Lesson 5: Adding the Ordering Methods

If you remember from earlier when you reviewed it in the DB Server Editor, the Customer data server has two associated orders, CUSTNUM.NTX and CUSTNAME.NTX. As a final enhancement to the Order Entry application, we'll create methods to switch the controlling order for the Customer data server between these two.

To accomplish this, we will first add two commands to StandardShellMenu's View menu, "By Name" and "By Number," and link them to events named "ByName" and "ByNum," respectively. (These events represent the two methods, ByName() and ByNum(), to be associated with the two menu commands—more about these later.)

Note that because the By Number menu command represents CUSTNUM.NTX, which is the default controlling order, it will be checked when the menu is initially displayed, indicating that the records in the data window are initially sorted by number.

We're adding the two new menu commands to StandardShellMenu, so they're going to be displayed as soon as the user opens one or more .DBF files in the application. However, these menu commands are specifically designed to work with the Customer Orders window and should only be available when that window is open.

Therefore, we'll disable the By Name and By Number commands when we create them, so that when StandardShellMenu is first displayed, the commands appear grayed, or dimmed. We'll then add code to the CustOrd window's Init() method to enable the commands when the Customer Orders data window is opened.

Note: An *enabled* menu command can be selected by the user. A *disabled* menu command appears dimmed and cannot be selected; it remains unavailable until it is enabled by the application.

We will also import two methods, named ByName() and ByNum(), which contain the necessary code to change the controlling order for the data displayed in the Customer Orders window: ByName() changes the controlling order to CUSTNAME.NTX, and ByNum() changes it to CUSTNUM.NTX. Note that these are the *event-handling methods* associated with our two new menu commands; these methods also contain code to check (and uncheck) the appropriate menu command so that the View menu always reflects the current order in the window.

Modifying the Menu

Let's start, then, by customizing StandardShellMenu to add the two new menu commands. You should still be in the Standard Menus Entity Browser, so all you need to do is double-click on the StandardShellMenu menu entity to load it into the Menu Editor.

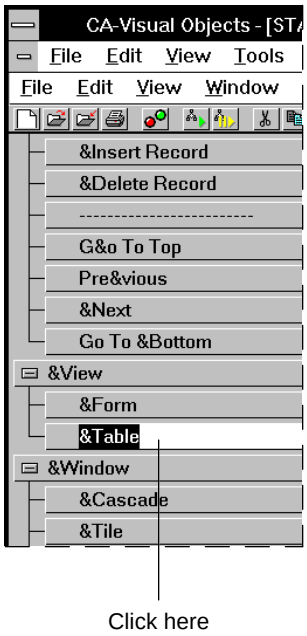
Adding Commands

Since the methods that we are going to add change the manner in which the data is displayed, it makes sense to put the commands on the View menu.

Note that the two new menu commands are distinct from the other commands on the View menu. Therefore, we're going to add a separator to the menu before we actually add the commands. (In Windows applications, separators are used to logically group items within a menu.)

Follow these steps to insert a separator and the two new menu commands at the bottom of the View menu:

1. Use the scroll bar in the Menu Editor to scroll down to the View menu, then click on the line containing the text &Table.



2. From the topmost Edit menu, choose the Add Item command, then choose Add Separator from the resulting pop-up menu.

   Note: Remember, the menus in the lower menu bar are just prototypes of the menus you are creating in this editor.

This immediately adds a separator to the menu structure:

3. Press Enter to create a new, blank menu item, and type **By &Name** in the new line.
4. Press Enter again, and type **By Nu&mber** in the next new line.

The new menu commands are now part of the menu structure; next, you need to define properties for them.

Defining Menu Properties

In order to operate properly, the program needs to know what event names the new menu commands are associated with, as well as what their initial status should be. We'll also define status bar descriptions for them.

By Name

Let's start with the By Name menu command:

1. Click on the By &Name line in the menu structure.
2. In the Properties window, click on Event Name, type **ByName**, and press Enter.

This is the name of the method that this menu command should be associated with; it will switch the controlling order to CUSTNAME.NTX. (You'll import this method later.)

3. Click on Description, type **View orders by customer name**, and press Enter.
4. Click on the Init. Enabled property, click on the down-arrow button, and choose No from the resulting list box.

   This changes the value of this property to "No," thereby causing the By Name menu command to be disabled when the menu is first displayed. (Later, we'll add code to the CustOrd:Init() method to enable this menu command when the CustOrd data window is open.)

By Number

Now, you will follow similar steps for the By Number menu command:

1. Click on the By Nu&mber line in the menu structure.
2. In the Properties window, click on Event Name, type **ByNum**, and press Enter.

   This is the name of the method that this menu command should be associated with; it will switch the controlling order to CUSTNUM.NTX. (You'll import this method later.)
3. Click on Description, type **View orders by customer number**, and press Enter.
4. Click on the Init. Enabled property, click on the down-arrow button, and choose No from the resulting list box.

   Like the By Name menu command, By Number applies only to the CustOrd data window, so it will be disabled when the menu is initially displayed. (Later, we'll add code to the CustOrd:Init() method to enable this menu command when the CustOrd data window is open.)
5. Click on the Init. Checked property, click on the down-arrow button, and choose Yes from the resulting list box.

The default controlling order for the CustOrd data window is CUSTNUM.NTX. Therefore, the By Number menu command should be checked when the CustOrd data window is initially displayed. (The ByName() and ByNum() methods we'll import later will contain code to appropriately toggle the check mark between the two menu commands.)

That's all you need to do to the menu (if you like, you can use the preview menu bar to see the new commands). When you are finished, close the Menu Editor, save your work, and close the Standard Menus Entity Browser.

Note: You will not be able to save your menu entity if there are any *empty* menu items. Use the Edit Delete Item menu command to delete a blank menu item, if necessary.

Creating the Methods

The next step is to add the functionality for the By Name and By Number menu commands. Since both the data server on which the methods will operate and the menu to which the methods are attached are owned by the CustOrd class, we'll add the methods to the App Windows module in the Order Entry application.

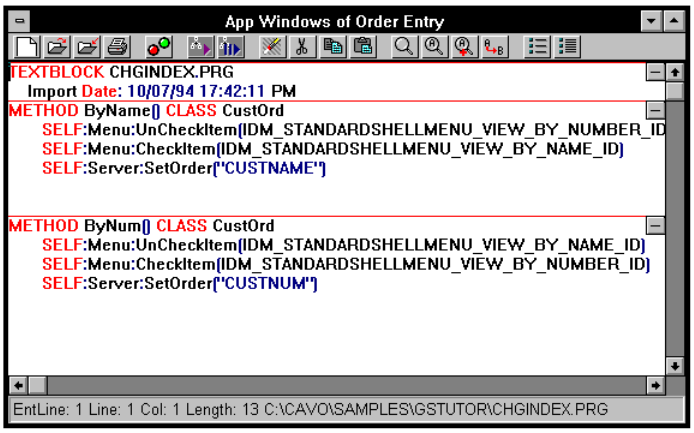
To do this:

1. Select the App Windows module in the Order Entry Module Browser.
2. Click on the New Entity toolbar button and choose Source Code Editor from the resulting pop-up menu.

   A new, blank Source Code Editor window is opened.
3. Choose the File Import command.

   This presents you with a standard Import dialog box for .PRG files.
4. Choose the file named CHGINDEX.PRG from your CA-Visual Objects \SAMPLES\GSTUTOR directory.

The code from the .PRG file is imported into the Source Code Editor:



There are three entities in this source code: a TEXTBLOCK statement and two methods belonging to CLASS CustOrd. (The CustOrd class was created for you by CA-Visual Objects when you saved the CustOrd window in the Window Editor.)

The TEXTBLOCK Entity

The TEXTBLOCK entity is automatically inserted by CA-Visual Objects every time you import a source file to denote the name, date, and time of the imported file.

The Methods

The two methods, ByName() and ByNum(), are just a few lines long:

```
METHOD ByName() CLASS CustOrd
   SELF:Menu:UnCheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NUMBER_ID)
   SELF:Menu:CheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NAME_ID)
   SELF:Server:SetOrder("CUSTNAME")

METHOD ByNum() CLASS CustOrd
   SELF:Menu:UnCheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NAME_ID)
   SELF:Menu:CheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NUMBER_ID)
   SELF:Server:SetOrder("CUSTNUM")
```

The source code for these two methods is nearly identical. Basically, the first one changes the controlling order to CUSTNAME.NTX, and the second changes it to CUSTNUM.NTX.

Let's take a line-by-line look at this code, using the ByName() method as an example, to see exactly what these methods do.

```
METHOD ByName() CLASS CustOrd
```

This statement declares the ByName() method as belonging to the CustOrd class.

```
SELF:Menu:UnCheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NUMBER_ID)
```

This method call unchecks the View By Number command in the StandardShellMenu. If you remember from reviewing the source code for both menus in this application, each option has several define constants associated with it. One of these is a unique number, that identifies the menu item. In this case, IDM_STANDARDSHELLMENU_VIEW_BY_NUMBER_ID is the unique identifier for the View By Number command.

The UnCheckItem() method is defined in the Menu class (all menus created in the Menu Editor automatically inherit from the Menu class). The method takes the unique identifier as an argument, so it knows which menu item you are referring to. Finally, "SELF:Menu" returns the menu owned by this data window, directing the UnCheckItem() call to the proper object.

Note: SELF:Menu:***MethodCall*** is the standard construction that you will use to direct a method invocation to the owned menu of any window. You will see it again later in this lesson when modifying the CustOrd:Init() method.

```
SELF:Menu:CheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NAME_ID)
```

This method call checks the By Name command on the View menu, using code almost identical to that described above.

Note: Since the ByName() method is called when the View By Name menu command is chosen, it will remove the check mark displayed next to the By Number command and place a new check mark next to the By Name command. This way, the menu will always reflect the correct controlling order.

```
SELF:Server:SetOrder("CUSTNAME")
```

This method call changes the controlling order to CUSTNAME.NTX. This is accomplished using the SetOrder() method (which is defined in the DBServer class, from which our data servers inherit). Similar to the manner in which the data window's menu was identified, "SELF:Server" identifies its data server, properly directing the SetOrder() call.

Note: SELF:Server:***MethodCall*** is the standard construction that you will use to direct a method invocation to the owned data server of any window.

Now, close the Source Code Editor, saving the imported code, and we'll move on to the final step in this lesson. What we have done so far is sufficient to add the menu commands to the StandardShellMenu and define their functionality, but since both menu commands are initially disabled, we could not use them if we stopped here.

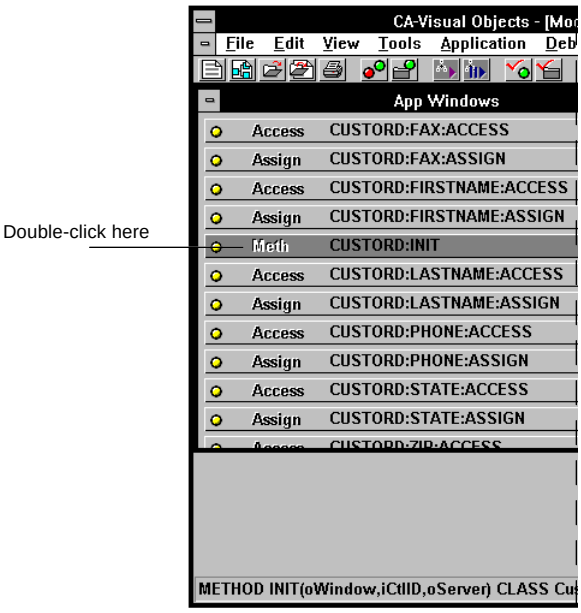
Enabling the Menu Commands

The reason behind initially disabling these menu commands was that the StandardShellMenu is shared by many different data windows, and none of them, besides CustOrd, have the associated index files necessary to properly utilize these commands. Our strategy was to disable them by default and enable them only when they apply (that is, when CustOrd is open).

The CustOrd class has an associated Init() method that is called whenever the class is instantiated, and the class is instantiated only when the CustOrd window is opened. So, this is the logical place to put the source code to enable our menu commands.

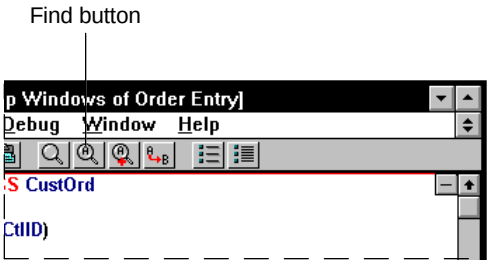
To customize the CustOrd:Init() method:

1. Double-click on the App Windows module to display its Entity Browser.
2. In the Entity Browser, locate the CustOrd:Init() method entity, and double-click on it.

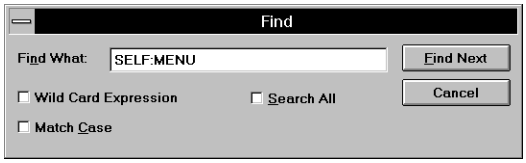


The method is loaded in the Source Code Editor—next, you will add code to enable those two menu commands.

3. Click the Find toolbar button.



4. Type **SELF:MENU** in the resulting dialog box, and choose Find Next.



   This will find the line of code that instantiates the StandardShellMenu class. It reads:

```
SELF:Menu := StandardShellMenu{SELF}
```

5. Click Cancel to close the Find dialog box.
6. Press End to move the cursor to the end of the line, then press Enter to insert a blank line below it.
7. In the new, blank line, type the following line of code:

```
SELF:Menu:EnableItem(IDM_STANDARDSHELLMENU_VIEW_BY_NAME_ID)
```

8. Press Enter again, and type the next line of code:

```
SELF:Menu:EnableItem(IDM_STANDARDSHELLMENU_VIEW_BY_NUMBER_ID)
```

   These two lines of code enable the View By Name and By Number menu commands for the CustOrd window only, similar to the way in which the ByName() and ByNum() methods checked and unchecked the menu commands.

   To save the changes you have made, double-click on the system menu and choose Yes when prompted.

Summary

With just a few lines of source code and a slight change to the StandardShellMenu, you've added quite a bit of functionality to the Order Entry application. You've learned some valuable tips, both on managing menus—by exploring some of the methods defined for the Menu class (such as EnableItem() and CheckItem())—and accessing methods defined for the data server (such as SetOrder()).

Both of these techniques were accomplished using the ownership relationships between the window and its menu and data server, and the knowledge of these relationships by the data window object through its Menu and Server properties. So, you've also learned a lot about where to put the source code to manage events (such as menu commands) and are probably feeling pretty comfortable about how all of the components in the Order Entry application fit together.

Next, you will build Order Entry and see all of your customized features in a working application, but before moving on, close the App Windows Entity Browser.