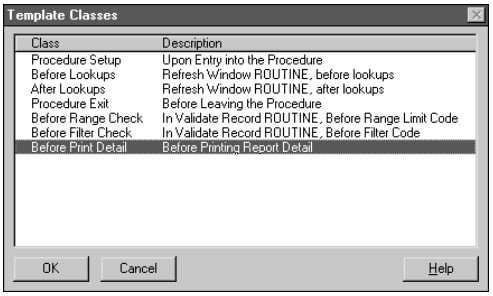


2. **Press** the **New** button and this dialog displays:



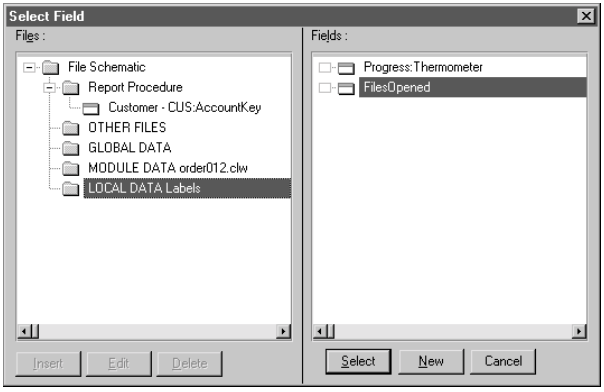
3. **Enter** *Conditional Company Name* in the **Name** entry.
4. **Press** the ellipsis(...) button to the right of the **Class** prompt.

   This displays the classifications of where the code generates.



5. **Highlight** *Before print detail* and **press** the **OK** button.
6. **Enter** *Company or Contact Name* in the **Description** entry.
7. **Press** the ellipsis(...) button next to the **Result** entry field.

   This displays a field dialog so you can select a variable to store the results of this formula.



   The variable needed does not yet exist. We need a new local variable, so **highlight** the *LOCAL DATA Labels* band and **press** the **New** button. The New field dialog displays.

8. **Enter** *LOC:Name* for the name of the variable.

> **Note: The "LOC:" portion of the field name is legal, and used here simply as a naming style. It is shorthand for "local". It aligns with the style used by the ABC templates. "GLO:" for global variables and "MOD:" for modular variables.**

9. **Enter** an optional description (recommended). Enter *Conditional company or contact name.* This way, you document the purpose of this variable.

10. The data type of STRING is what we want, so no changes here. The size of the string is too small, so **enter** *30*.

> **Tip:** **Use the *Derived From* lookup to define this variable from any existing variable that is a STRING data type of size 30. Since this is all that is needed, all other settings can safely be ignored.**

11. Press the **OK** button.

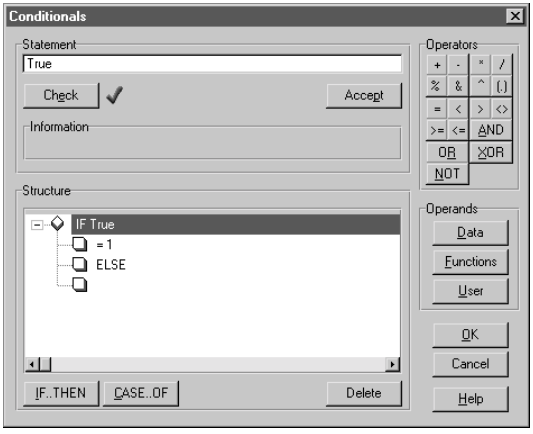
The local variable is placed in the **Results** entry for you.

12. Press the **Conditionals** button.

This opens the conditional dialog.

13. Enter *1* in the Statement entry and **press IF..THEN**.

You see this display:



This helps you write conditional formula code. You may either type in the expression to test for or use the buttons to place the field names and Clarion functions in the various entry fields. Note that all code you enter or lookup is the right-hand side of the equals sign. This means that you do not need to enter it, nor any IF or CASE commands. The formula editor writes these for you.

14. DOUBLE-CLICK the word *True* in Statement entry. This selects the entire word.

15. Press the **Data** button and then **select** *CUS:CompanyName* from the *Customer* file**.**

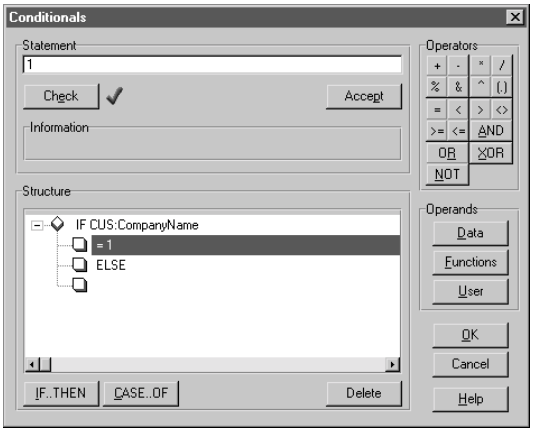
The field *CUS:CompanyName* is place in the Statement entry for you.

16. Press the **Check** button and then **press** the **Accept** button, in that order.

The condition is placed in the IF structure below. However, in order to see it, simply click anywhere in the structure diagram.

17. Click on the *1* on the line below the IF.

It is the first line with a square symbol on it. Two things happen. The first is that you see *CUS:CompanyName* appear on the IF line and the second is you see the *1* entered in the Statement entry. You should see this so far:



18. DOUBLE-CLICK on the *1* in the Statement entry to select it.

This line is the "true" expression. If CUS:CompanyName has a value in it (not blank), then what do we want to place in the LOC:Name variable? We want whatever non-blank data it contains.

19. Press the **Data** button and **select** *CUS:CompanyName* from the *Customer* file.

The field *CUS:CompanyName* is place in the Statement entry for you.

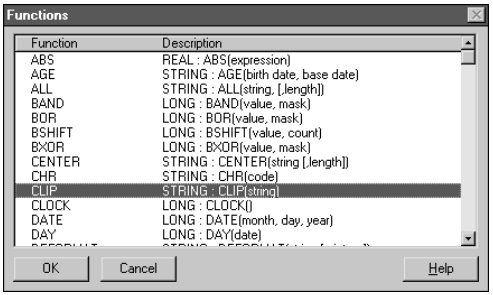
20. Press the **Check** button and then **press** the **Accept** button, in that order.

21. Highlight the line below the ELSE.

You see CUS:CompanyName in the second line. This line is our "false" condition. If CUS:CompanyName does not have data in it (it is blank), what should be passed to the LOC:Name variable? We want the contact's first and last name, plus we want it formatted correctly so it looks nice. In this case, the first thing we need is a Clarion function.

22. Press the **Functions** button.

The functions dialog appears. We need to clip off the trailing spaces on the variable we need, so **highlight** CLIP and **press** the **OK** button.



This places the CLIP function with a 0 (zero) highlighted in the parenthesis. This is what we want because the variable we select next replaces the highlighted zero.

23. **Press** the **Data** button and **select** *CUS:ContactFirstName* from the *Customer* file.

    The field *CUS:ContactFirstName* is placed inside the CLIP function for you, replacing the 0.

24. **Click** on the end of the statement so far.

    This places the cursor at the end of the statement. We need to concatenate a literal text. We need to do this as CLIP removes all trailing spaces. One must be put back, otherwise, the last name will touch the end of the first name.

25. **Press &** then type a single quote, then a space and then another single quote.

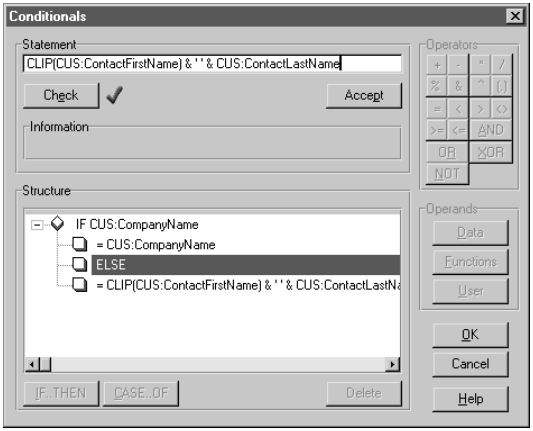
    This concatenates, or joins a literal space.

26. **Press &** again.

27. **Press** the **Data** button and **select** *CUS:ContactLastName* from the *Customer* file.

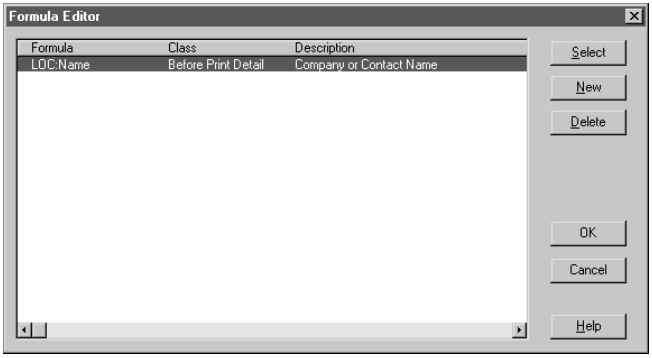
28. **Press** the **Check** button and then **press** the **Accept** button, in that order.

29. **Click** anywhere else in the structure and your conditional formula looks like this:



30. **Press** the **OK** button twice.

You are returned to the formula list dialog:



31. **Press** the **OK** button to return to the application tree and save your work so far.

We need to place this local variable on the report.

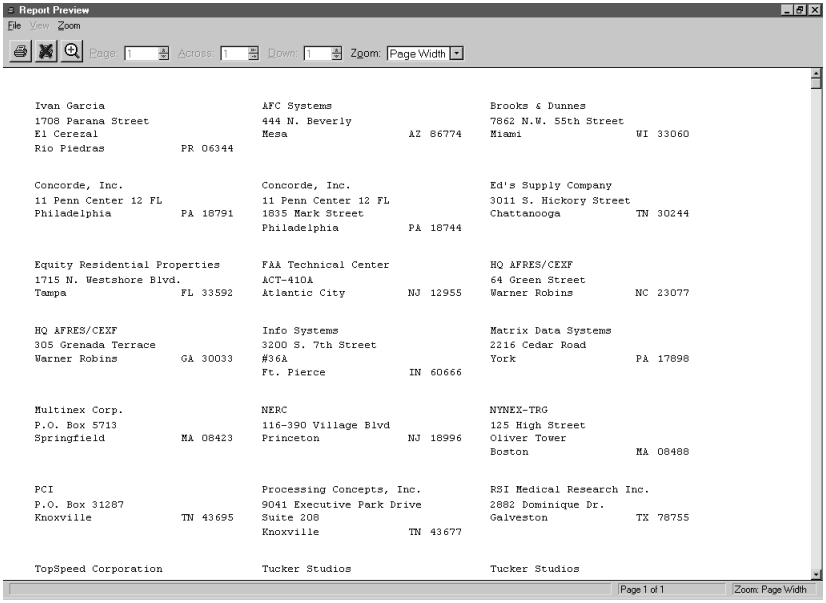
32. DOUBLE-CLICK on the *Labels* procedure. **Open** the report formatter by **pressing** the **Report** button. Highlight *CUS:CompanyName*.

This should be the first string on your report. Verify this by looking in the property toolbox.

33. Go to the string properties and **press** the ellipsis button (...) in the **Use** entry. **Highlight** *LOCAL DATA Labels* and **select** *LOC:Name*.

This changes the Use variable from *CUS:CompanyName* to *LOC:Name*. **Press** the **OK** button.

34. **Save** and **exit** the report. **Press** the **Run** button to test your changes so far.



The first label shows the contact name, while the rest show company names. It worked!

35. When you are done viewing the report, close the print preview window and exit the application.

Finding ABC embed points

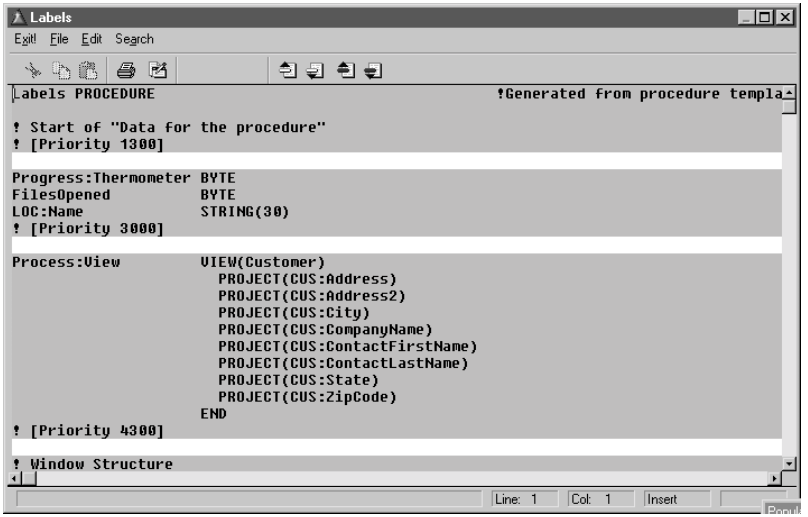
As discussed previously, we need to fix up the City, State and Zip Code line of our label. We could use the formula editor again, but what we really want is to locate the ABC embed where we would place the code. Also, we have no idea where it is.

The key question that must be asked is "The formula editor code worked, so I wonder where *that* code was placed?" It had to be placed in the correct embed as it did work. So how does one find it *quickly*?

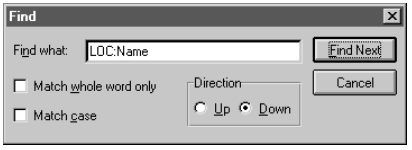
The answer is to use the *Embeditor*. This is the tool that shows you all the potential generated code if every embed is filled. Since the templates generated the correct code, we should find it without much trouble at all. Here are the steps needed:

1. **Highlight** the *Labels* procedure and RIGHT-CLICK on it.
2. **Select** *Source* from the popup menu.

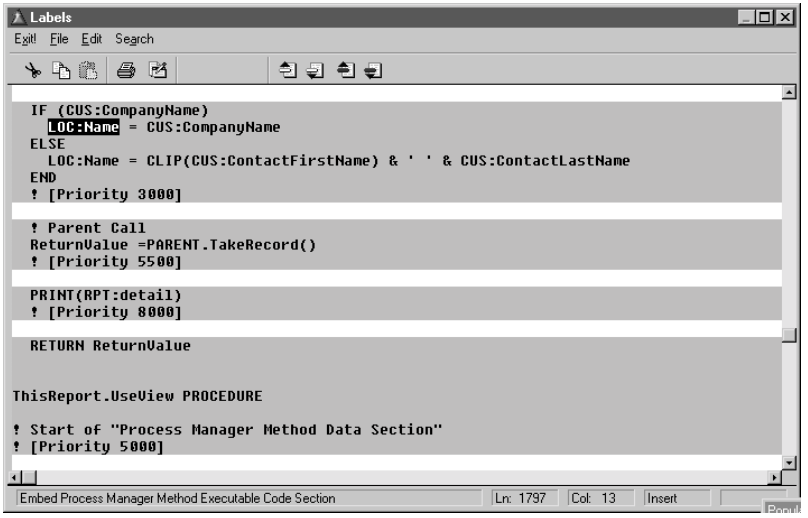
   This is what we see:



3. **Choose Search ➤ Find** from the editor menu and enter the following data:



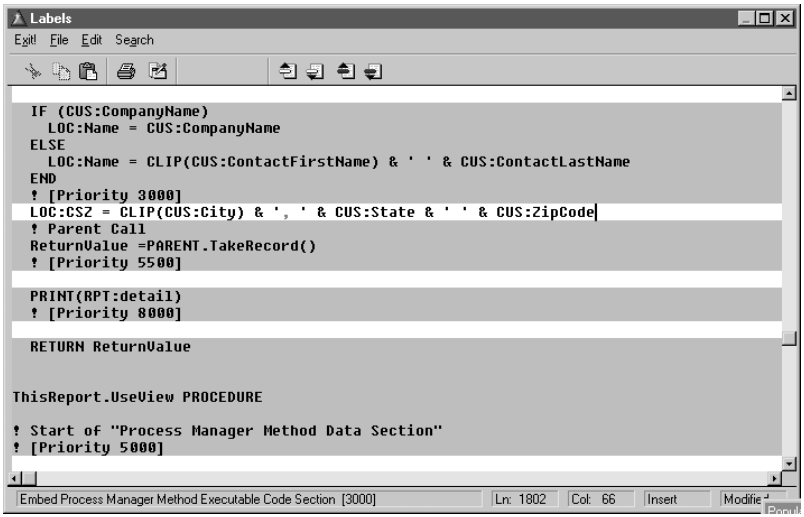
4. **Press** the **Find Next** button 3 times and then **press** the **Cancel** button. Our formula is visible.



We can now see what the embed point name is. You can do this in several ways. The first is to look for the line that says *PARENT.SomeMethod*. The "SomeMethod" is the name of the embed point. In this case, "*TakeRecord*".

We can scroll up a few lines until we find the name of the procedure. In this case, it is called *ThisReport.TakeRecord*.

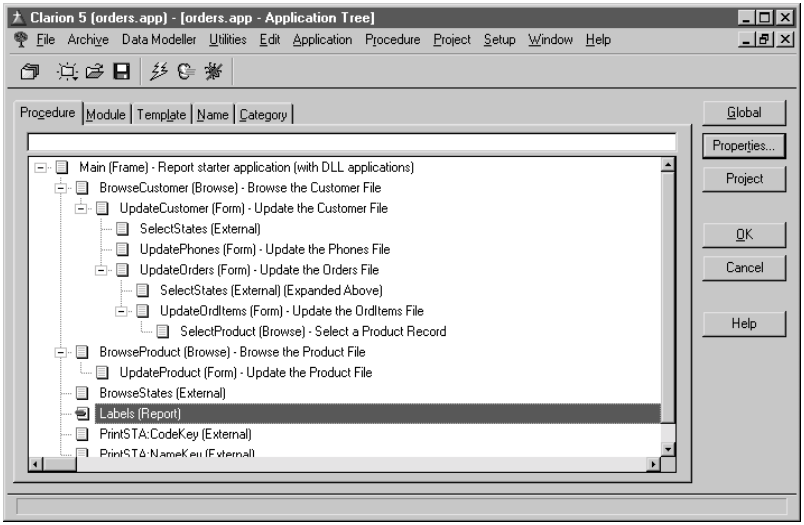
5. **Find** the first white line after the code generated by the formula editor. This is the embed where we place the next part of our code. Enter the code as shown below:



The variable LOC:CSZ is the name of the local variable to hold the results of this expression. Please note, that it is not defined yet and if you compile now, you will get an error.

6. **Press Exit!** from the editor menu exit to save the edits.

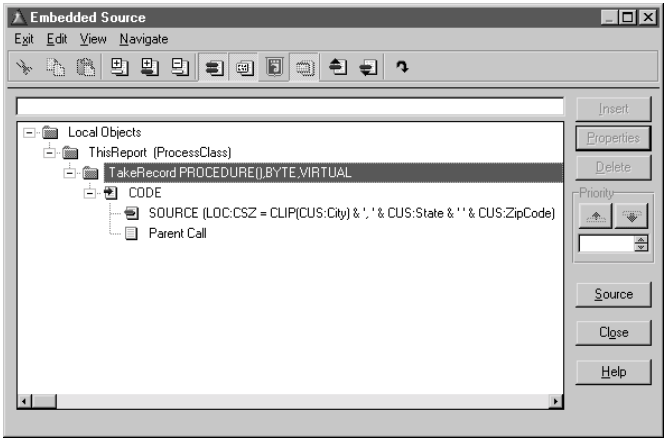
Notice that the *Labels* procedure's icon changed to indicate that there is a filled embed point.



7. RIGHT-CLICK on the procedure and **select *Embeds*** from the popup menu to see the exact embed point from the embed tree.

8. From the toolbar menu, **press** the **Show Filled Only** button.

The embed point is clearly visible.



It is the same name as we found in the embeditor. Notice that we never once opened the Help or the *Application Handbook.* The IDE simply told us exactly where to put the code!

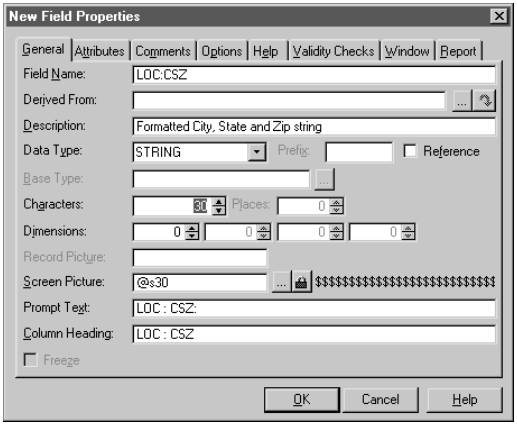
9. **Press** the **Close** button to return to the application tree**.**

10. RIGHT-CLICK and **select Data from the popup menu.**

Before we do anything else, the local variable used in the source embed must be declared.

11. **Press** the **Insert** button to define a new local variable.

    Enter the data as shown below.



12. **Press** the **OK** button to save the definition and then **press** the **Cancel** button. **Press** the **Close** button to close the dialog and return to the application tree.

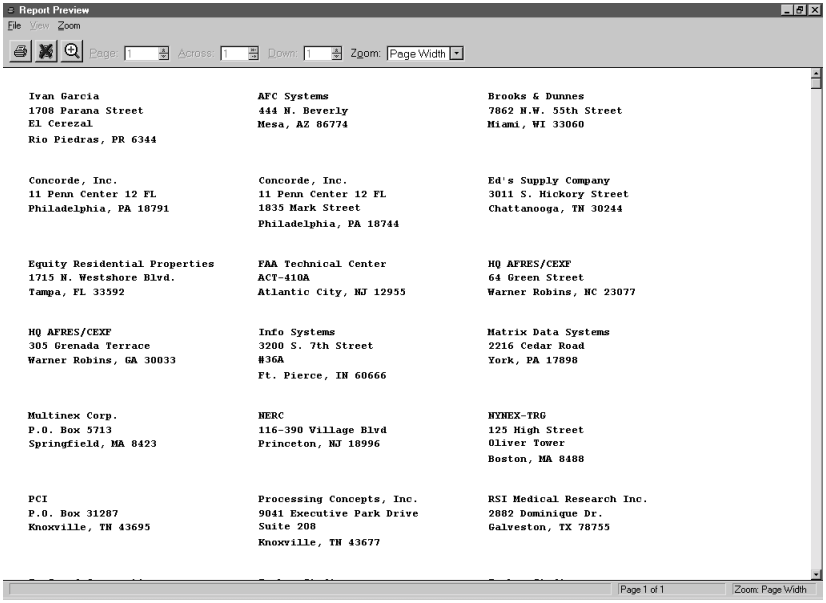
13. **Open** the Report Formatter.

    We need to use our local variable in our report. Since *CUS:ZipCode* and *CUS:State* are not used, delete them by selecting them and then **press** DELETE.

    Select *CUS:City* and **open** the properties dialog. Change its Use variable to *LOC:CSZ*. If needed, make any alignment adjustments and then save your work.

14. **Save** the application and then run it.

    Your report should look like this:



If you look closely at your report, some of the labels will not work. The zip codes that begin with a 0 (zero) now have 4 digits printing instead of the required 5 digits.

Since the zip code is stored as a numeric (DECIMAL), Clarion converts this number to a string (our local variable) according to the rules described in the *Language Reference Manual* on page 468. In essence, the value is left justified in the destination (our local variable string).

15. **Open** the embed tree and find our source embed. Change the code to either of the following:

```
LOC:CSZ = CLIP(CUS:City) & ', ' & CUS:State & ' ' & |
  FORMAT(CUS:ZipCode,@n05)
```

or

```
LOC:CSZ = CLIP(CUS:City) & ', ' & CUS:State & ' ' & |
  FORMAT(CUS:ZipCode,@p#####p)
```

If you notice, the FORMAT statement forces a picture type and the leading zero now prints. The first code example uses a numeric with leading zeroes (@n05). The second used a pattern picture that says zeroes can appear anywhere in the string (the # symbol).

Summary

In this lab you:

- Covered how to properly set up any report.
- How to use the Page Layout view.

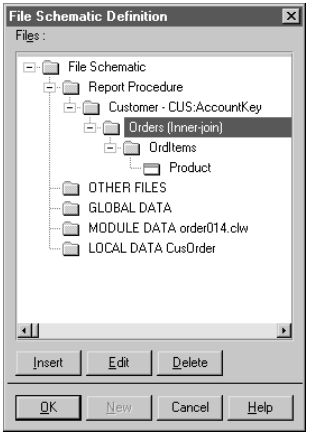
- How to make a label report.
- Used the formula editor to discover an ABC embed point.
- Using an ABC embed to write our code.

Multiple File and Group Breaks

This lab will guide you in making an invoice type report using multiple files and group breaks. It also shows you how to do totalling in a report.

This lab starts from the Orders application you completed in the last lab or you may use the Starter order application located in the *X:\Clarion5\CBT\Essentials\Lab03\Start* folder.

1. **Add** a new procedure. Press INSERT and type *CusOrder* for the procedure name.
2. **Select** *Report* as the procedure type. Ensure that the Procedure wizard box is *not* checked.
3. You will now see an empty procedure properties window. Next to the Files button, you will see a red question mark. This means that a file is required. **Add** files to the <ToDo> band so that it looks like this (the Inner join and key name is set from the Edit button):



This sets up the View structure so that working with the report is much easier. This gives you an implied filter and makes the view efficient.

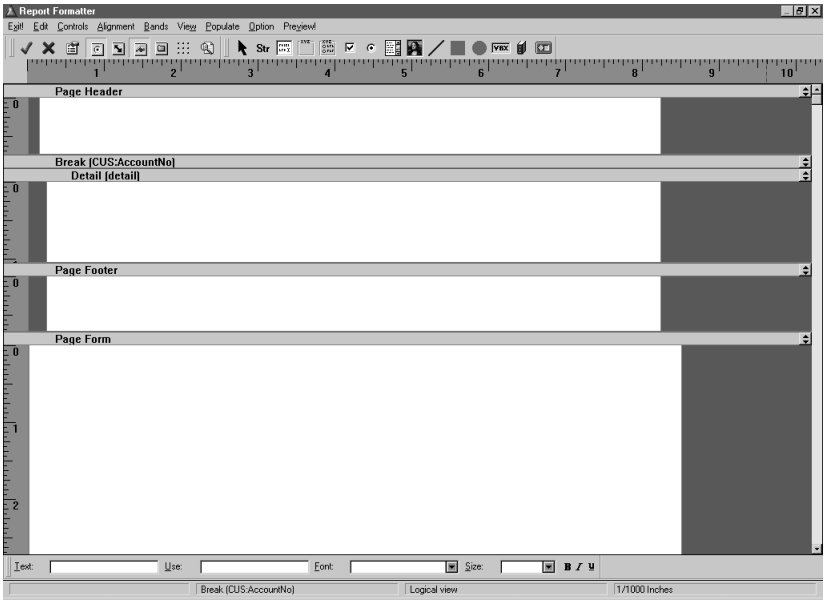
4. **Open** the Report structure and you will see an empty report.

   To make the report look better, **select** the Edit menu and **select** Report Properties. Choose a *Courier New* **Font**, *Regular* **Font Style**, *and a* **Size** of *8*. You may optionally fill in the **Job Name** (recommended). Set the **Paper size** to *Letter*. Close the dialog box when you are done.

5. **Choose View ➤ Page Layout View** from the Report Formatter menu and adjust the size and position of the report bands. Refer to the previous *Labels* report, if needed. Switch back to band view when you are done.

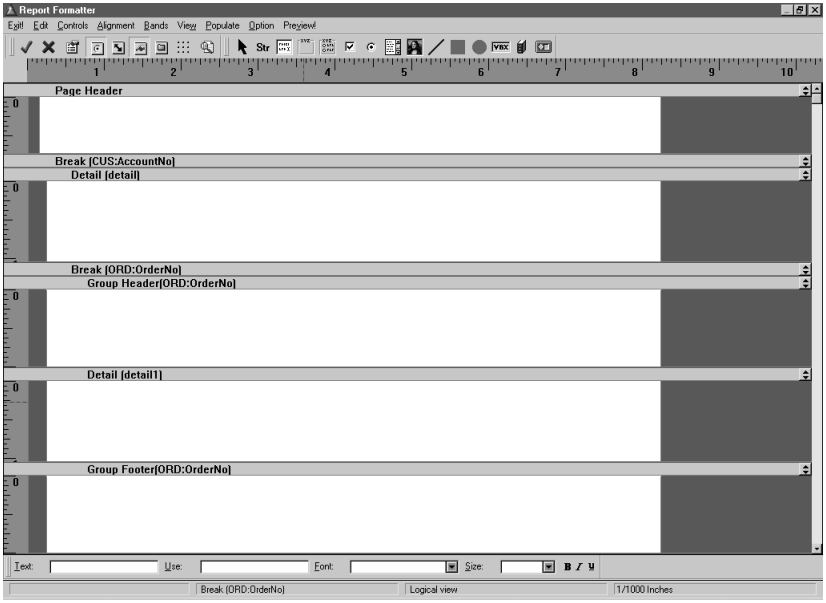
6. We need to add some bands to make this report work. **Choose Band ➤ Surrounding Break** from the Report Formatter Menu. Click anywhere on the detail band to add it.

   You are prompted for a break variable. **Select** *CUS:AccountNo* as this group should print when the customer account number changes. Give the break a friendlier label by entering *CustBreak*. This way, if you ever need to refer to it in code, it is easier to work with than "break1". So far, your report layout should look similar to this:



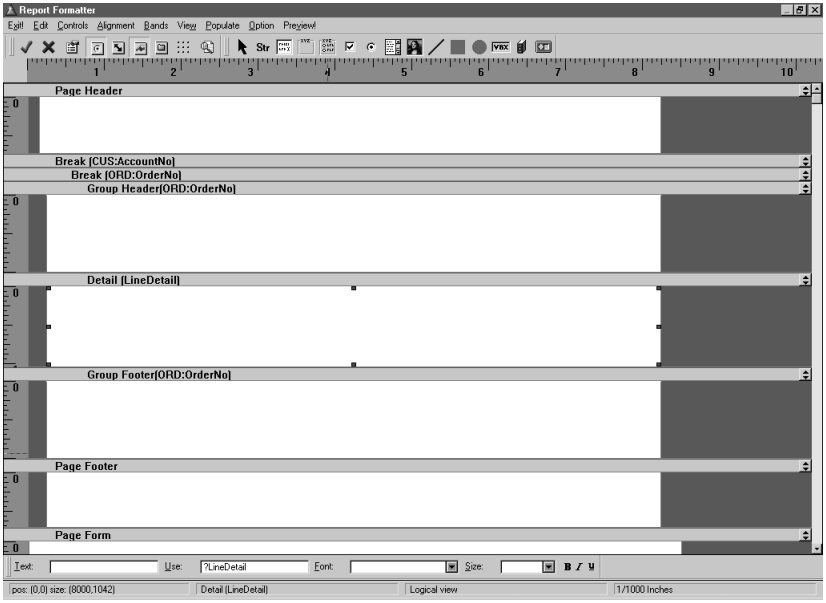
7. We need to add a group break for the Order file. **Choose Band ➤ Break Group** from the Report Formatter Menu.

   Click on the detail band. The variable that we need is ORD:OrderNo. Lets use the label, "OrderBreak". This adds a group header, another detail and a group footer. So far, your report bands appear similar to this:



8. **Delete** the *original* detail band by clicking on it and press DELETE. It is no longer needed.

The only detail band remaining, inside of the ORD:OrderNo break, needs a friendlier label. RIGHT-CLICK on the band and choose **Properties** from the popup menu. **Change** the **Label** to *ItemDetails* and name the **Use** variable, *?ItemDetails*. If you ever use a filter on this band, you must have a field equate label. The layout appears similar to this so far:



The previous steps showed two ways to add a group break to a report. One simply added a break, the other added a break, group header and footer structures, and a detail band. We need to add group structures to the original break.

9. **Click** on the *Break (CUS:AccountNo)* band.

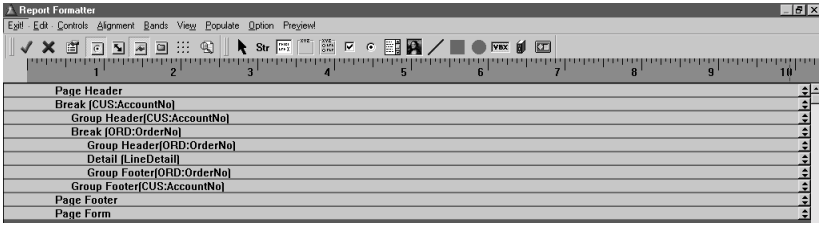
10. **Choose Bands ➤ Group Header** from the Report Formatter Menu. The mouse cursor changes to a cross.

11. **Select** the *Break (CUS:AccountNo)* band to drop the group header band into this group break.

12. **Choose Bands ➤ Group Footer** from the Report Formatter Menu. The mouse cursor changes to a cross.

13. **Select** the *Break (CUS:AccountNo)* band to drop the group footer band into this group. You may need to scroll down to see it.

14. **Choose View ➤ Expand Bands** from the Report Formatter Menu. This is a toggle menu item. It collapses all bands. Your report structure should look like this:



Tip: **Sometimes the bands do not nest within each other as you expect. If you find this is the case, the remedy is to** select **the band that is out of place, then select Edit/Set Control Order. The band that you have selected is the one selected in the list. Simply move it up or down by clicking on the up/down buttons until you have it placed where it belongs. This action can be done at anytime.**

15. **Press** the button on the far right of the Page Header band.

    The button looks like two up/down arrows. This expands just the Page Header band. You could also expand all the bands.

    Add two Control Templates, *Report Date Stamp* and *Report Time Stamp* to the header. The **Actions** tab of each can be used to change the picture tokens. Give yourself some room if either control template is used on the right-hand side as it places two controls on your report and the second can be placed beyond the right hand edge of the report.

Important:
Don't populate either control template withits controls beyond the borders of the report. Place one control template to the far left of the band, then place the other just to the right of it. You can then drag the controls to their final position.

16. **Add** a report title string by dropping a string control near the top of the header band. Stretch the width of the control so it is as wide as the header band.

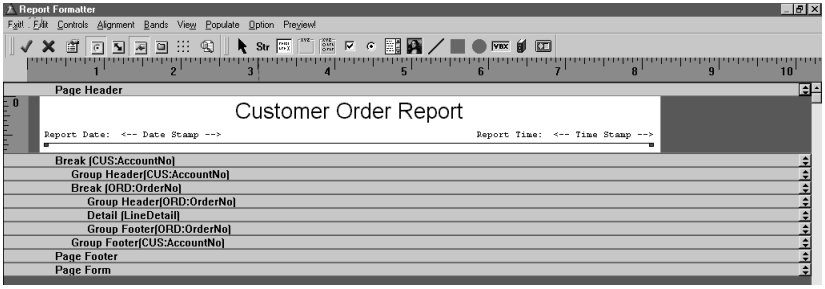
17. **Press** the **Properties** button on the command toolbar.

    Change the string text to *Customer Order Report*. Change the Use Variable to *?ReportTitle*. This makes it easy to refer to it in code, if you ever need to address it. Change the justification to *Centered*. This is a nice thing to do if you need to ever change the title with PROP:Text, it will always be centered on the band.

18. On the **Position** tab, **change** the **Height** to *Default*. Change the **Font Size** to something bigger (your discretion) and different font family if you wish. The default height ensures that no matter what the size or text or font family you choose later, it will always look right.

19. **Add** a horizontal line and stretch it across the band near the bottom if you wish.

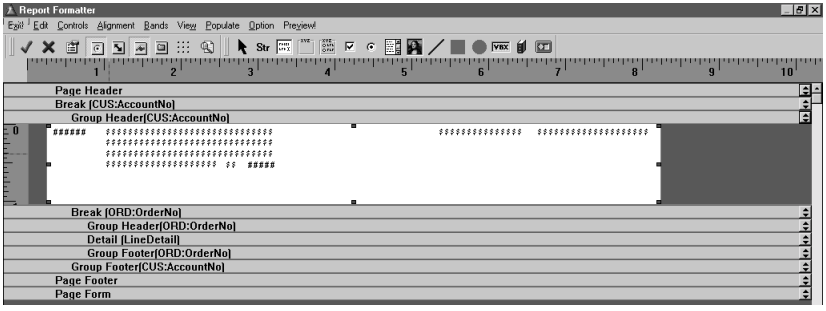
    Using alignments, clean up the appearance to your tastes. The report should look similar to this:



20. Collapse the Page Header band and open the *CUS:AccountNo* group header.

Add the customer information here (from the Customer file). The best way is to add multiple fields. You can do this using the **Dictionary fields** button from the control toolbar or by choosing **Populate ➤ Mutliple Fields** from the Report Formatter Menu. As you add each one, the dialog reappears allowing you to choose another field.

Press the **Cancel** button when you are done. Align them so they look neat. Collapse this band when you are done. Your report should have an appearance simliar to this:

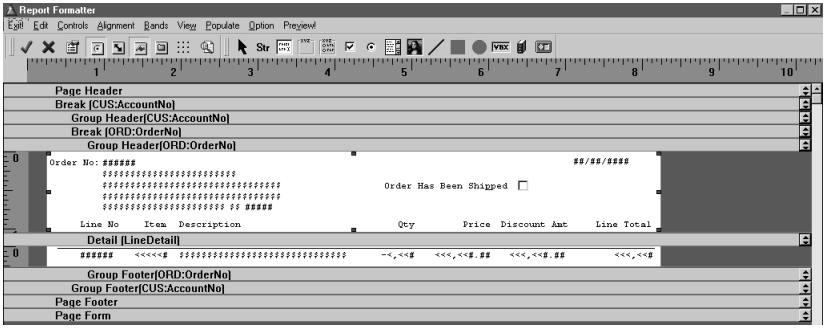


21. **Open** the *ORD:OrderNo* group header band and add multiple fields from the Order file. **Add** some strings where needed to identify these fields. Align them so they look neat. **Collapse** the band when your done.

22. **Open** the detail band. **Add** the following fields in this order, all on one line; *ITE:ItemNumber, PRO:Description, ITE:Quantity, ITE:Price, ITE:DiscountAmount, ITE:Total.*

Align them so that they look neat. As an added step, **open** the *ORD:OrderNo* band and click on it. If needed, drag the bottom of the band to give you more room. Add some string controls for the above fields. **Arrange** them so they appear above the details fields.

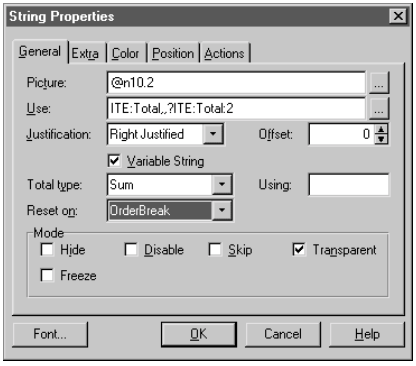
Add a horizontal line above these fields if you wish. **Resize** the group header and detail height as needed.

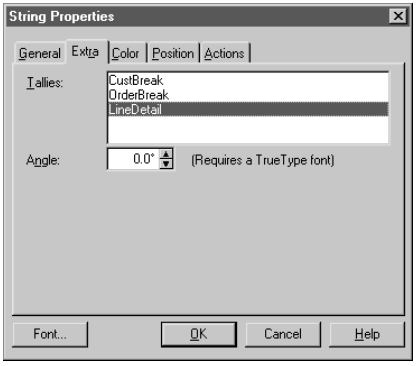


23. **Open** the *ORD:OrderNo* group footer.

You need to add a total field. The easy way to do this is to copy the field you want to total. Click on the *ITE:Total* field from the detail band. Without clicking on a mouse button, move your mouse cursor to where you want the total field to be placed. **Press** F2.

The field appears where your mouse cursor is. **Change** the properties on the **General** and **Extra** tabs so it looks like the following:





This sets up totaling for this band and will total only when the order number changes. **Add** a string control next to this total. **Change** the text to read “Order total:” **Position** both controls so that they look good under the line total field.

24. **Open** the *CUS:AccountNo* group footer band and add a string control.

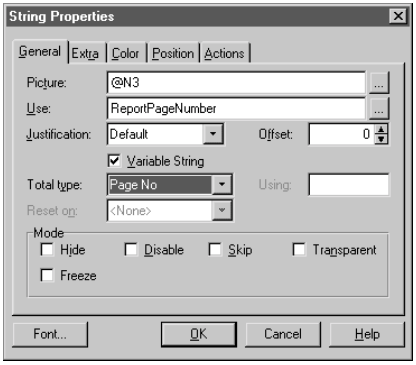
Change the text to read “Customer total:”.

Copy another total field like we did above (using the F2 method). The **General** tab will be the same as the above figure, except reset on *Customer Break*. The **Extra** tab is the same.

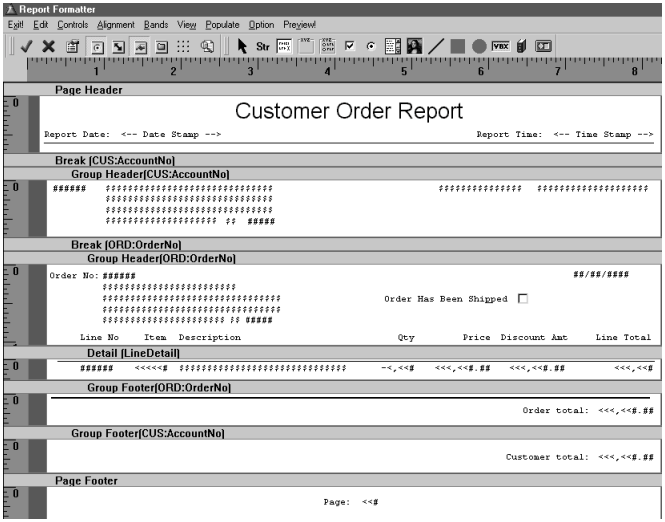
If you want to do a grand total for the entire report, **add** another total field like the above, except do not enter anything in the “Reset on:” entry.

25. Now lets set up the page footer. **Add** a string control and change the text to read “Page No:”

Drop the control template, “Report Page Number” next to your literal. **Change** the properties to look like the following:



26. When you are done, your report should look similar to this:



Add the report procedure to your main menu if you have not already done so. **Save** your work and **run** the report.

You will find that the report works, in that only customers with orders print. This is the Inner Join filtering out customers with no orders (and it is faster than a filter).

This report could use some improvement in how it flows from page to page. All the orders print one after another.

Controlling Page Overflow

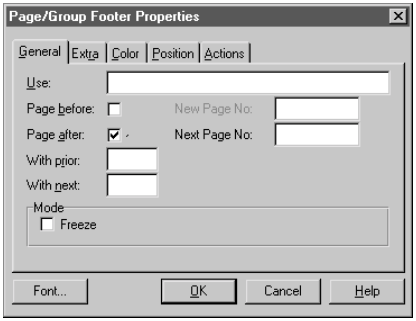
In this last section of the report lab, you need to make some final adjustments to the report. In order to make the report behave like an invoice, you need to instruct the report engine to start a new page in areas that make sense.

In this design, we want a new page when the order changes and when the customer changes. This way, each order prints on its own group of pages, with a total for that order, and we get a summary total page for each customer when all their orders print.

The starting point is the *CusOrder* procedure, with the report formatter open.

1. RIGHT-CLICK on the *ORD:OrderNo* group footer band and **select Properties** from the popup menu.

Change the properties to look like this:



This change forces a new page before the next order is printed. If you need to reset the page number, enter the next page number in the **Next Page No:** prompt. This is useful if you send invoices to customers and you wish to have each invoice that prints on multiple pages print with their own page numbers. Leave this blank for this report.

2. Make the same changes to the *Cus:AccountNo* group footer band.

   This causes the customer totals to print on their own page.

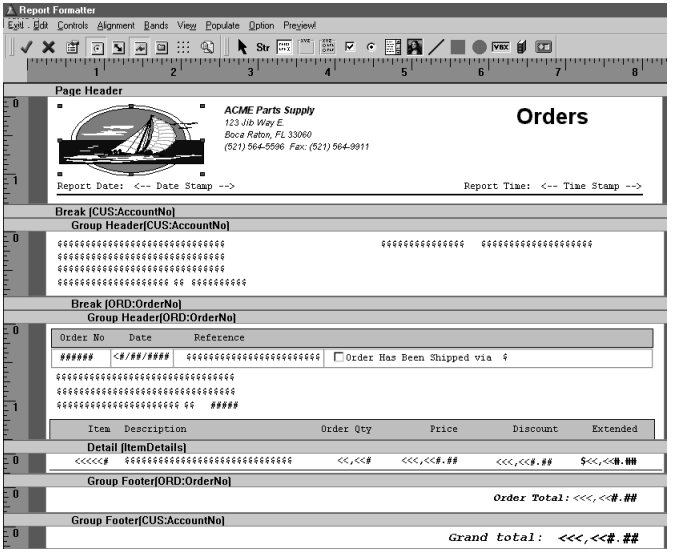
3. **Save** your work and test the report.

Wrap up

You made two reports in this lab, one a simple label report that needed some embeded source (and a new ABC embed point was discovered). The other report used group breaks and multiple files and total fields.

In the solutions application, some embellishments were added. It uses an image on the report and has boxes around the detail line controls. It also used font changes and some shading. It works exactly like what you did with this lab, the differences are cosmetic. Feel free to explore and inspect how it works.

Here is the formatter view of it:



Summary

You have learned:

- How to make label reports.
- The useful menu functions to control how a report is properly set up.
- How to layout your printable areas using Page Layout View.
- How to set a default font for the entire report.
- How to adjust control order (including groups).
- How to manage page breaks.
- How to total on a field.

LAB 4

Finding ABC methods and Embed Points

Introduction

What We Are Going to Accomplish

In Lab 4, the goal is to discover how to find and use ABC methods and ABC embed points. This lab assumes that you have no knowledge about how to proceed.

Again, we are continuing the Orders Application. We'll use the Orders application we used in Lab 3 as a starting point for this Lab.

What we will accomplish during Lab 4:

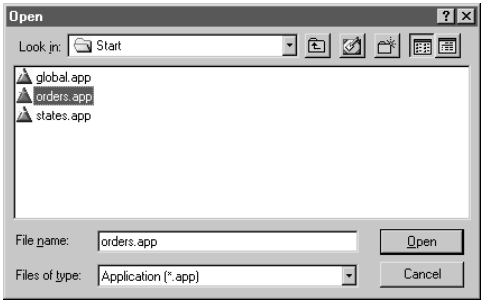
- Pursuant to a customer request, we will change the BrowseCustomer procedure to conditionally color a column in the Browse Box.
- Using the IDE, we will discover not only where to put an embed (if needed), but we want ABC to do the task for us.
- If embed code is needed, then we want to write as little as possible.

Exercise Starting Point

We are now going to launch the Clarion environment and add a feature to our existing Orders application.

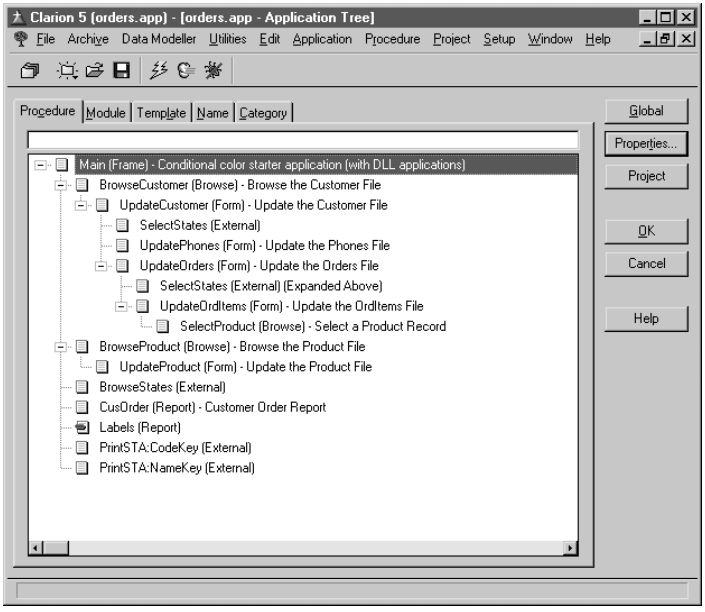
You should have just opened the Clarion development environment. The Pick dialog should be open.

1. **Press** the **Open** button.
2. **Navigate** to the *X:\Clarion5\CBT\Essentials\Lab04\Start* folder.
3. **Select** *Application(*.app)* from the **Save as Type** drop down list box.



4. **Highlight** *Orders.app*, and **press** the **Open** button.

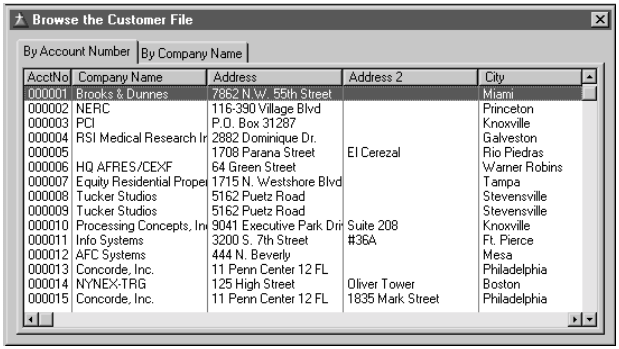
   The Application Tree displays.



This application has the two DLLs linked in (Global and States) from the DLL lab. The DLL libaries and links are provided. However, you can recompile the entire project if you wish.

The Customer Request

Here is our running BrowseCustomer procedure:



What the client wants is to see some indication on this window, which customer has (or does not have) an order on file. This way, his sales personnel can glance at this window and call the customer to get a new order. The Client has stated that we cannot add any new controls on this window, we must use what is there. Changing the color of the account number is acceptable, but only if there is an order on file for that customer, otherwise we leave the colors alone. This is what the client wants.

Also, to add another “handicap”, we want to have ABC do all the hard work for us. As a developer, we want to avoid using embeds if possible, or write the minimum of code needed.

This means, in very simple terms, we have two major goals:

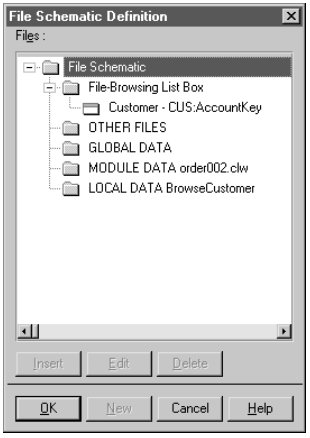
1) Access the Orders file.

2) Conditionally change the color of one of the columns.

Accessing the Orders file

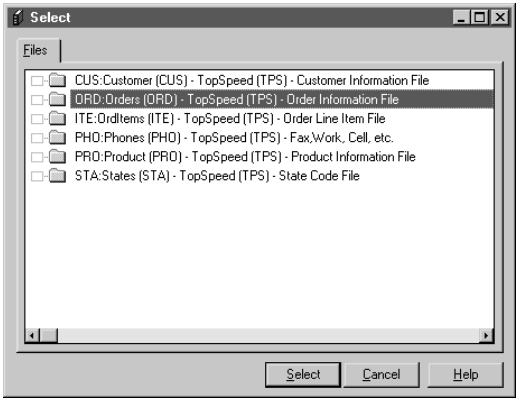
1. **Highlight** the *BrowseCustomer(Browse)* procedure in the Application Tree, and **select Files** from the RIGHT-CLICK popup menu.

   The File Dialog displays showing all the files the procedure knows about.



2. **Highlight** *Other Files* and **press** the **Insert** button.

   The Select File dialog displays.



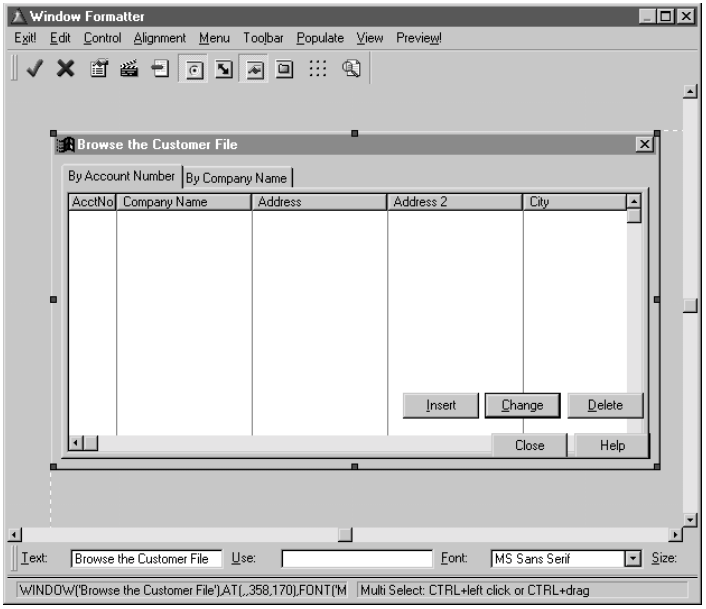
3. **Choose** *Orders* and **press** the **Select** button.

   The Orders file is now known to this procedure, and more importantly, the ABC templates know about it. More on this later. **Press** the **OK** button to close the dialog. The first major step is now done (access the Orders file).

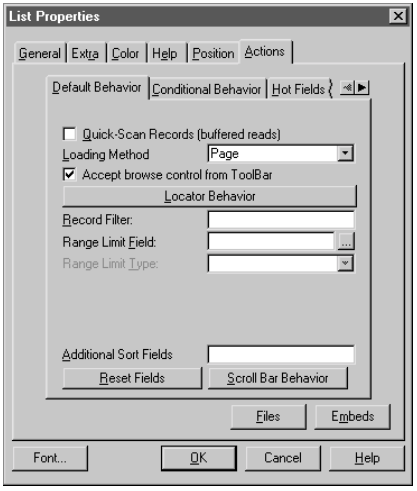
Enable colors in the Browse Box

We need to set our color conditions. This is done via the Window formatter.

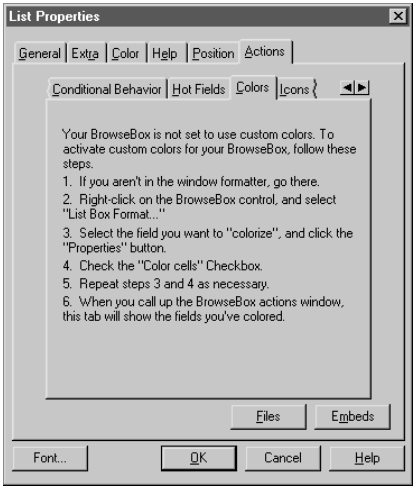
1. With the *BrowseCustomer* procedure highlighted, RIGHT-CLICK and **select** *Window* from the popup menu.



2. RIGHT-CLICK anywhere in the list box and **select Actions** from the popup menu.

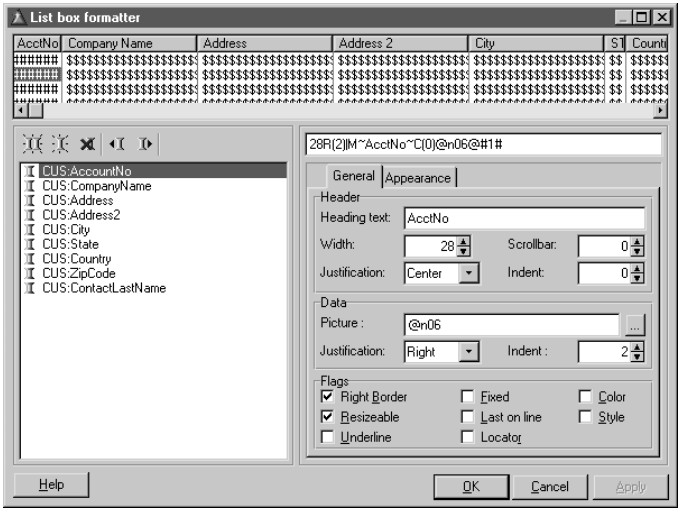


3. **Press** the right scroll button until you see the **Color** tab.



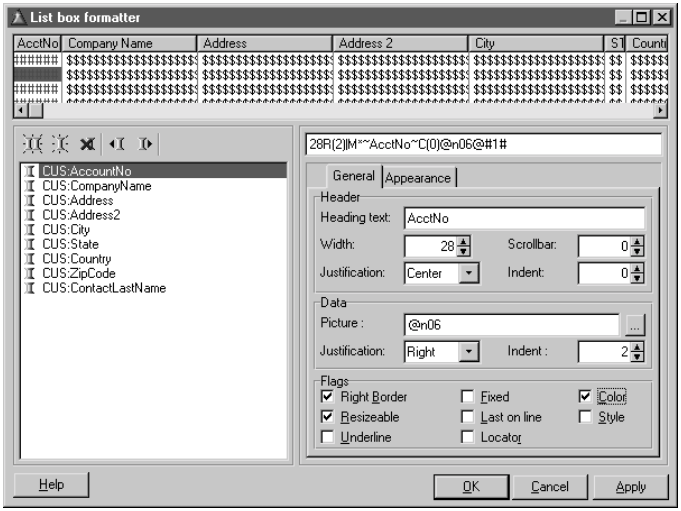
We cannot do anything yet as there are no columns in the list box that are color enabled. Fortunately, there is text on this tab that describe what you need to do. **Press** the **OK** button to close the dialog. We will come back to it.

4. RIGHT-CLICK anywhere in the list box and **select List box format...** from the popup menu.



5. **Check** the **Color** box.

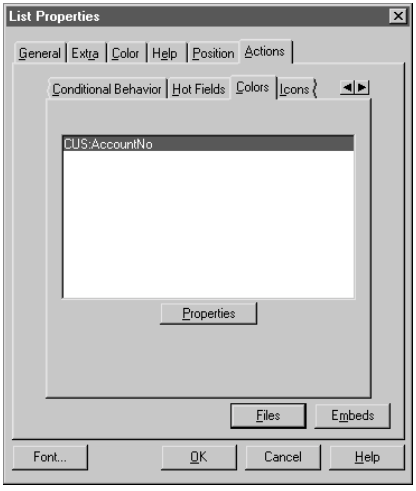
The CUS:AccountNo column turns red indicating that color is now active.



6. **Press** the **OK** button to exit the List box formatter.

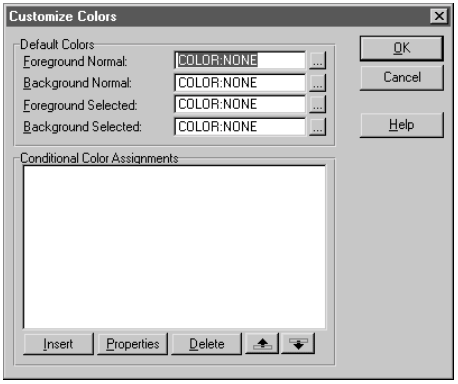
7. RIGHT-CLICK anywhere in the list box and **select Actions** from the popup menu.

8. **Press** the right scroll button until you see the **Color** tab.



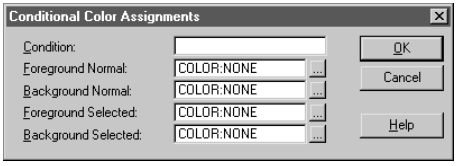
The Colors tab now changed from a display of text telling you how to turn on the feature, to a list showing all columns that have color turned on.

9. **Press** the **Properties** button.



You can see there is a conditional color group.

10. **Press** the **Insert** button. This dialog displays.

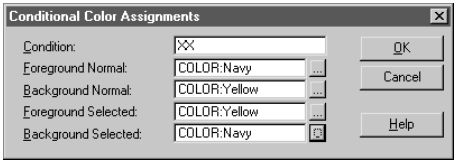


Now lets think: What is the condition? We cannot code something to try a get a record as this templates generates an IF code structure. We need something that the compiler will accept.

Think about this: If we *did* have a valid expression suitable for an IF, where would the ABC templates write the code?

Lets play a little "what if". **Enter** *XX* or something else as meaningless. This entry should be easy to find on a search. Since we do not plan to compile this, it won't matter anyway.

11. **Change** each color to something else.



12. **Save** your work so far by **pressing** the **OK** button three times. You are back in the window formatter.
13. **Press** the **Save Changes** button. You are back in the application tree.

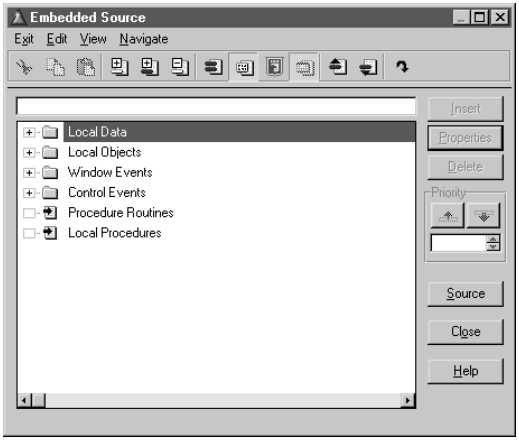
Where did ABC put that code?

In this section, we'll find exactly where the code was placed. Logic tells us that it must already be in some ABC embed point somewhere.

You should still have the Application Tree displayed.

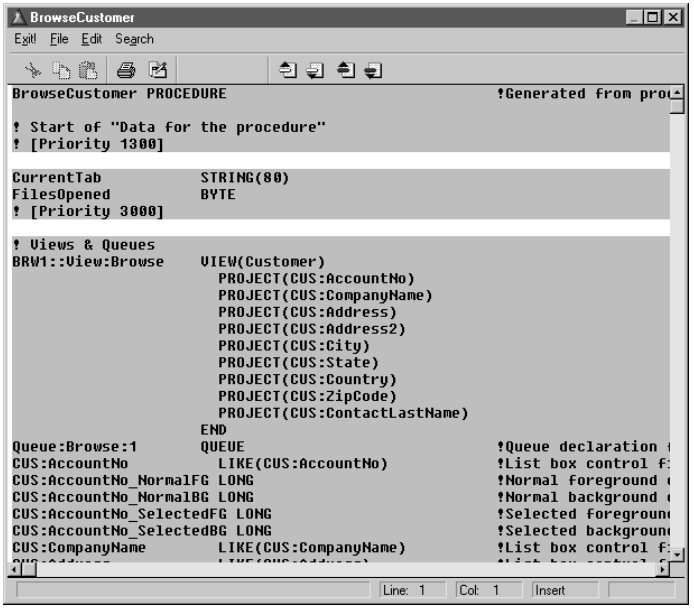
1. **Choose Project ➤ Generate** from the Clarion menu to generate code.
2. **Highlight** the *BrowseCustomers(Browse)* in the Application Tree, RIGHT-CLICK and **choose Embeds** from the popup menu.

   The Embed tree displays. **Press** the **Contract all** button on the toolbar.



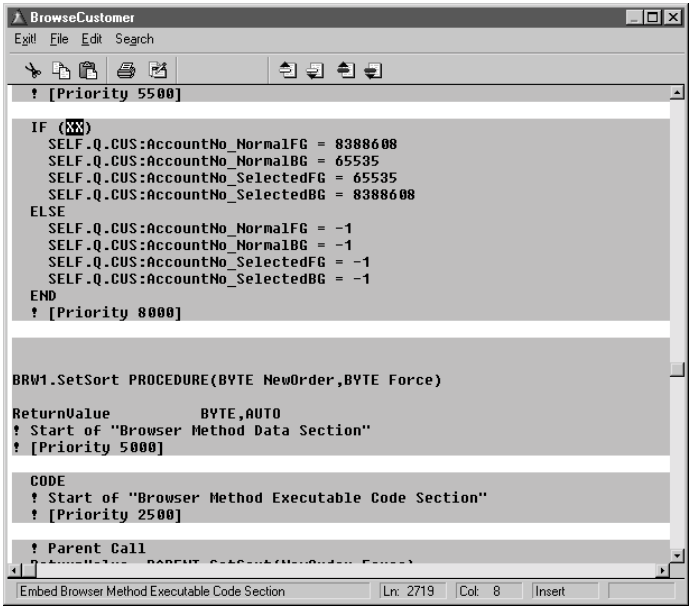
3. **Press** the **Source** button to start the embed editor.

Tip: **Launching the embed editor via the embed tree, allows you to switch back and forth between these two embed tools easily.**



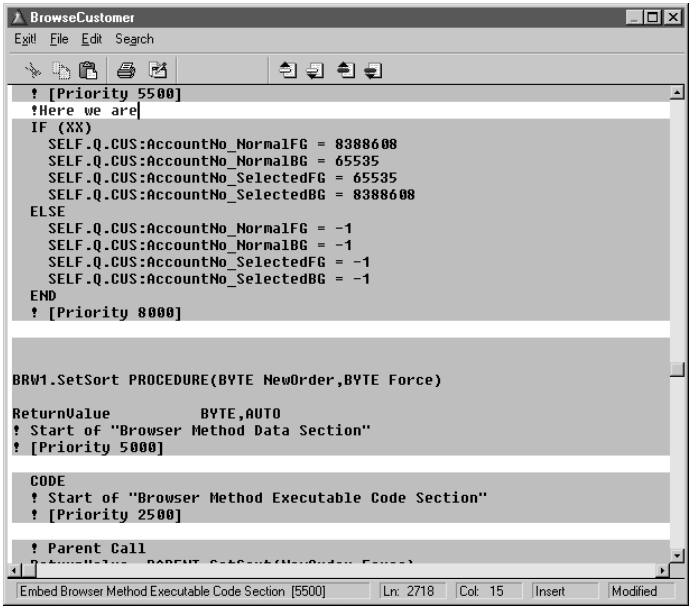
4. **Choose Search ➤ Find** from the editor menu.

Enter *XX* (or whatever value you entered in the condition) in the **Find** entry. **Press** the **Find next** button until you find the value we are searching for. **Press** the **Cancel** button to close the Find dialog.



We can see the code generated by the conditional color dialog. We can assume that the template generated this code in the correct location.

5. **Place** your cursor on the white line just above the IF (XX) line. **Enter** *!Here we are* as a comment.

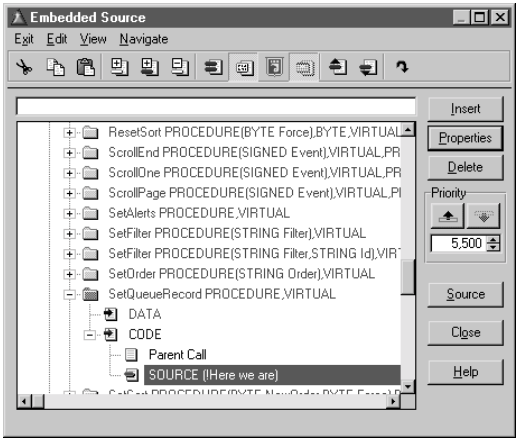


6. **Press Exit!** from the editor menu and save your changes.

The embed tree shows one icon that has changed color, so there is a filled embed point somewhere.

7. **Press** the **Next filled embed** button on the toolbar (second from right).

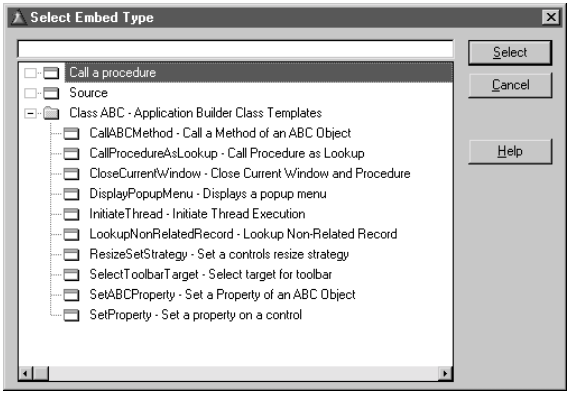
The ABC embed point is revealed.



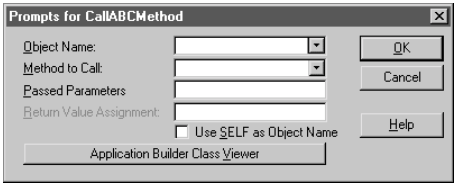
The embed is called *SetQueueRecord*. This makes sense as a Queue controls a list box's appearance (colors, tree list, icons, etc). Both legacy and ABC templates generate Queue structures based on how you designed your list box columns. If you checked the Color, Icon or Tree boxes in the Listbox formatter, then extra Queue fields are generated for each column to control that column's appearance.

So we have found the ABC embed point easily. But the problem still remains, how do we get a matching record from the Orders file? We could, of course, hand code this to get what we want, but remember, we want ABC to do all of the work. How can one find an ABC method to do this?

8. Press the **Insert** button and the *Select Embed Type* dialog displays.



9. Highlight *CallABCMethod* and **press** the **Select** button.



This dialog displays only ABC objects that are applicable to the current procedure. Drop down the Object Name. Based on name only, what is it we are trying to do? Access the Orders file, right? The list displays an object name almost exactly like the task we wish to perform, *Access:Orders.* **Select** it.

10. Drop the **Method to Call** drop list.

This list contains all the methods for *this* object only. This means that you do not have to go through the thousands of methods that ABC provides. Based on the name of these methods, write down which method(s) sounds like it will do the task. Do not worry about picking the wrong one, just make a list. You may have one, two, three or more methods that sound like what we want to do.

Most Clarion developers pick these methods for their list:

1) Fetch(KEY K)

2) GetField(KEY K)

3) TryFetch(KEY K)

11. **Press** F1 to open the Help window.

12. **Press** the **Search** button from the toolbar and **enter** *Fetch*. **Choose** *Fetch (get a specific record by key value).*

    The key piece of information from the Help says, "*The Fetch method gets a specific record by its key value and handles any errors. You must prime the key before calling Fetch.*" and "*Fetch tries to get the specified record. If it succeeds, it returns Level:Benign. If it fails, it returns Level:Notify*". This sounds like what we want to do. Put a check next to *Fetch* on your list. Lets see if the other methods are also applicable.

13. **Press** the **Search** button from the toolbar and **enter** *GetField*. **Choose** *GetField (return a reference to a key component).*

    The key piece of information from the Help says, "*The GetField method returns a reference to the specified key component.*." This does not sound like what we want to do. Cross *GetField* on your list. Now let examine our last item on the list.

14. **Press Search** from the toolbar and **enter** *TryFetch*. **Choose** *TryFetch (try to get a specific record by key value).*

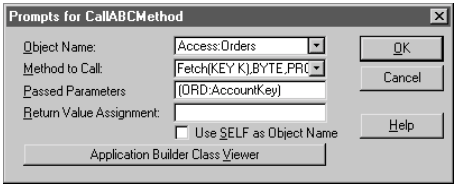
    The key piece of information from the Help says, "*The TryFetch method gets a specific record by its key value and handles any errors. You must prime the key before calling TryFetch.*" and "*The Fetch method provides a slightly different (automatic) alternative for fetching specific records.*" Also notice that *TryFetch* returns the same error values. This sounds like what we want to do. Put a check next to *TryFetch* on your list.

    Notice that the Help refers to *LEVEL:Benign* and other error levels. *LEVEL:Benign* is zero, all others are not. So *Fetch* (or *TryFetch*) returns *LEVEL:Benign* if it succeeds. This means that there were no errors getting the record asked for. If it cannot get a record (most likely because it does not exist), then a non-zero value is returned.

    This is the action we need, as it answers the question, "Does this customer have a related order on file?" The answer can only be "Yes" or "No" as that is all we are interested in.

15. **Select** *Fetch* from the drop-down list.

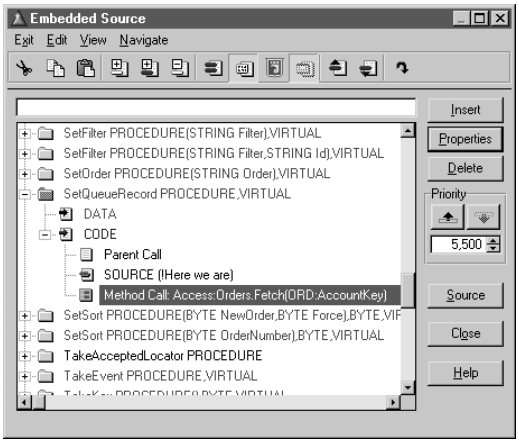
16. The template needs a name of the key that is used to relate the two files together. **Enter** *(ORD:AccountKey)*. You must enter the parenthesis too.



Tip: **Start the dictionary viewer before you open any dialog. The dialogs are application modal and you will not be able to open it until you close the dialog. It is started by choosing Application ➤ Dictionary Viewer from the menu.**

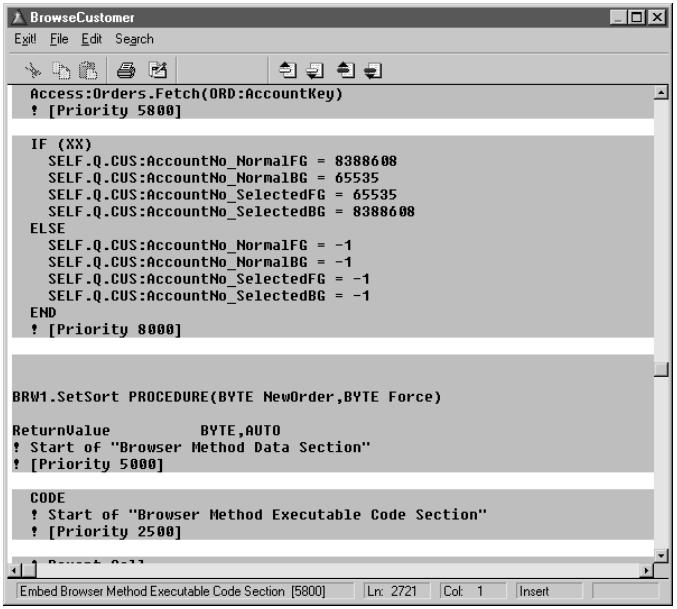
To see the Dictionary Viewer while working with the embed tools, be sure to click the "Stay on top" button (looks like a pushpin).

17. **Press** the **OK** button to close the code template.



18. **Press** the **Source** button to start the embed editor.

    You are now positioned near the code the code template generated.



If you look at the first line, you see the code the template generates. The next gray area shows the condition for the colors. Remember the two key things the Help told us about Fetch; 1) You must prime the key before calling this method (or it will always fail) and 2) returns a zero (LEVEL:Benign) if it finds a record or something else if not.

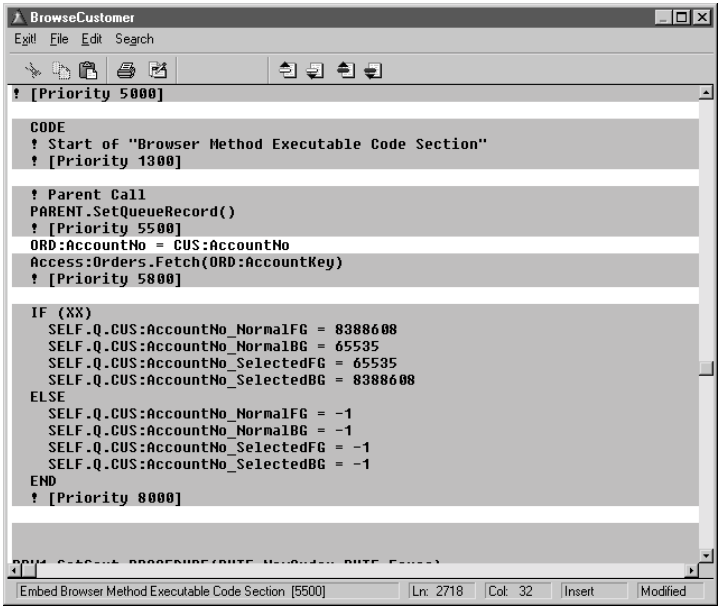
This sounds like the exact condition we want to test for.

19. **Scroll** up to our comment line and highlight all the text in it.

20. Using the Populate field toolbox that should be floating on your Clarion desktop, drop down the list and choose the Orders file. DOUBLE-CLICK on *AccountNo* (be sure not to click on the AccountKey). The field is written to the embed point, overwriting the highlighted text.

21. **Enter** =.

22. **Select** *Customer* from the drop list in the Populate field toolbox. *AccountNo* should already be highlighted. DOUBLE-CLICK on *AccountNo* regardless and it is written to the embed point. Your code should look like this:



This handles the condition where the key values must be primed before calling the Access:Fetch(ORD:AccountKey) method.

23. **Highlight** the entire *Access:Orders.Fetch(ORD:AccountKey)* method. **Press** the **Copy** button on the toolbar or **press** CTRL-C. This copies the code to the Windows clipboard.

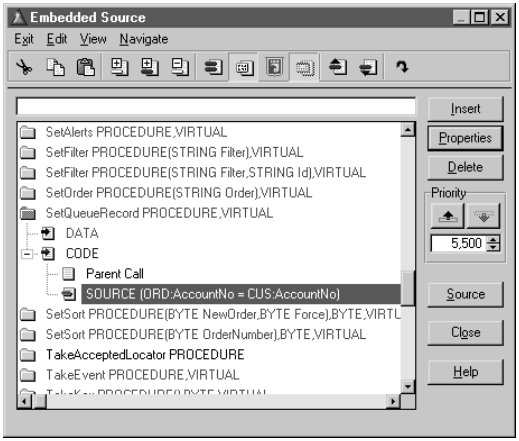
    In the code that begins with IF (XX), we could paste the *Access:Orders.Fetch(ORD:AccountKey)* then we know that would be an ideal place for the condition. Remember, it returns a zero if a record was found, anything else if not.

    However, we cannot cut or paste or do any edits in the gray areas. So we need to save our edits so far.

24. **Press Exit!** from the editor menu and **save** your edits**.**

25. **Highlight** the code template line, *Method Call, Access:Orders.Fetch,* and **press the Delete** button. Do not use the **Cut** button on the toolbar. Confirm the deletion.

    Depending on refresh options you may have used for the embed tree, you may need to **press** the **Next filled embed** button to see the embeds. Your embed tree should look like the following:



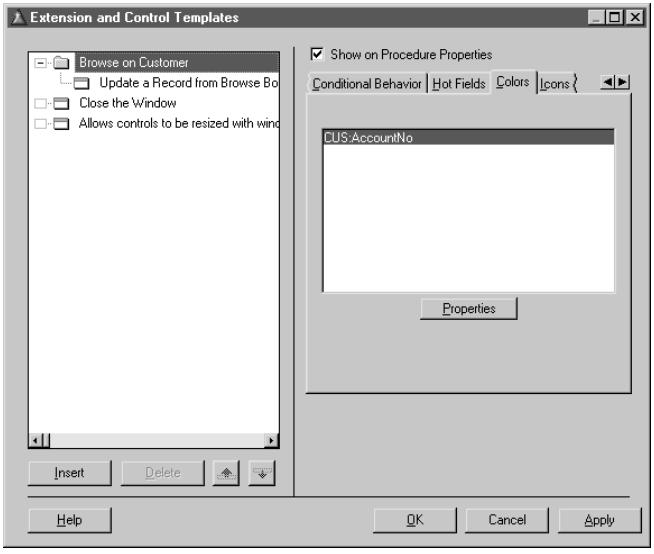
26. Close the embed tree and **save** your application when you are returned to the application tree**.**

Adding the ABC method call to the Conditional Colors

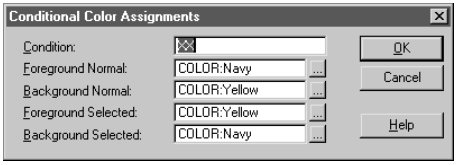
Our next step is to replace the XX condition with our method call. This way, we are letting ABC do all the work. The starting point here is the Orders application is open to the procedure tree and *BrowseCustomer* is highlighted.

1. RIGHT-CLICK on the *BrowseCustomers* procedure and **choose Extensions** from the popup menu.

2. Highlight *Browse on Customer* and then scroll right until the **Color** tab is visible.

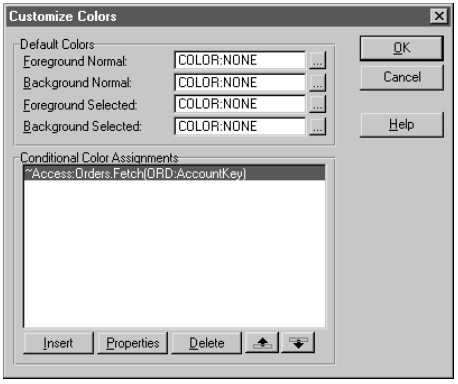


3. Press the **Properties** button twice to bring up the conditional color dialog.



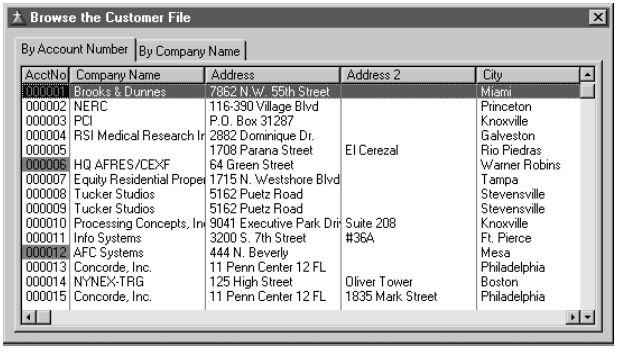
4. With *XX* highlighted, **enter** ~. This is the same as entering *NOT*.
5. **Press** CTRL-V to copy the contents of the clipboard into this entry.

   The entry should now have *~Access:Orders.Fetch(ORD:AccountKey)* in it.
6. **Press** the **OK** button to close the dialog.



What was done with these steps? Remember, the Fetch method will return a zero (success - there is a related record) or a non-zero (failure - no releated record for this customer). Since Clarion “knows” a zero is a False condition, the ~ or NOT is needed to make this a True condition. We are interested in the “not zero” or “not false” condition.

7. **Close** and **save** all dialogs, and save the application. Compile and test**.**



Summary

This lab exercise showed how you could get a dramatic change with very little actual embed code. We use only one line of code!

What we covered in this lab are:

- Using the IDE to find embed points. Other such areas are the formula editor, conditional icon usage, basically any entry in the IDE that will force a change in behavior.
- Using the embed tree and embed editor to find embed points and using code templates to discover which ABC method we would like to use. There is another code template called *SetABCProperty*, if you wish to find out which properties you would like to affect.
- Using the Help system to get information about the methods we are interested in using (and only those methods). No need to read the *Application Handbook* cover to cover.

With the above exercise done, you should now get a feel on how you can do similiar exercises in your own projects.

LAB 5

The Clarion Debugger

Introduction

What We Are Going to Accomplish

In this Lab, the goal is to understand how to use the debugger. By the time you finish this lab, you should have a better understanding of the debugger and how to use it to quickly identify problem code.

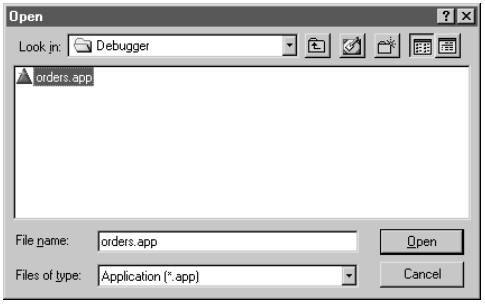
This lab has a special Orders Application. It is a simple application that has a deliberate bug added to it. In other words, it is broken! You can see the bug during the running of the program (and its not exactly a trivial one either).

Exercise Starting Point

We are now going to launch the Clarion environment and open the broken Orders application.

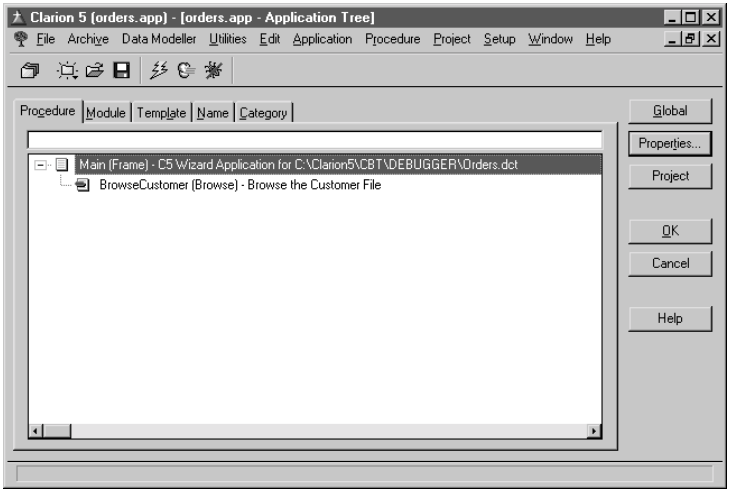
You should have just opened the Clarion development environment. The Pick dialog should be open.

1. **Press** the **Open** button.
2. **Navigate** to the *X:\Clarion5\CBT\Essentials\Lab05* folder.
3. **Select** *Application(*.app)* from the **Save as Type** drop down list box.



4. **Highlight** *Orders.app*, and **press** the **Open** button.

   The Application Tree displays.



5. **Make** and **run** the application.
6. **Open** the *Browse menu* and **choose** *Browse Customer Information File*.
7. **Highlight** a customer in the middle of the list.

   Press the Down arrow key. Nothing happens. That is definately a bug. Press the Up arrow. Scrolls up. Now press the next record toolbar button. The highlight moves up! Major bug! Press the previous record toolbar button. It scrolls up, as expected. However, all the other toolbar buttons work as expected, as do the PgUp and PgDn keys.

   The bug in this program is that the Down arrow and next record toolbar button do not work as expected.
8. **Close** the application.

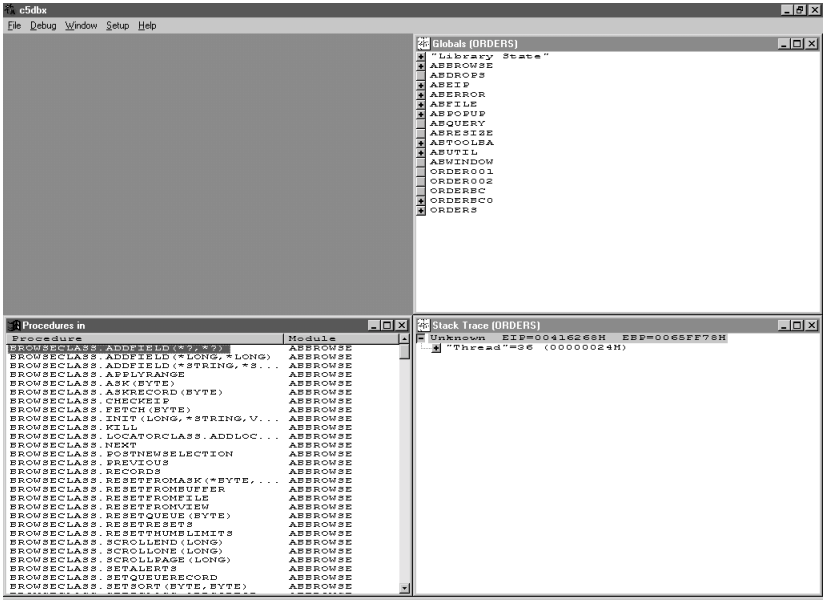
Starting the debugger

1. From the Clarion environment, **press** the **debug** button on the toolbar to start the debugger.

   You may want to shut down other applications to conserve resources because you will be running the Clarion environment, the debugger, plus the Orders program in addition to any others.

   Tip: You can run the 32-bit debugger independent of the development environment to save even more resources: run C5DBX.EXE.

   The Clarion environment verifies all files are current, then starts the debugger. The debugger opens the Globals window, the Procedures in window, and the Stack Trace (local variables) window. The debugger also opens the Trace window and the Disassembly window in the iconized state.



Arrange the Debugger Windows

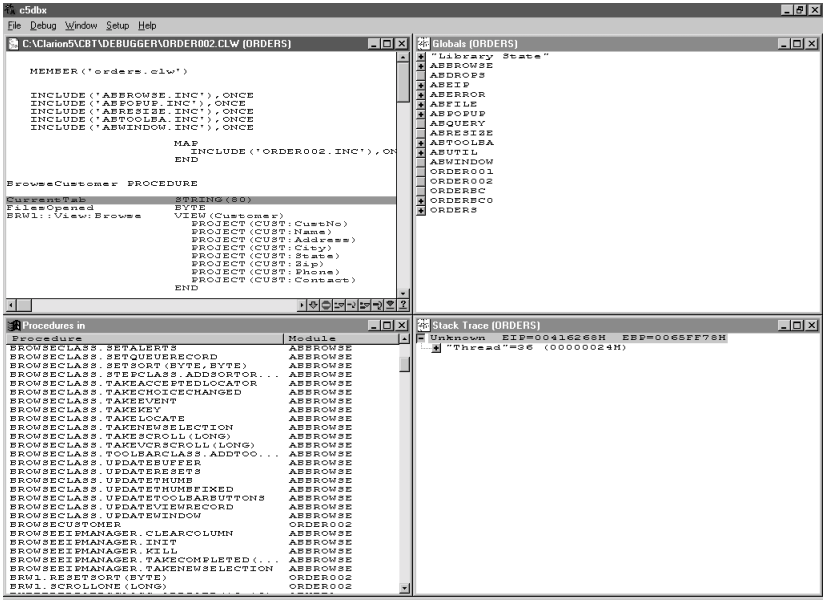
To successfully use the 32-bit debugger you should arrange the debugger windows so you can quickly access pertinent information. You should restore only those windows you need to see often, and you should arrange the windows so they are easy for you to work with.

1. **Maximize** the debugger.

   You will usually want to see several debugger windows at once, so you need as much screen space as possible.

2. In the *Procedures in* window, click *BROWSECUSTOMER* to open the source window for the BrowseCustomer procedure.

   The procedure names are alphabetical in this list. The debugger opens a source code window for ORDER002.CLW, the source module containing the *BrowseCustomer* procedure. The *Procedures in* window shows all the procedures and routines for the program you are debugging. Click on any procedure to open the source module for that procedure, if needed.



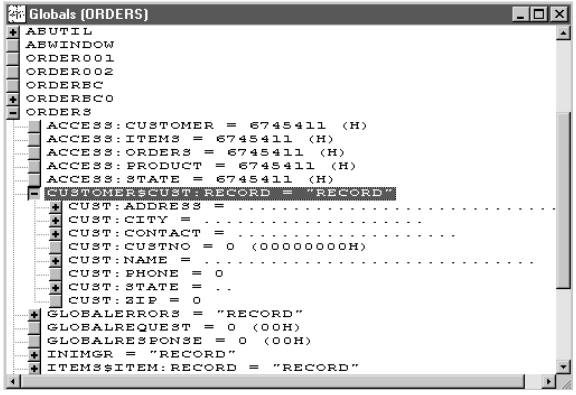
3. **Reposition**, **minimize**, and **resize** the windows so they are easy for you to work with.

   We recommend a layout that shows lots of source code, local variables (in the Stack Trace window), global variables (in the Global window), and perhaps a small overlap for easy “switching” between windows**.**

4. Ensure the global window is open.

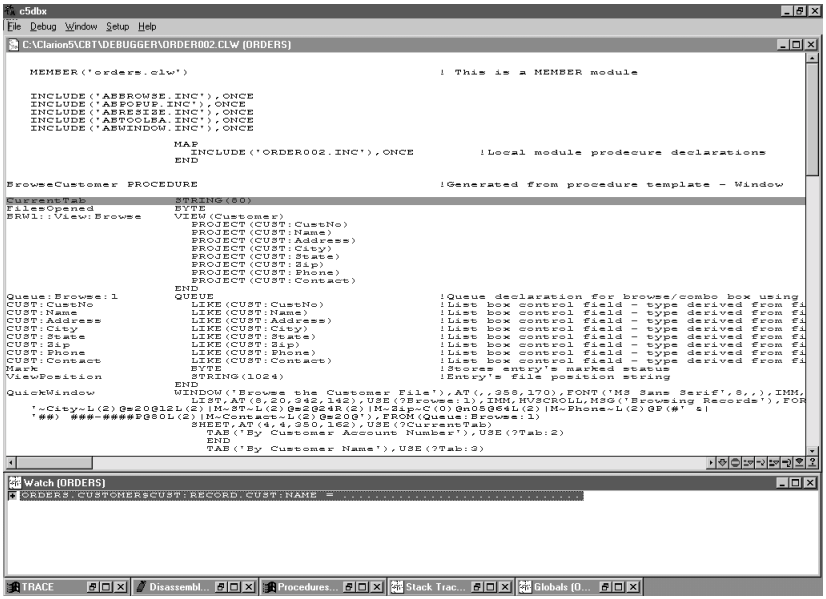
5. Find the Orders line (it may be last on the list) and click on the plus button to expand the tree for it.

   This is where you will find all dictionary files and fields used in the application. Since this application only uses the Customer file, it is the only file shown in the list.

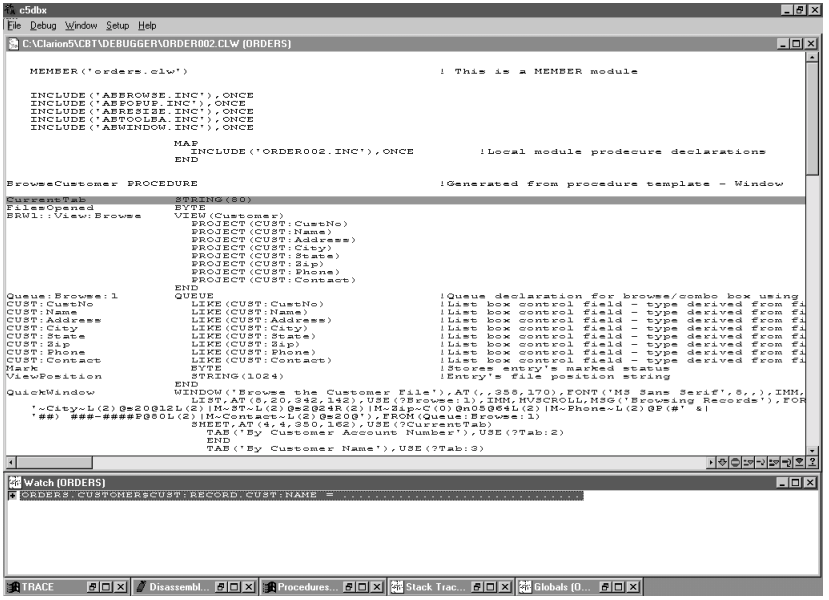


6. RIGHT-CLICK on *CUST:Name*. A popup menu displays. **Choose** *Watch window* to add this field to the watch window.

7. Adjust the size and position of the watch window so that it is shown below the source window.



Note: **The stack trace window does not yet have any local data in it as the debugger has not yet loaded the local procedure. This is because the program is suspended at program load by the debugger at this time.**



Set BreakPoints

You want to suspend (or break) the program at the point the bug occurs, then look for the cause of the problem, usually in the form of incorrect variable values or incorrect execution sequence. Remember, the bug is associated with scrolling downward.

1. **Click** on the ORDER002.CLW window to give it focus.
2. **Press** F to open the *Search for* dialog.

> **Tip:** RIGHT-CLICK **on the source code window for a popup menu of all the source window functions and their keyboard shortcuts.**

3. In the Search for dialog, type *scroll*, then **press** the **OK** button.

   This finds the *BRW1.ScrollOne* method declaration.

> **Tip:** **The search is not case sensitive.**

4. **Press** A (search again) until you come to the method code. This will be the *BRW1.ScrollOne PROCEDURE(Signed Event)* line.
5. **Find** the line that says *IF LOC:Flag* and DOUBLE-CLICK on it.

   This is the first executable line in this method. The color changes to red indicating that you have set a hard break. This means that when this line is to be executed, the debugger stops executing the program. You may set as many break points as you wish.

Once the break points are set, you should let the program run normally until it gets to the first breakpoint. Once this happens, the debugger takes over.

Note: Hard breaks appear yellow in color as the current cursor line is green. Green and Red combine to make Yellow.

6. **Choose Debug ➤** Go from the Debug menu (or **press** ALT-G).

   The program runs normally. Open the browse procedure and try to scroll down using any method that causes the bug to happen. If you have set your breakpoints correctly, the debugger will now have focus. If you have not set any breakpoints, or set them in the wrong spots, then the code is executing somewhere else.

7. **Restore** the Stack Trace Window.

   Since the debugger is in a local procedure, local variables are now available.

   The *BRW1.ScrollOne* method is already exapand for you. We can now add local variables to the watch window.

8. RIGHT-CLICK and **choose** *Watch variable* for the local variables, *Event* and *LOC:Flag*.

   These variables are added to the watch window. Also notice that the *CUST:Name* has a value in it (current highlighted customer).

9. **Press** S to step through this code, keep an eye on the watch window as you do.

   When the line Event = LOC:Flag executes, both local variables now have the same value. The current line is now on *PARENT.ScrollOne*. This means that the ABC method is to execute next. Lets step through this method and see what it is doing.

10. **Press** the *step through source* button on the toolbar in the lower right of the source window.

    The source file containing this method opens and you are at the first executable line. **Open** the Stack Trace Window if you have minimized it. *BrowseClass.ScrollOne* is expanded for you.

11. RIGHT-CLICK on *Ev* and add to the watch window.

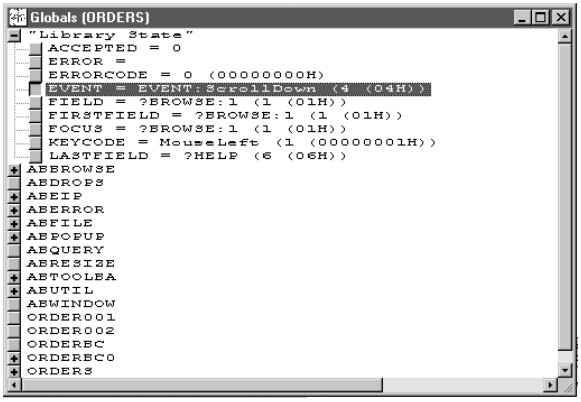
12. **Step** through the code by **pressing** on the Step source button on the toolbar.

    The line of code that says `IF Ev = EVENT:ScrollUp AND SELF.CurrentChoice > 1` executes and the code in this IF statement is next to get focus. However, the code decrements the current choice, which means that the highlight moves up!

13. **Restore** the global window and close any open tree lists.

14. **Expand** the library state tree.

    Notice the actual event that is detected.



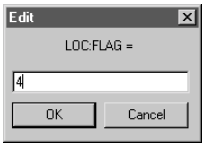
    We can see that the ABC method is executing perfectly, it would appear that it was sent wrong information. But before we start changing code, lets test our theory.

15. Choose **Debug ➤ Go** from the Debug menu (or **press** ALT-G).

    This lets the program run normally again. Now **press** the scrolldown button on the toolbar and the debugger gets focus again.

16. RIGHT-CLICK on *LOC:Flag* and this time choose Edit variable.

    Change the value from 3 to 4 and then **press** the OK button.



17. **Step** through the source again and watch how the ABC method executes the code. You will find that it now works as you would expect. Inspect the global window and look at the library state. What event is detected in this pass through the code?

18. We have found the bug.

    Do not just shutdown the debugger at this point. It should be exited gracefully.

19. **Run** the application by **pressing** ALT-G. **Exit** the application normally.

20. You may now exit the debugger.

Summary

This lab exercise showed how to run the debugger and find a bug in the program.

What we covered in this lab are:

- Clearly stating what the bug is. You cannot effectively debug an application if you cannot state what the bug is.
- How to launch the debugger.
- Arranging windows and adding variables to the watch window.
- Setting break points.
- Letting the program run normally until a break point is met.
- Watch the actual values of variables change as code executes.
- Editing values in variables and then see a different branch of code execute based on that value. This is a way to confirm a possible fix in the code.

LAB 6

OCX Controls

Introduction

What We Are Going to Accomplish

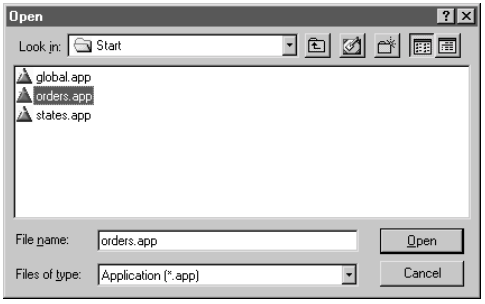
In Lab 6, the goal is to add an OCX control to the Orders application. This lab uses the free Calendar OCX control from Microsoft. If you have Microsoft Access installed on your system, you may use the control that ships with it.

The design of this lab exercise is to allow a user to look up an order date when placing an order.

Exercise Starting Point

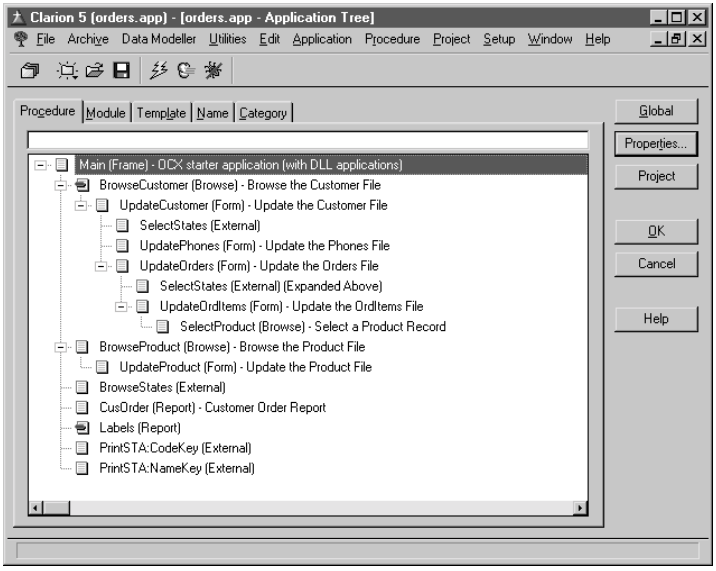
You should have just opened the Clarion development environment. The Pick dialog should be open.

1. **Press** the **Open** button.
2. **Navigate** to the *X:\Clarion5\CBT\Essentials\Lab06\Start* folder.
3. **Select** *Application(*.app)* from the **File of Type** drop down list box.



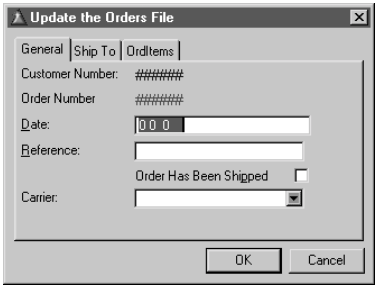
4. **Highlight** *Orders.app*, and **press** the **Open** button.

   The Application Tree displays.



5. **Highlight** the *UpdateOrders* procedure. RIGHT-CLICK on the procedure and **select Window** from the popup menu. This opens the window formatter.

   The window should look like this:



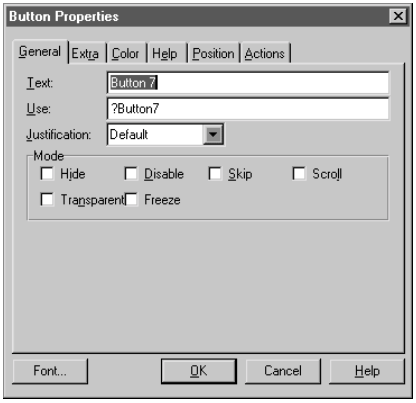
6. RIGHT-CLICK on the Date entry control and **choose Position** from the popup menu.

   This selects the **Position** tab for this control. **Choose** *Default* for the **Width** and **press** the **OK** button.

7. Using the control toolbox or the control menu, **drop** a push button control to the right of the date entry control.

   This adds a push button control to the window. This is the control that will start the process of selecting a date. However, the button needs some enhancements.

8. RIGHT-CLICK on the new push button control and **choose Properties** from the popup menu.



9. **Change** the **Text** to read *Calendar.*

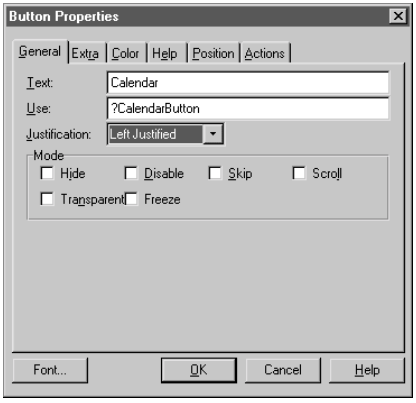
10. **Change** the **Use** variable to *?CalendarButton*.

    This makes the button easier to refer to in code.

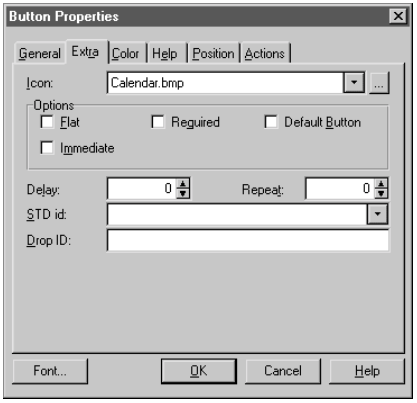
11. **Change** the **Justification** to *Left Justified*.

    Since we plan on adding an image, it would look better if it were displayed to the left of the text.

> **Tip:** **Depending on the design, you can have a button display an image only. Ensure the text entry is blank. If you wish the image to display to the right of any text, change the justification to *Right Justified.* You may also have the image appear on top of any text, by leaving the justification to *Default.***



12. On the **Extra** tab, **add** the *calendar.bmp* file found in the current folder, to the **Icon** entry field.



13. **Select** the **Position** tab. **Change** the **Width** to *Default*.
14. **Select** the **Actions** tab.
15. **Press** the **Embeds** button.
16. **Highlight** the *Accepted* embed point and **press** INSERT.
17. **Select** *Call a procedure* code template and **enter** *CalendarOCX* as the procedure name.

Press the **OK** button.

18. **Press** INSERT again and **select** a *source* embed. Ensure that it comes after the *Call a procedure* embed.
19. **Enter** the following code:

```
!Accept the date if the OCX passes one
IF GLO:Date
  ORD:OrderDate = GLO:Date
END
```

Exit and **save** the changes.

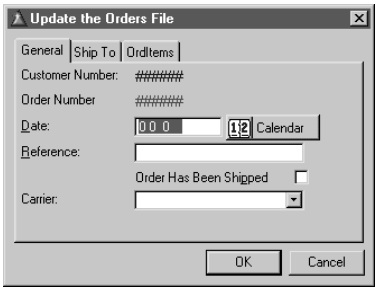
20. **Highlight** the *Selected* embed and insert this code as a source embed:

```
!Clear the global date variable
GLO:Date = 0
```

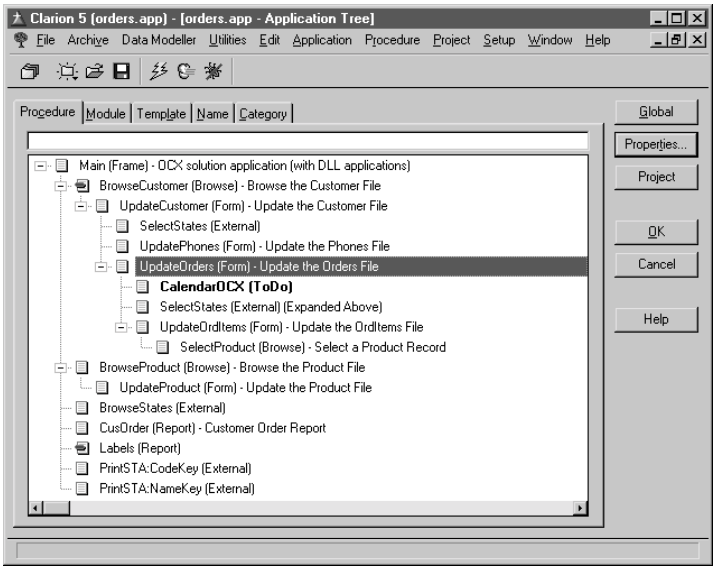
Exit and **save** the changes. You will define the GLO:Date variable in a later step.

21. **Press** the **Close** button and then **press** the **OK** button. You are returned to the window formatter.
22. With the window selected, **press** the **Properties** button and then **select** the **Extra** tab. **Check** the **Auto display** box.

Make any cosmetic changes you feel needed. Your window should look similar to this:



23. **Exit the** Window formatter. Don't forget to save your work. You are now back at the application tree and have a new ToDo procedure.



The CalendarOCX procedure

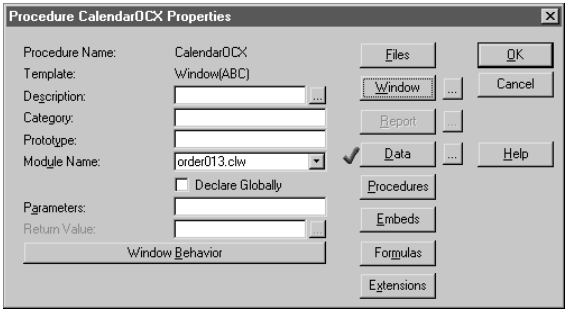
This procedure is a simple window with minimal controls.

1. DOUBLE-CLICK on the *CalendarOCX (ToDo)* procedure.

   This opens the *Select Procedure type* dialog.

2. **Choose** *Window - Generic Window Handler*. Make sure the **Procedure Wizard** box is not checked.

   The procedure properties dialog displays. Notice that there is no window for this procedure.



3. **Press** the **Window** button.

   A dialog appears asking for what type of window you would like.

4. **Choose** *Window*.

   You are presented with a blank window in the window formatter. We will add our own controls to the window.

5. **Press** the **Properties** button on the toolbar to open the window properties.

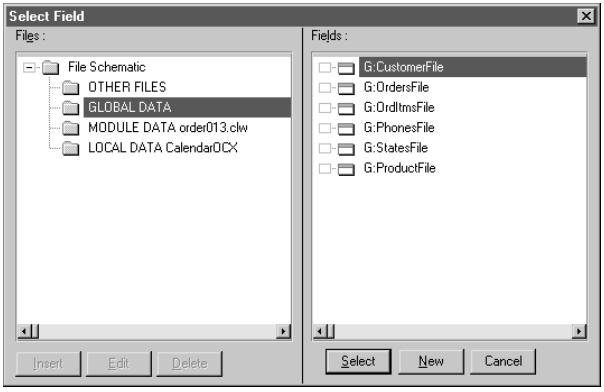
   Change the **Text** entry to *Select a date...* and **press** the **OK** button. You are returned to the window formatter.

6. **Drag** the bottom of the window down to make it square (approximately).

   We are going to add a few controls to this window and we need some room. Since the OCX will return a date, we need a variable to store the date. Lets populate this cotnrol first.

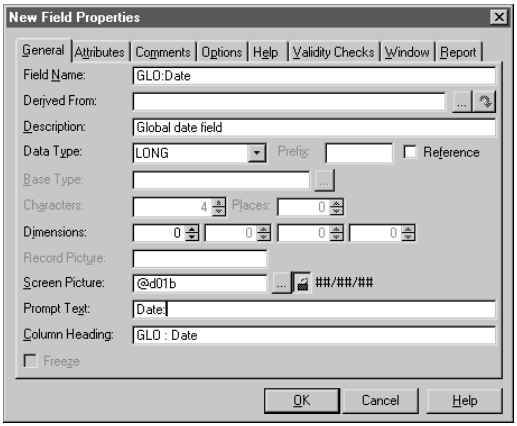
7. **Choose Populate ➤ Field** from the menu.

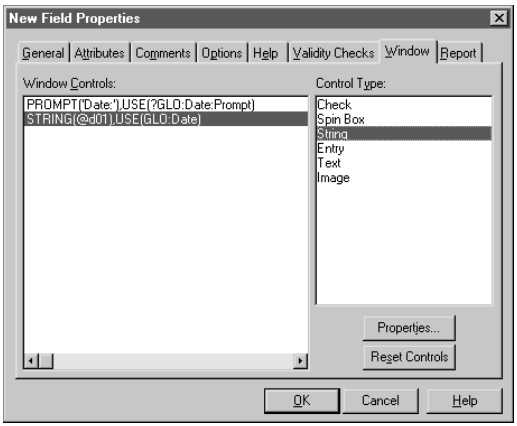
   A Select field dialog displays. We really need a global variable to store the date returned from the OCX. However, there aren't any appropriate global variables defined for this use. You can see them if you **highlight** the *Global data* band.



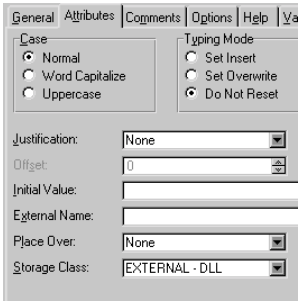
8. **Press** the **New** button to add a new variable in the global data area.

The New field dialog displays. It is the same one used in the dictionary when editing fields. Fill in the entries as shown:





Note: **If you use a data DLL (like this application does), you can define the variable there. It is still needed by this application and it would be defined exactly the same, except on the attributes tab, the *storage class* drop list must be set to *External - DLL*. This is optional and you do not need to do this for this lab.**



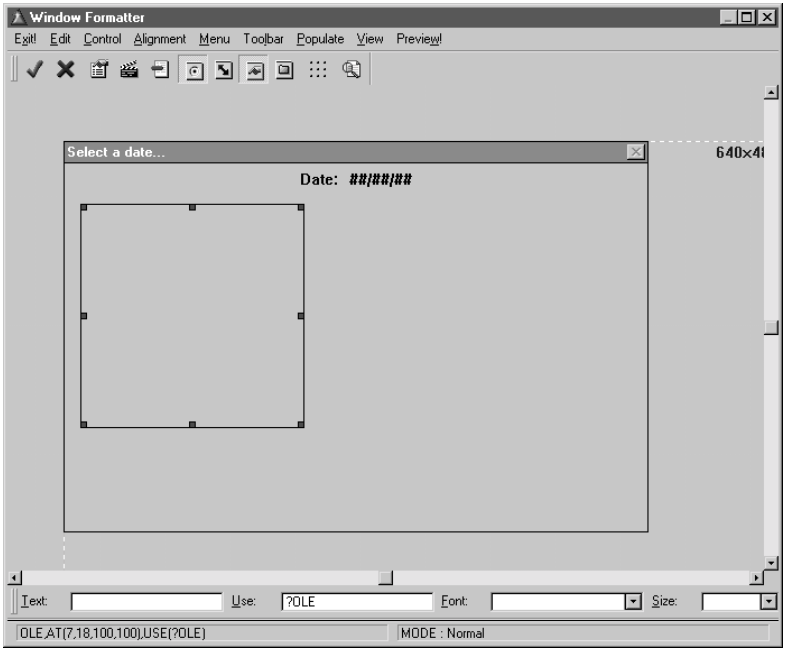
9. **Press** the **OK** button and your mouse cursor changes shape.

Drop this new control on the top-center of the window. **Drag** the controls and use alignment functions as needed. We need to add a control template now.

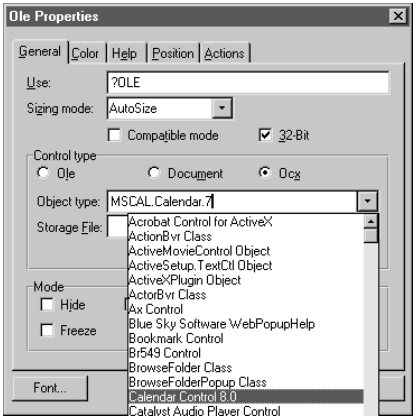
10. **Press** the **Control template** button on the control toolbox. This button looks like a flowchart.

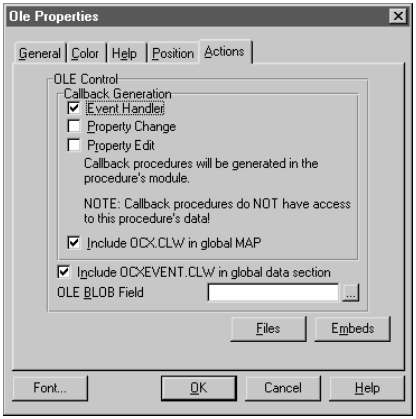
11. **Highlight** *OLE Control - OLE or OCX Control* from the *Select control template* dialog and then **press** the **Select** button.

The mouse cursor changes shape. **Drop** the control beneath the date string and just to the right of the left border of the window.



12. With the OLE control selected, **press** the **Properties** button on the command toolbar and change the entries as follows:



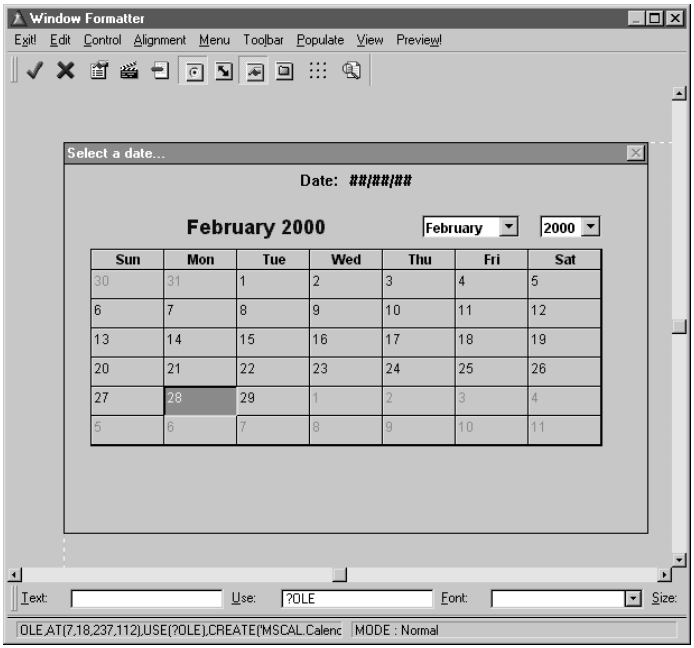


Note: **On some systems, the name of OCX control may appear as a blank in the list. If this is the case, select the blank line in the list. If the name is *MSCAL.Calendar.7,* this is correct and it will work. You may also enter the name of the control. If you wish to do this, close the drop list and enter *MSCAL.Calendar.7***

If you do not have the OCX registered on your machine, it will not appear on the list. In this case, open Explorer and navigate to the *X:\Clarion5\CBT\Essentials\Lab05\Solutions*. In this folder you will find a program called RegSrv32.EXE. Locate the MSCAL.OCX, drag and drop the OCX onto RegSrv32.EXE. You will get a message that the control was successfully registered. Once this is done, you may proceed with the rest of the lab.

13. **Press** the **OK** button and you will see the calendar display on the window.

Resize the control to fill most of the window and you could see a display similar to the following:

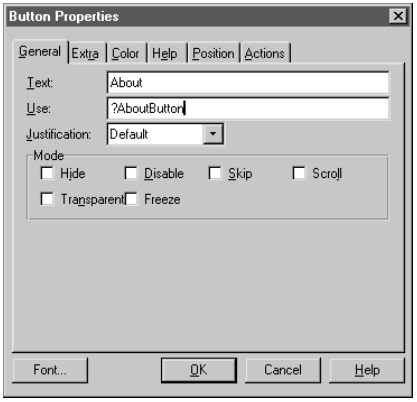


The control template used to populate this OCX has taken care of all the low-level tedium for you. See the Help topic *OCX Library Procedures* and *Callback functions* for more information.

Since we have a calendar to work with, we should add some buttons to this window to call the OCX's methods.

14. **Press** the button control on the control toolbox and drop it under the OCX control, on the left side of the window.

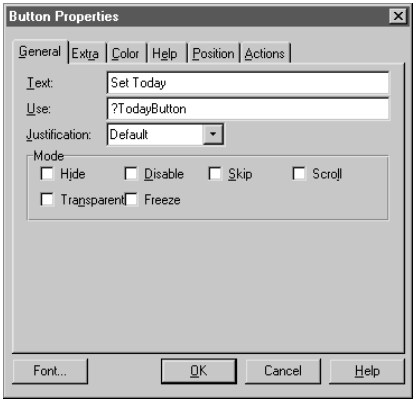
15. RIGHT-CLICK on the button control to **open** the properties dialog. **Change** it as shown below:



Press the **OK** button to close the properties dialog for the button.

16. **Press** the button control on the control toolbox and drop it to the right of the **About** button.

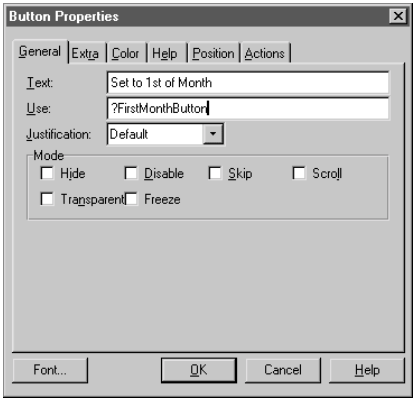
17. RIGHT-CLICK on the button control to open the properties dialog. **Change** it as shown below:



Press the **OK** button to close the properties dialog for the button.

18. **Press** the button control on the control toolbox and drop it to the right of the **Set Today** button.

19. RIGHT-CLICK on the button control to open the properties dialog. **Change** it as shown below:

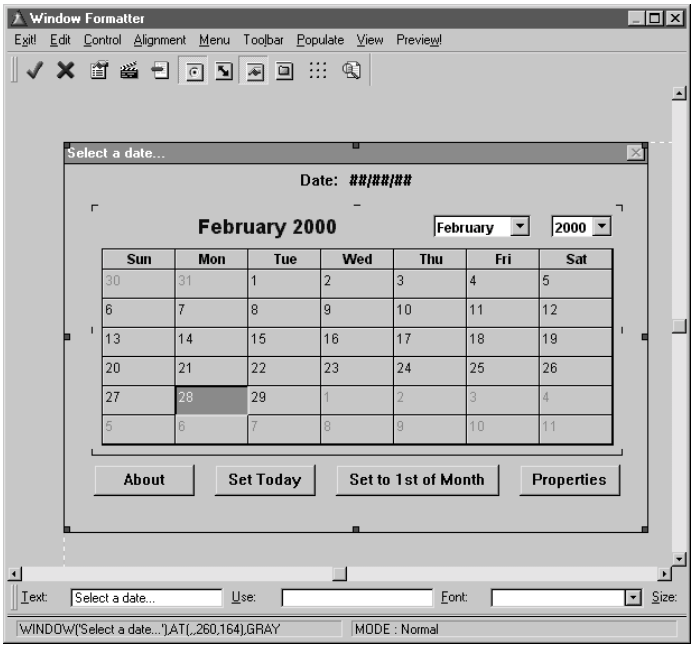


Since this button has more text, change the **Width** on the **Position** tab, to *Default*. **Press** the **OK** button to close the properties dialog for the button.

20. **Press** the button control on the control toolbox and drop it to the right of the **Set to 1st of Month** button.

21. RIGHT-CLICK on the button control to open the properties dialog for the fourth button. **Change** it as shown below:

You window should look similar to this:



The last thing we need is a button to close the window. You may need to extend the bottom of the window to give yourself some room to place the button.

22. Press the **Control template** on the control toolbox and **select** *Close button - Close the Window*.

Drop this button underneath the row of buttons we just created. Center it horizontally on the window. No further changes are needed for this button.

23. **Exit** and **save** your work.
24. **Press** the **OK** button to save the procedure properties. **Press** the **Save** button on the toolbar to save your work.

    In the next set of steps we will add some embeded source code to call the OCX methods.

Calling the OCX methods

Our Orders application is now aware of the Calendar OCX. If we run it right now, nothing will happen. We need to add some code to two procedures, *CalendarOCX* and *UpdateOrders*. First, we need to finish the *CalendarOCX* procedure.

Start with the window formatter open in the *CalendarOCX* procedure.

1. **Select** the **About** button.
2. **Press** the **Embeds** button on the control toolbar to open the embed tree for this control.
3. **Insert** a source embed in the Accepted embed point:

```
?OLE{'AboutBox'}    !Call the control's About Box
```

4. **Close** the embed tree.
5. **Select** the **Set Today** button.
6. **Press** the **Embeds** button on the control toolbar to open the embed tree for this control.
7. **Insert** a source embed in the Accepted embed point:

```
?OLE{'Value'} = FORMAT(TODAY(),@D01)    !Set control to today's date
```

8. **Close** the embed tree.
9. **Select** the **Set to 1st of Month** button.
10. **Press** the **Embeds** button on the control toolbar to open the embed tree for this control.
11. **Insert** a source embed in the Accepted embed point:

```
!Set control to begining of month
?OLE{'Value'} = MONTH(TODAY()) & '/1/' & SUB(YEAR(TODAY()),3,2)
```

12. **Close** the embed tree.
13. **Select** the **Properties** button.
14. **Press** the **Embeds** button on the control toolbar to open the embed tree for this control.
15. **Insert** a source embed in the Accepted embed point:

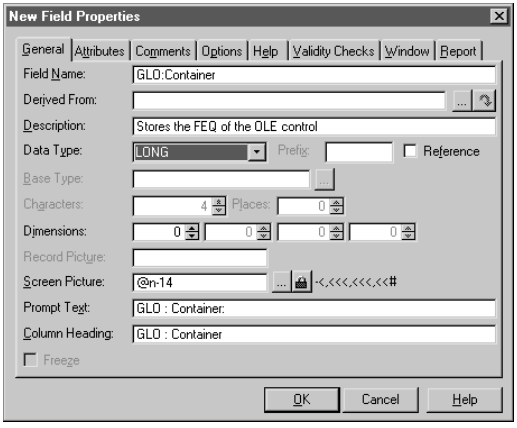
```
?OLE{PROP:DoVerb} = 0    !Open the control's properties
```

16. **Close** the embed tree.
17. **Close** the window formatter and save all changes.
18. If you are at the procedure properties dialog, **press** the **OK** button to close the dialog. **Save** your application.

Another Global variable

Since our application must communicate with the OCX control, we need a global variable to store the value of the field equate label of the OCX container. Currently, the field equate lable is local to *CalendarOCX*. The starting point is the Orders application open to the application tree.

1. **Press** the **Global** button.
2. **Press** the **Data** button.
3. **Add** a variable as shown below.



We are not concerned with display properties as this variable is used behind the scenes. **Close** the dialog and then **press** the **Cancel** button when the new field dialog displays again. **Press** the **Close** button and then the **OK** button to close the global dialog.

Finish the *CalendarOCX* procedure

We have just a few more embeds to take care of so we can complete this procedure. The Calendar OCX is capable of generating events. The problem is that the ACCEPT statement does not know about these events. So the following steps handle that condition.

The starting point of this section is the Orders application open to the application tree. The *CalendarOCX* procedure should be highlighted. It is assumed that the application is saved up to this point.

1. RIGHT-CLICK and **choose Embeds** from the popup menu.

the **2.** The embed tree is highlighted.

3. **Enter** *open* in the locator entry at the top of the embed tree. **Press** CTRL-ENTER twice.

This advances the highlight bar to the next match. What we are looking for is *ThisWindow.Open PROCEDURE, VIRTUAL*.

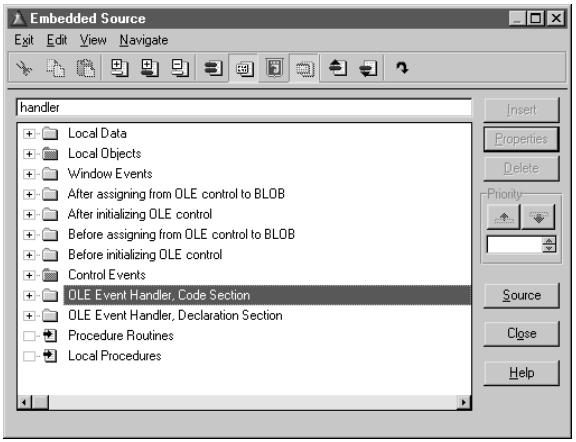
4. **Insert** a *Source* embed at the *Code* embed point. Enter the following code:

```
! Store the FEQ into the global variable
GLO:Container = ?OLE
```

This takes the field equate value and saves it to a global variable. Since there are two procedures involved, we need to know the field equate so we can send it a message (event). This is covered in the following steps.

Press the **Exit!** button and **save** your changes.

5. **Press** the **Contract all** button on the toolbar.



What we need to do here is inform the ACCEPT statement that the OCX has generated an event. We then send that event to the OLE container. This is done with one line of embedded source code.

6. **Open** the embed tree for *OLE Event Handler, Code section*. **Insert** the code into a *source* embed point:

```
POST(Event:Accepted,GLO:Container)
```

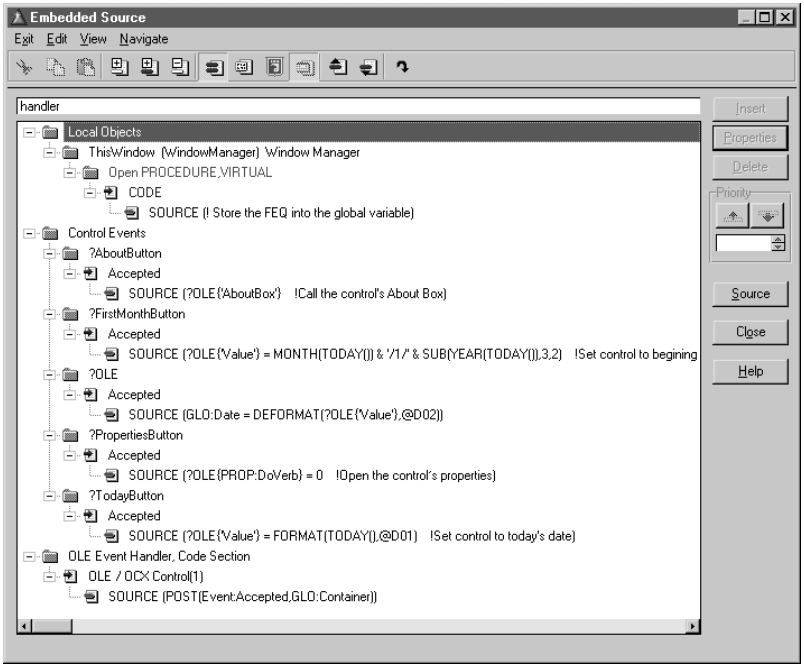
Note: **Be sure that the word *POST* is in blue, not red. If it appears red, then insert two spaces before it.**

7. **Insert** the code into a *source* embed in the *Control Events / ?OLE / Accepted* embed point:

```
GLO:Date = DEFORMAT(?OLE{'Value'},@D02)
```

The OCX returns a string (the 'Value' part) with the format of MM/DD/YYYY. The DEFORMAT function converts this to a Clarion Standard Date. Since it is a global variable, any procedure can access it.

The embed tree for *CalendarOCX* should look like this after pressing the **Show filled only** and **Expand All** toolbar buttons:



8. **Close** the embed tree when you are done and **save** the application.
9. Compile and test.

Summary

This lab exercise showed how to add an OCX control in a window.

What we covered in this lab are:

- Using the OCX control template to add the OCX we are interested in using.
- How to tie events to this control.
- Calling the OCX's methods.

LAB 7

Application Programming Interface

Introduction

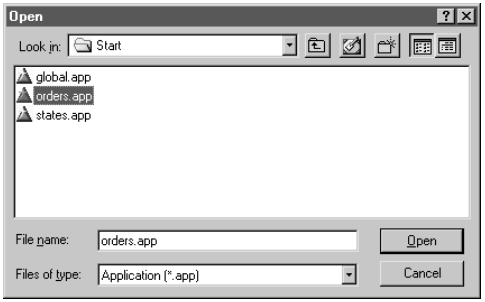
What We Are Going to Accomplish

In this short Lab, we will use a simple API call to play a wav file. It will execute when the Orders application starts. It requires your system capable of playing sound files.

Exercise Starting Point

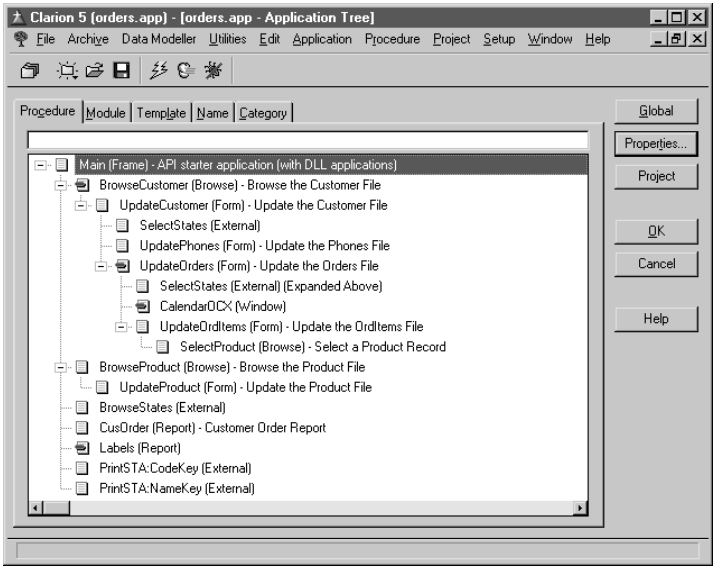
You should have just opened the Clarion development environment. The Pick dialog should be open.

1. **Press** the **Open** button.
2. **Navigate** to the *X:\Clarion5\CBT\Essentials\Lab07\Start* folder. **Select** *Application(*.app)* from the **File of Type** drop down list box.

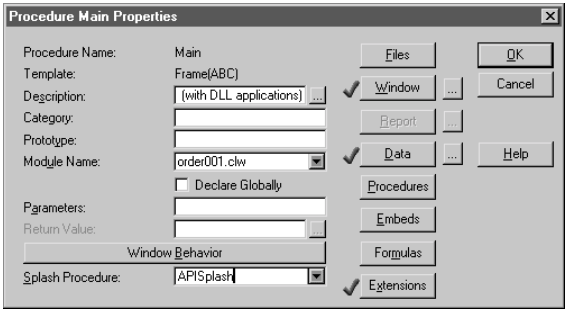


3. **Highlight** *Orders.app*, and **press** the **Open** button.

   The Application Tree displays.



4. **Highlight** the *Main* procedure and **press** the **Properties** button.
5. **Enter** *APISplash* in the **Splash procedure** drop combo control.



6. **Press** the **OK** button.

   You now have a procedure listed as a ToDo.

Using the WinAPI Viewer

Protyping the API function

In order to use any external procedure, it must be prototyped in Clarion. Once this is done, it is simply a matter of using the API call (procedure) as you would any Clarion statement. This is the heart of API coding.

Clarion ships with a utility that assists you in this step. It is called *WinAPI Viewer*. It is located in the Clarion5\bin folder.

For more details, see the *Programmer's Guide*.

Tip: **You may create your own menu in the IDE to call this utility. All you need to do is edit the C5EE.INI file using any non-Clarion editor (like Notepad). This means Clarion cannot be running as it keeps a copy of the INI settings in memory and then writes them out to the disk when you leave Clarion. Make these edits and restart Clarion:**

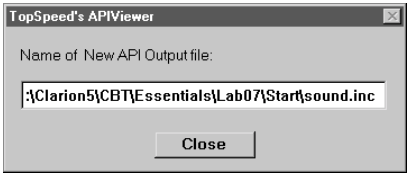
```
[User Menus]
_version=41
1=&Utilities/&Win API|WINAPI

[User Applications]
_version=41
WINAPI=C:\clarion5\bin\winapi.exe
```

1. **Start** the WinAPI utility either by running winapi.exe in the bin folder or using the method above.

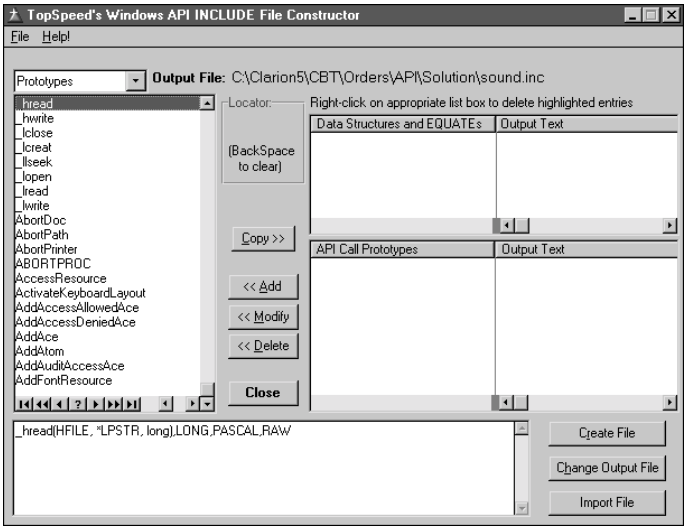
   You will see a splash window. Just click on it to close it. Another dialog opens asking for the name of the file to be generated. The default is WINAPI.CLW and the default location is the bin folder.

2. **Change** it to read like this:



3. **Press** the **OK** button to close the dialog.

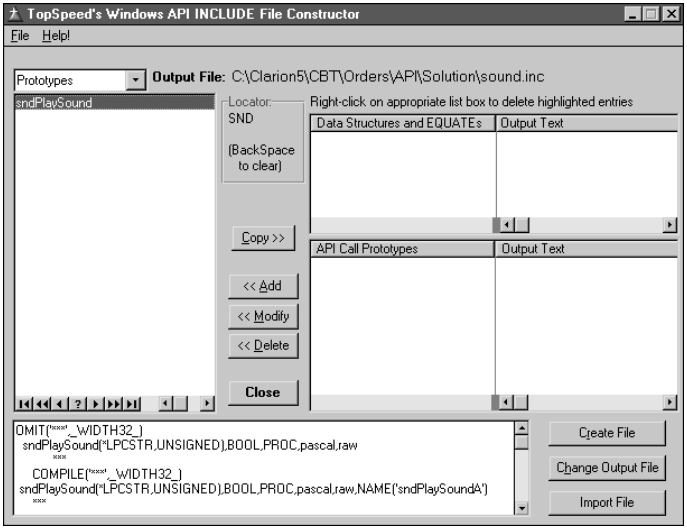
   You then see the desktop for the API generator.



Tip: Have an API reference handy such as API books or documents from MSDN. This helps you locate the API procedure you wish to use.

4. We need to locate the *sndPlaySound* procedure**, enter** *snd*.

This initializes the locator to find the procedures that start with *snd*. The locator is a filter type locator and the program now looks like the following:



Notice how everything not beginning with *snd* is filtered from the list. The bottom of the display shows the Clarion prototype.

5. **Press** the **Copy** button and the prototype is copied to the *API call prototypes* section.

   All API calls in the viewer application are prototyped as documented in API references. This makes them easier to find in your favorite API reference. If you notice, there is a data type unknown to Clarion called *LPCSTR*. If we tried to use the API procedure as it exists so far, we would get a compiler error.

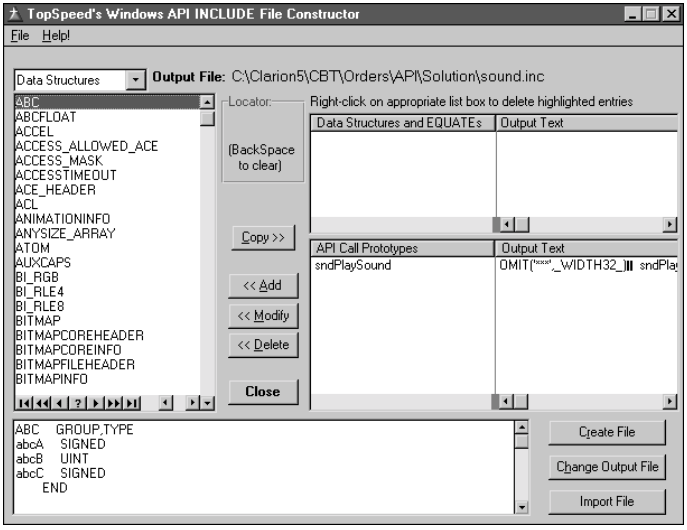
   However, the API Viewer knows about data types too.

6. **Press** BACKSPACE three times to clear the locator.

   All the procedure calls are visible again.

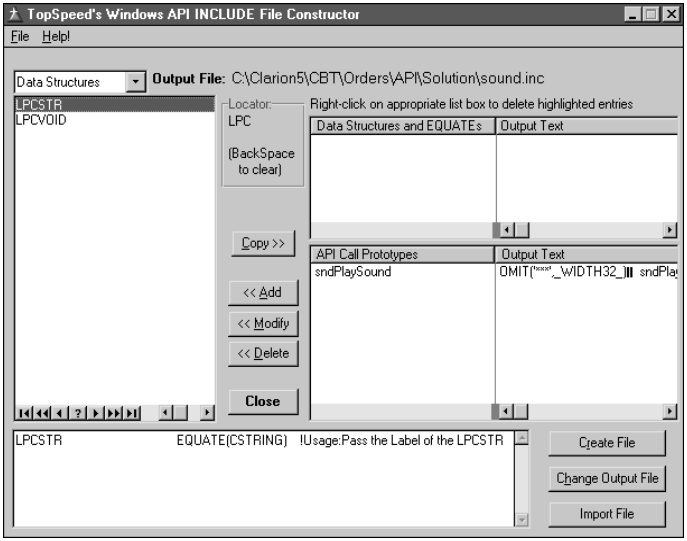
7. From the drop list at the top, **choose** *Data Structures*.

   The list now shows all data structures documented in API references and their Clarion equivilents.



8. **Enter** *LPC*.

   All data structures starting with *LPC* are shown.



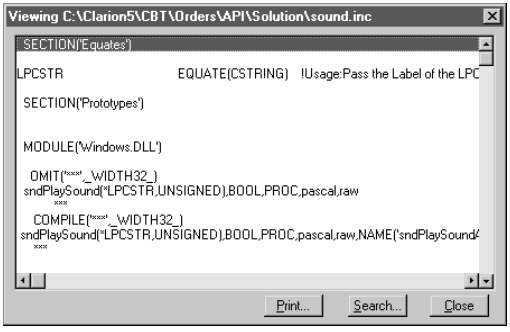
With LPCSTR highlighted, look at the bottom section of the window. You can see that it is EQUATEd to a CSTRING, a known Clarion data type.

9. Press the **Copy** button and the data structure is copied to the *Data structures and EQUATEs* section.

This is all we need at this time, but if you have more API procedures, simply repeat the above steps for each procedure and data structure you wish to use.

> **Tip: Some data structures refer to another data type that is unknown to Clarion. Find the referred data structure *before* you define the one you are interested in. This ensures that they are placed in the correct order in the list.**

10. Press the **Create File** button and you see a window showing you the prototypes you chose. This is the contents of the include file we must incorporate into our application.

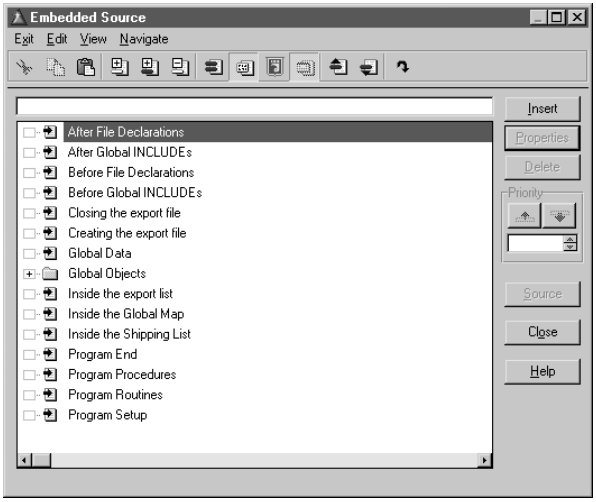


11. Press the **Close** button and quit the API Viewer application.

Including the SOUND.INC file in the application

Now that you have a file with the appropriate prototypes and data types, we need to include it in your application. You should have the Orders application open.

1. **Press** the the **Global** button.
2. **Press** the **Embeds** button.
3. **Press** the **Contract All** button on the toolbar to close up the embed tree.



4. Find the embed labeled *After Global Includes* and **press** the **Insert** button and **select** the *source* embed type.

   Enter the following code (ensuring that it does not start in column one):

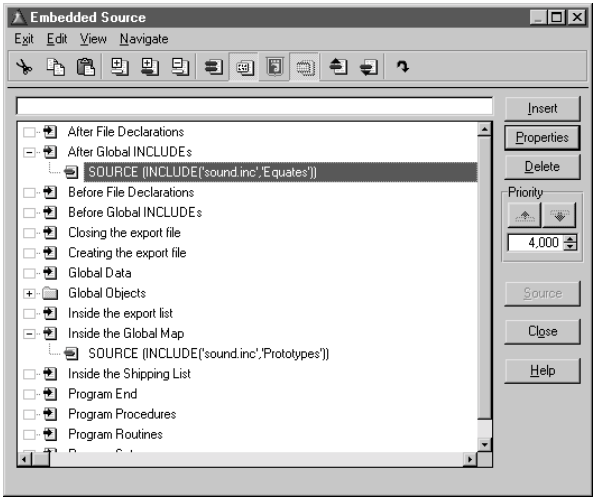
   ```
   INCLUDE('sound.inc','Equates')
   ```

   If you see INCLUDE in red, it is in column one and this is considered a data label. INCLUDE is a compiler directive.

5. **Exit** and **save** the changes. You are returned to the embed tree.
6. With the source embed highlighted, **press** the **Copy** button on the toolbar to copy this embed into the clipboard.
7. **Highlight** *Inside the Global map* and then press the **Paste** button on the toolbar to paste this embed from the clipboard.
8. **Press** the **Properties** button and change the source to read:

   ```
   INCLUDE('sound.inc','Prototypes')
   ```

9. **Save** your work and you are returned to the embed tree.



That is all you need to do to inform Clarion about this API procedure. You may now use the prototyped function like any other Clarion statement.

10. Press the **Close** button and then the **OK** button to return to the application tree.

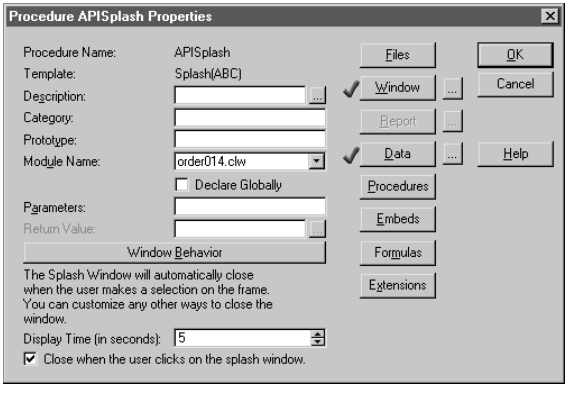
The APISplash procedure

This procedure displays when the program first starts. The default behavior for splash procedures is that it displays for 5 seconds and then closes or closes when the user presses a menu item, whichever occurs first.

1. DOUBLE-CLICK on the *APISplash (ToDo)* procedure.

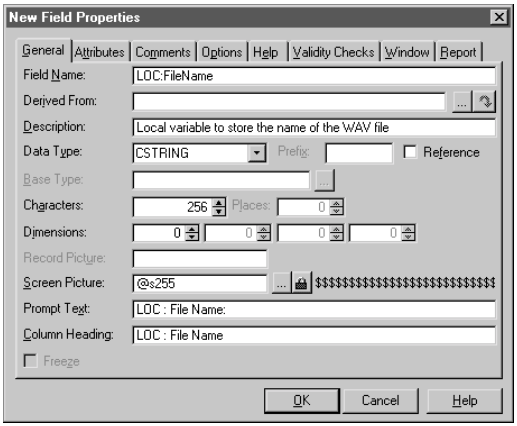
This opens the *Select Procedure type* dialog.

2. **Choose** *Splash- Splash Window*. Make sure the **Procedure Wizard** box is not checked.



3. Press the **Data** button.

Add the following local data variable:



Press the **OK** button to save the variable. **Press** the **Cancel** button to close the next *New Field Properties* window. **Press** the **Close** button to get back to the procedure properties.

4. **Press** the **Embeds** button.

   Find *Local Objects*, *ThisWindow.Open* embed. Open the tree under the *Code* embed. **Select** *ParentCall*.

5. **Press** the **Insert** button and add the following *source* embed:

   ```
   !Play the wave file
   LOC:FileName = 'start.wav'
   sndPlaySound(LOC:FileName,1)
   ```

   This API procedure expects a variable and thus the reason why the file name was passed into a variable first. Exit and save your changes.

6. **Press** the **Close** button and then the **OK** button to go back to the application tree to save your work.

7. **Compile** and **run** the application.

   When the APISplash procedure opens, you should hear the wav file playing.

Summary

This lab exercise showed how to add an API call to play a wave sound file.

What we covered in this lab are:

- Using the WinAPI utility to prototype the API procedure we want to use.
- How to include the generated prototype file in our application.
- Calling the API procedure.

LAB 8

Drag and Drop

Introduction

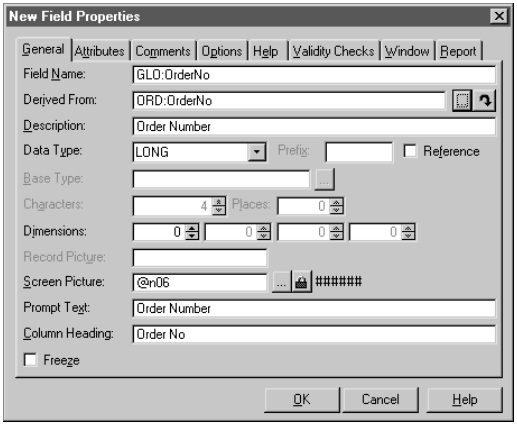
This final lab builds on the CustOrder report procedure created inLab 3, the Reports lab. In this lab we will add the ability to drag an order to a print image to start a report process.

Exercise Starting Point

Open the Orders application located in *X:\Clarion5\CBT\Essentials\Lab08\Start*. We need to add some new global variables. We will define these first.

Global variable

1. **Press** the **Global** button and then the **Data** button.
2. **Press** the **Insert** button and add this variable (you will notice the use of deriving this variable based on a dictionary field):



Press the **OK** button.

3. **Press** the **Cancel** button when the next new field dialog appears.

You are returned to the global variables list.

4. **Press** the **Close** button and then the **OK** button.

   You are returned to the application tree.

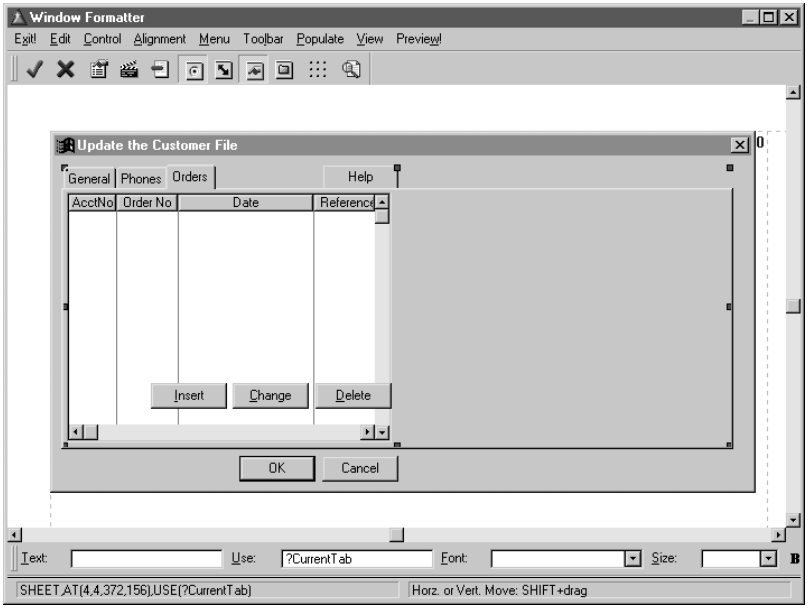
Setting up the Orders list box

1. RIGHT-CLICK on the on the *UpdateCustomer* procedure and choose **Window** from the popup menu. This will bring you into the window formatter.

   Change the width of the screen to be a bit wider. You will need to make it wide enough to store an image, so give yourself plenty of room.

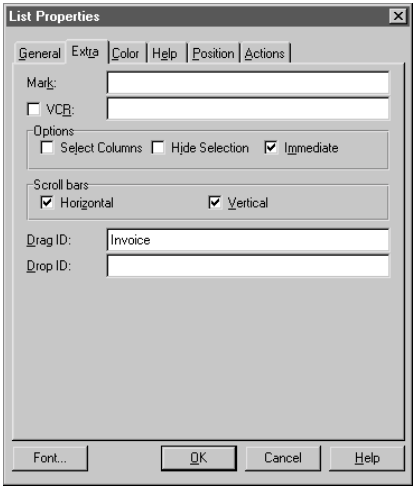
2. Stretch the sheet control to fill in the window, again just the width dimension is all we are concerned with.

   Choose the *Orders* tab.



3. RIGHT-CLICK on the *Orders* list box and **select Properties** from the popup menu. S**elect** the **Extra** tab.

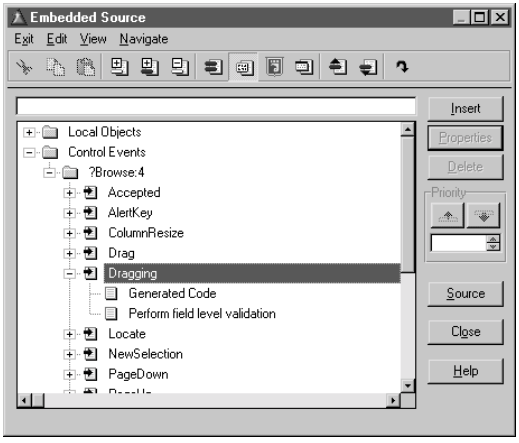
Enter a label for the **Drag ID** entry. Remember what you typed. If you cannot think of a name, then use ***Invoice*.**



Close the dialog. This action has created some new embed points for this procedure.

4. With the *Orders* list still selected, **press** the **Embeds** button on the command toolbar.

The new embeds are under *Control Event Handling*. Find the *Dragging* embed point.



Add a *source* embed type after the *Generated Code* embed:

```
!Get the current order number and pass it
ThisWindow.Update()         !Ensures we get the current info
GLO:OrderNo = ORD:OrderNo
```

If you wish to set your own mouse cursors, overriding the defaults, then you may use these optional steps:

5. **Add** another *source* embed after the above embed point:

```
!Change the mouse cursor to whatever you want
SetCursor('MyCursor.CUR')
```

6. **Add** a *source* embed to the *Drag* embed point:

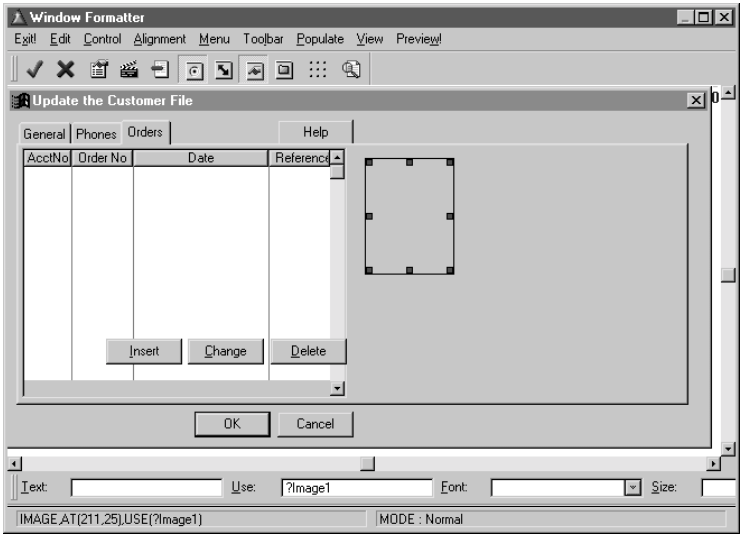
```
!Restorethe mouse cursor to defaults
SetCursor()
```

Press the **Close** button to return to the Window formatter.

This is all that is needed for the list box. We added the Drag ID name and this gave us new embed points to trap the events that occur when the user starts to drag an item and another point for when the user releases the mouse button. Now we need to add a picture.

7. **Press** the **Image** button on the control toolbox to add an image to this tab.

Drop it to the right of the list box (now you know why we made it wider).

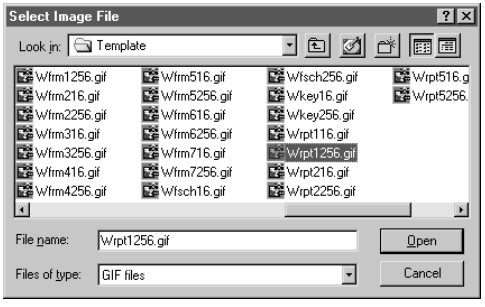


8. **Press** the **Properties** button on the command toolbox.

9. **Press** the ellipsis(...) button to look up the image name.

10. Using the file dialog, walk to the *X:\Clarion5\template* folder.

The **Files of type** drop list should be changed to *GIF files*. **Choose** *WRPT1256.GIF*



11. **Press** the **Open** button to use this image. **Press** the **OK** button when you are returned to the properties dialog.

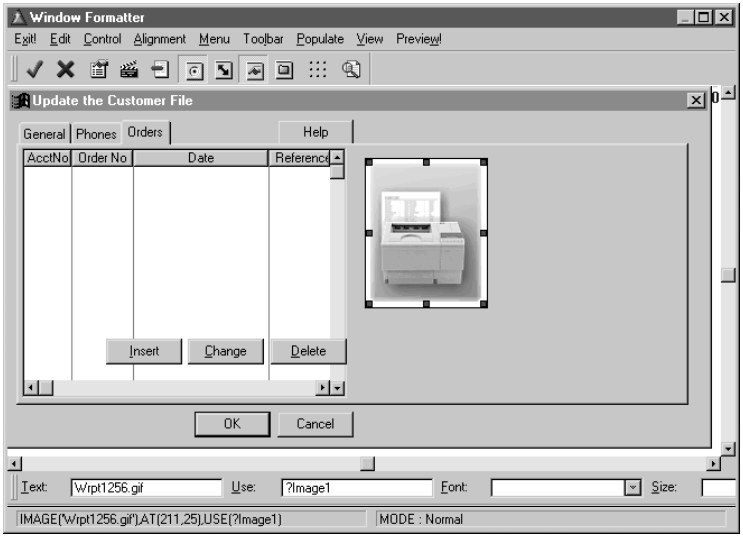
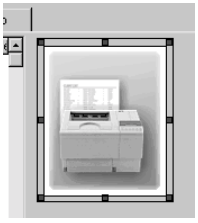


Image controls cannot trap events. However, we need to trap certain events when the order is dropped onto the printer image. In order to trap these events we will use a region control. We will drop a region control around the image. The region control will generate the needed events at runtime.

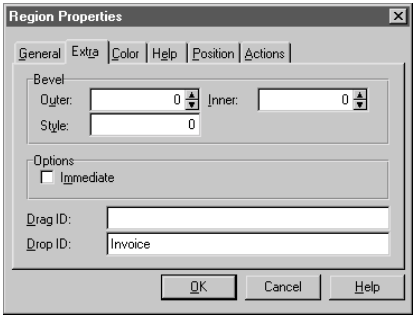
12. **Select** the **Region** control (next to the image control – looks like a dotted outline of a box) from the control toolbox.

Drop it on top of the image control. **Resize** the region so that it overlaps and slightly extends the dimension of the picture.



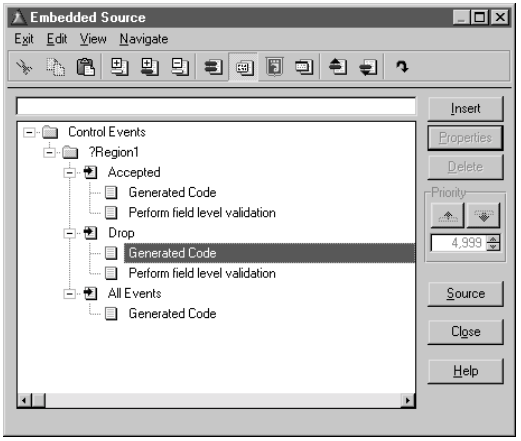
13. RIGHT-CLICK on the region control and **select Properties** from the popup menu.

On the **Extra** tab , add a label to the **Drop ID**. This should be the same as you did for the list box Drag ID. If you used *Invoice*, then it should be *Invoice* here too.

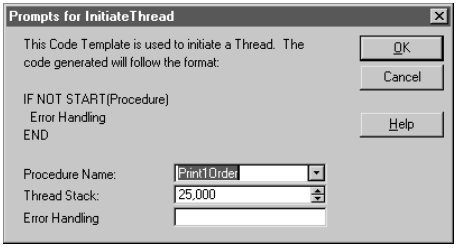


14. **Press** the **OK** button to save your changes.

15. With the region control still selected, **press** the **Embeds** button from the command toolbar.



16. Find the Drop embed point and add a code template, *Initiate Thread Execution*. **Enter** *Print1Order* for the **Procedure Name**.



17. **Press** the **OK** button to save your changes.

Since we are reporting on the same file we are viewing, there will be nothing to display on our browse when the report is done . This is because the browse is range limited by Customer and the pointer for the record is now at some other customer ID. We need to reset the browse list to get back our settings.

18. **Add** this *source* code after the *Initiate Thread Execution* code template:

```
!Reset the browse back to original range limits
ThisWindow.Reset(True)
```

19. **Close** all dialogs until you are back to the window formatter.

    You may wish to add images to the other tabs (they now have a big empty space) so they look better. If the width of this window and sheet control are too wide, simply shrink the width until it looks right. **Close** the window formatter when you are done.

20. **Save** the application and your work.

Setting up the Print1Order Report

Let's setup the Print1Order report procedure. You should be at the *Orders* application tree.

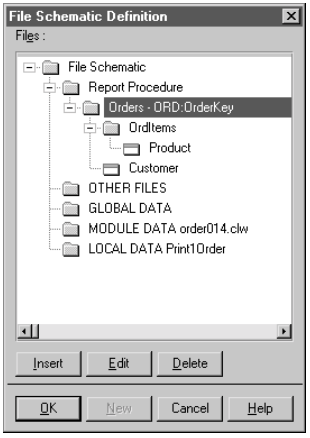
1. From the application tree, find the *CusOrder* report procedure and **highlight** it. From the **Procedure** pulldown menu, **choose Copy**. **Enter** *Print1Order* as the new **Procedure Name** and **press** the **OK** button.

2. DOUBLE-CLICK on the *Print1Order* procedure to bring up the procedure properties window.

3. **Press** the **File** button and remove all files listed.

   We need to add them back in a slightly different order. You should now have a *ToDo* in the file schematic. Whatever you do, *do not exit* at this point! If you do, all controls from the files will be removed from the report.

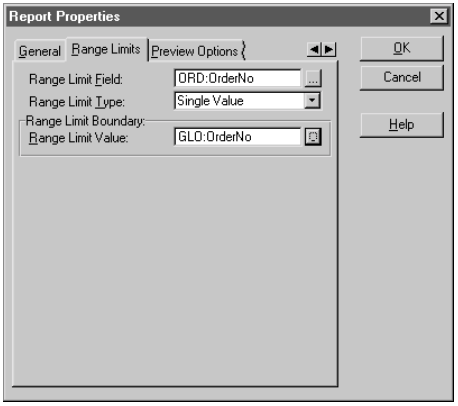
4. **Add** the *Orders* file as the primary file and set its key to *ORD:OrderKey* (set this from the **Edit** button). With the *Orders* file highlighted, **add** the *Customer* and *OrdItems* files. **Highlight** *OrdItems* and **add** the *Products* file. Your file schematic should look like this:



5. **Press** the **OK** button to save your changes.

6. **Press** the **Report Properies** button and **select** the **Range Limit** tab.

   The range limit field is *ORD:OrderNo*. The range limit type is *Single Value*. The Range Limit Value is *GLO:OrderNo*.



   Press the OK button several times until you are returned to the application tree.

7. Make, run, and test the application by dragging a customer's order and dropping it on the printer image.

Summary

This lab exercise showed how to add drag and drop functionality to an application.

What we covered in this lab are:

- Putting a Drag ID on the source control to identify what is the source for dragging.
- Adding a Drop ID on the target control to indentify what control receives a drag event.
- New event embed points once these IDs are added to a control.
- How to copy a working procedure to minimize editing.

The Next Step

This concludes the Interactive Self-Study Lab Exercises.

Our Documentation Staff has attempted to give you a good overall understanding to the powerful development world of Clarion. However, there is more.

We will continue to create Interactive Self-Study seminars that cover the more advanced tasks you can perform with Clarion.

Now, where can you go from here while you are waiting for the next CD???

There are many additional resources available to help you learn Clarion:

Online Help

The first place to look for more information is the How do I ... ? section in the on-line Help. These topics answer many of the common questions that newcomers to Clarion have. Press the How do I ... ? button on the Help Contents page to get to this section.

The Help system in Clarion is very extensive. The hypertext help appears when you press the F1 key, or choose one of the commands on the Help menu. The on-line help is arranged by dialog box, to provide you with precise context-sensitive help when you need it.

Printed Documentation

There are also a number of books included with Clarion (printed and/or on CD in .PDF format):

Getting Started
The Getting Started book is your first introduction to the Clarion environment. It includes a Quick Start Tutorial and an introduction to Clarion programming

Learning Clarion
This book contains two step-by-step tutorials. The first is the Application Generator tutorial which provides a thorough introduction to all the tools in the Development Environment. The second introduces the Clarion programming language itself.

Application Handbook
The handbook to the Clarion tools that you'll use the most during your application development. It describes the templates that ship with this product, and fully documents the Application Builder Class (ABC) Library used in the template-generated code. The full text of the Application Handbooks ABC Library Reference is also in on-line Help. When working in the Text Editor, place the insertion point on any ABC method or property, and then press the F1 key to view help for the item.

Language Reference
The complete guide to the Clarion language. This book provides descriptions of all Clarion language statements, with examples for each. The Language Reference is organized by functional topics. The full text of the Language Reference is also in on-line Help. When working in the Text Editor, place the insertion point on any Clarion language statement or function, and then press the F1 key to view help for the item.

User's Guide
Provides a task-oriented description of the development environment, arranged by its major components (on CD in .PDF format in the Professional Edition).

Programmer's Guide
A collection of in-depth articles on various aspects of Clarion language programming and articles on customizing the development environment (on CD in .PDF format in the Professional Edition). It also contains the complete guide to Clarion's Template language, providing descriptions of all its statements and functions, with examples for each, clearly demonstrating how to write your own templates. This book also provides in-depth information on all the file drivers available for Clarion.

ReportWriter User's Guide
Documents Clarion's ReportWriter for Windows™—a stand-alone end-user report writing tool (on CD in .PDF format in the Professional Edition—also available as a separate product).

Enterprise Tools
Documents the Data Dictionary Synchronizer, Version Control Team Developer, and Data Modeller tools. This book also documents the Business Math Library (standard business and statistics functions) and Wise for Clarion (an end-user installation set creation tool). This book is only in the Enterprise Edition (some components are also available as separate products).

Wizatron Handbook
Documents the Wizatrons, Clarion's next-generation wizard technology, which learn how you work and become your automated programming assistants. This book is only in the Enterprise Edition.

Master Index
A complete index listing for all printed and .PDF documentation with that edition of Clarion (printed only in the Enterprise Edition—on CD in .PDF format in all other editions).

All books on CD in .PDF format are accessible through the Adobe Acrobat Reader program, which you can also install from the Clarion CD. These .PDF format files are full-text searchable (and fully-indexed for fast searching) and can be printed on your printer if you wish.

Important: if any part of the on-line help text conflicts with the printed documentation, the information in on-line help should take precedence. TopSpeed Corporation makes every reasonable effort to ensure the printed documentation is up to date. However, the lead-time required by printers may create a lag in the documentation; while we can update the help files that ship concurrently with a product revision, printed materials must "catch up" later.

The TopSpeed Web Site

You can find other Clarion resources on the Internet by visiting TopSpeed's site on the World Wide Web:

www.topspeed.com

This web page also provides access to the TopSpeed KnowledgeBase, the latest news and announcements, and useful downloads.

News Groups

TopSpeed's internal news server offers newsgroups for all TopSpeed products. On this news server you can exchange ideas with other Clarion programmers as well as receive help from Team TopSpeed members and TopSpeed employees.

You can use any newsreader (e.g., Outlook Express, Netscape Collabra, or FreeAgent) and log into tsnews.clarion.com and subscribe to the groups for the products you want to discuss.

Education Resources

There is absolutely no better way to continue with Clarion training than with TopSpeed's hands-on educational courses. We offer many options to meet your individual requirements.

Essentials
Developed for users who wish to learn the essentials of the Clarion language and its new OOP implementation with the Application Generator Source embed environment.

Master's Series
Structured for experienced users who are ready to extend their skills to their fullest potential.

ReportWriter
Designed for end users or programmers to provide a powerful graphical user interface allowing rapid development of many types of reports.

On-site Training
Customize a training course to suit your corporate needs.

Certification
Become a Clarion Certified Developer.

Visit our web site for current seminar dates, locations, information about on-site training, and full course descriptions.

Call TopSpeed Educational Services for details at (800) 354-5444 or (954)785-4555, ext. 445

INDEX

TOPSPEED LICENSE AGREEMENT FOR FUNDAMENTALS OF CLARION 5 INCLUDING INDEPENDENT SELF STUDY LAB EXAMPLES

IMPORTANT-READ CAREFULLY: This License Agreement (“LICENSE”) is a legal agreement between you (either an individual or a single entity) and TopSpeed Corporation for the TopSpeed software product identified above, which includes computer software and may include associated media, printed materials, and “on-line” or electronic documentation (“SOFTWARE PRODUCT”). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this LICENSE.

SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. GRANT OF LICENSE. This LICENSE grants you the following rights:

Applications Software. You may install and use one copy of the SOFTWARE PRODUCT, or any prior version for the same operating system, on a single computer.

2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

Limitations on Reverse Engineering, Decompilation, and Disassembly. You may not reverse engineer, decompile, translate or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

Separation of Components. The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

Rental. You may not rent, lease, or lend the SOFTWARE PRODUCT.

Support Services. TopSpeed may provide you with support services related to the SOFTWARE PRODUCT (“Support Services”). Use of Support Services is governed by the TopSpeed polices and programs described in the user manual, in “on-line” documentation, and/or in other TopSpeed materials. Any supplemental software code provided to you as part of the Support Services shall be considered part of the SOFTWARE PRODUCT and subject to the terms and conditions of this LICENSE.

Software Transfer. You may permanently transfer all of your rights under this LICENSE, provided you retain no copies, you transfer all of the SOFTWARE PRODUCT (including all component parts, the media and printed materials, and this LICENSE), and the recipient agrees to the terms of this LICENSE.

Termination. Without prejudice to any other rights, TopSpeed may terminate this LICENSE if you fail to comply with the terms and conditions of this LICENSE. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

3. COPYRIGHT. All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, and “applets” incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by TopSpeed or its suppliers. The SOFTWARE PRODUCT is protected by copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material except that you may install the SOFTWARE PRODUCT on a single computer provided you keep the original solely for backup or archival purposes. You may not copy the printed materials accompanying the SOFTWARE PRODUCT.

4. DUAL-MEDIA SOFTWARE. You may receive the SOFTWARE PRODUCT in more than one medium. Regardless of the type or size of medium you receive, you may use only one medium that is appropriate for your single computer. You may not use or install the other medium on another computer. You may not loan, rent, lease, or otherwise transfer the other medium to another user, except as part of the permanent transfer (as provided above) of the SOFTWARE PRODUCT.

5. U.S. GOVERNMENT RESTRICTED RIGHTS. The SOFTWARE PRODUCT and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is TopSpeed Corporation, 150 East Sample Road, Pompano Beach, Florida 33064.

MISCELLANEOUS

If you acquired this product in the United States, this LICENSE is governed by the laws of the State of Florida.

If this product was acquired outside the United States, then local law may apply.

Should you have any questions concerning this LICENSE, or if you desire to contact TopSpeed for any reason, please contact the TopSpeed distributor serving your country, or write: TopSpeed Corporation, 150 East Sample Road, Pompano Beach, Florida 33064.

LIMITED WARRANTY

LIMITED WARRANTY. TopSpeed warrants that (a) the SOFTWARE PRODUCT will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt, and (b) any Support Services provided by TopSpeed shall be substantially as described in applicable written materials provided to you by TopSpeed, and TopSpeed support engineers will make commercially reasonable efforts to solve any problem issues. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the SOFTWARE PRODUCT, if any, are limited to ninety (90) days.

CUSTOMER REMEDIES. TopSpeed's and its suppliers' entire liability and your exclusive remedy shall be, at TopSpeed's option, either (a) return of the price paid, if any, or (b) repair or replacement of the SOFTWARE PRODUCT that does not meet TopSpeed's Limited Warranty and which is returned to TopSpeed with a copy of your receipt. This Limited Warranty is void if failure of the SOFTWARE PRODUCT has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE PRODUCT will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Outside the United States, neither these remedies nor any product support services offered by TopSpeed are available without proof of purchase from an authorized international source.

NO OTHER WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, TOPSPEED AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE PRODUCT, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

LIMITATION OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL TOPSPEED OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE

SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF TOPSPEED HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, TOPSPEED'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS LICENSE SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR U.S.$5.00; PROVIDED, HOWEVER, IF YOU HAVE ENTERED INTO A TOPSPEED SUPPORT SERVICES AGREEMENT, TOPSPEED'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.