Customizing Field Properties

The next task is to define some properties for the other remaining fields.

OrderNum

Let's start with OrderNum:

1. Click on the OrderNum field in the Include list box.
2. In the Properties window, click on the FS Caption property, overtype the current contents with the text **Order #**, and press Enter.
3. Click on the FS Description property, type **Enter the order number (required)**, and press Enter.
4. Scroll down to the Required property, click on it, click on the down-arrow, and choose Yes from the list.

   Changing this property to Yes will require that the user type a value for the field in the resulting application.
5. Click on Required Diagnostic just below, type **You must enter an order number**, and press Enter.

   If the user attempts to skip the OrderNum field, this message will be displayed.
6. Scroll down to Validation, click on it, type **{ |OrderNum| OrderNum > 0}**, and press Enter.

   This will require that the user type a positive order number. The validation rule specified here is in the form of a code block. Refer to the "Code Blocks" chapter in the *Programmer's Guide, Volume III* for more information.
7. Click on Validation Diagnostic just below, type **The order number must be positive**, and press Enter.

   If the user attempts to enter a negative number or zero into the OrderNum field, this message will be displayed.
8. Click on the Save toolbar button.

OrderPrice

For the OrderPrice field, follow these steps.

1. Click on the OrderPrice field in the Include list box.
2. Click on the FS Caption property, edit the current contents so that it reads **Order Price**, and press Enter.
3. Click on FS Description, type **Enter the order price**, and press Enter.
4. Scroll down to the Picture property, click on it, type **"$$$$$$$.99"**, and press Enter.

   This will cause the value to display with leading dollar signs.
5. Optionally, you can continue to customize the remaining fields in the data server by adding status bar descriptions and/or customizing the default captions to make them more readable (for example, for the Ship_Addrs field, you might change “Ship Addrs” to “Shipping Address”).

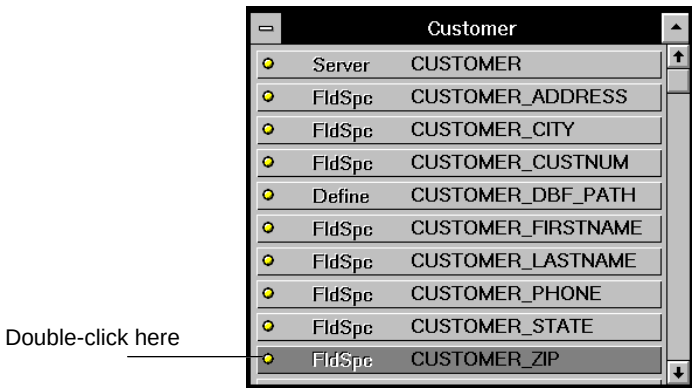
   If desired, however, you can skip this task.
6. Click on the Save toolbar button.

The FieldSpec Editor

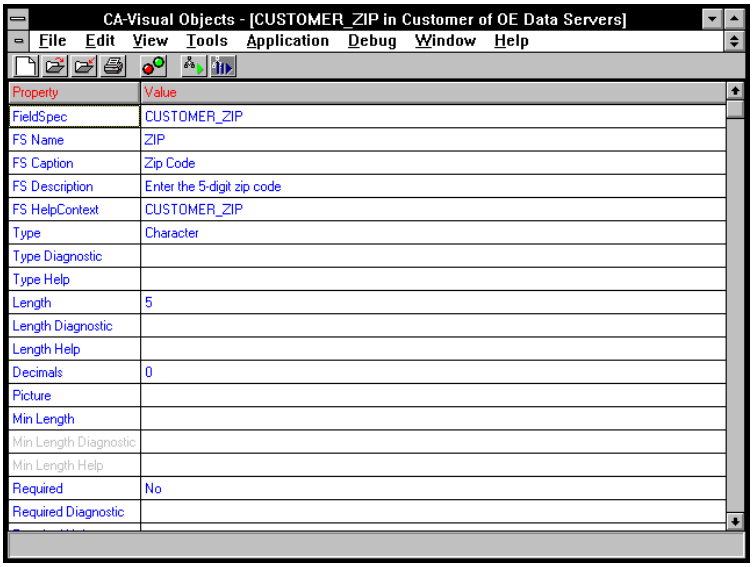
Now, you will get a chance to experience the power of using field specifications. What you’ll do is use the FieldSpec Editor to make a small change to one of the field specifications shared by the two data servers. You can then watch how the one change is automatically propagated to the two data servers using that field specification.

Before starting on this task, close the DB Server Editor by double-clicking on the system menu (save your changes if prompted).

Now, double-click on the Customer module (displaying its Entity Browser) and double-click on the Customer_Zip field specification entity:



This invokes the FieldSpec Editor, as shown below:



As you can see, the workspace of the FieldSpec Editor is remarkably similar to the Properties window of the DB Server Editor when a field is selected. The only difference is that the field-specific properties—that is, Name, Caption, Description, and HelpContext—are not present.

The FieldSpec Editor is provided primarily to let you create new field specifications, independent of a particular data server, which can then later be associated with fields in a data server. You can also use the FieldSpec Editor to modify existing field specifications. If there are any existing entities in the system already using field specifications that you modify here (for example, a data server or a data window), the changes you make in the FieldSpec Editor are automatically propagated to those entities.

Let's explore this last point in more detail. In our two data servers (Customers and Orders), the Customer_Zip field specification is used to format their individual zip code fields (if you recall, the Customer data server you imported already contained the Customer_Zip field specification, and you then associated that same field specification with the Ship_Zip field in the Orders data server).

Modifying the Field Specification

Let's make a small change to this shared field specification in the FieldSpec Editor:

1. Scroll to the Picture property and click on it.
2. Type **"99999"** and press Enter.
3. Close the FieldSpec Editor and save your changes.

Viewing the Automatic Change Propagation

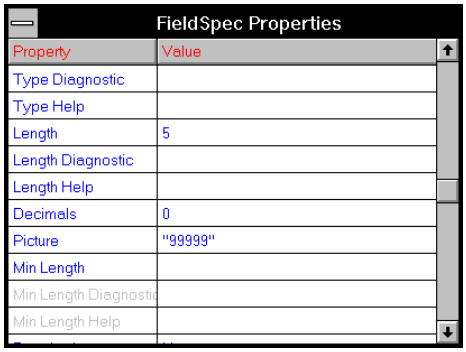
Now let's see how this one change has been propagated automatically in the two data servers that use this field specification:

1. In the Customer module's Entity Browser, double-click on the Customer server entity.

   The DB Server Editor is loaded, and the Customer data server is opened.
2. In the Include list box, click on the Zip field.

3. Click in the Properties window and scroll through it, until the Picture property comes into view.

   You will see the picture clause you just entered (which was not there before).



4. Close both the DB Server Editor and the Customer module's Entity Browser.

5. Now double-click on the Orders module and then double-click on the Orders server entity to open the Orders data server. Take a look at its Ship_Zip field's Picture property, and you will see that it has changed too.

Since both data servers pick up the Picture property from the Customer_Zip field specification entity you edited, both reflect the change. If any other data servers used this field specification, they would also reflect the change. In addition, if there were any data windows using either of these data servers, the windows would also pick up the change. CA-Visual Objects takes care of all of this automatically.

That's all there is to creating a data server. If you have not already done so, close all copies of the DB Server Editor currently in use, and return to the Entity Browser for the Orders module. From there, we'll take a closer look at the source code generated by the DB Server Editor for the Orders data server you just created.

The Source Code

As you may have been observing, each time you choose the Save toolbar button or menu command while in an editor, simply designing

pieces of an application in an editor and then saving your work automatically causes CA-Visual Objects to generate code. This code can be modified in the Source Code Editor if desired, but typically you will use the editors to make changes and then regenerate code when you save the changes.

Nevertheless, even though you probably won't be directly modifying the generated code, it may help to take a closer look at exactly what was just generated for you, so that you get a better understanding of how all of the generated source fits together.

> **Tip:** You can follow this discussion by scrolling through the Entity Browser to view the various entities or, if you like, you can double-click on the entities to view them in the Source Code Editor.

Server and Field Specification Entities

To start with, when you design a data server in the DB Server Editor, CA-Visual Objects creates a single *server* entity. There is also a *field specification* entity for each field in the data server. The purpose of this design is to let you easily edit the entire data server with the DB Server Editor or just an individual field specification in the FieldSpec Editor.

Class Entities

There are also *class* entities for the data server and each of its field specifications. If you double-click on the Orders class entity, for example, you will see that it inherits from the DBServer class, which is defined in the DBF Classes library. Notice, too, that the associated database file for this data server—ORDER_DBF_PATH—is defined as an instance variable of the Orders class as part of this inheritance.

Furthermore, each field specification class entity inherits from the FieldSpec class, also defined in the DBF Classes library.

Define Entities

There are *define* entities for the Caption and Description properties for the data server and each of its fields. This information is used throughout the generated source code (isolating it in this manner makes maintaining the source code easier).

Access/Assign Entities

Next, there are *access/assign method* entities for each field, which provide an object-oriented interface to the database fields.

Init() Method Entities

Finally, there are *Init() method* entities for the data server and each of its fields.

For the data server, the Init() method does the following:

- Sets up a file specification for the database file

  The FileSpec class helps you manage files, keeping track of information such as the name and location of the file (called the *file specification*.) (See the "File Handling" chapter in the *Programmer's Guide, Volume II* for more specific information.)

- For each field, defines a hyperlabel, instantiates the appropriate field specification class, and sets up a data field using the DataField class

  The HyperLabel class helps you keep track of certain information associated with an object (called the object's *hyperlabel*), such as a name, caption, description, and help context ID. (See the "Hyperlabels" chapter in the *Programmer's Guide, Volume II* for more specific information.)

- Sets up a file specification for each index file and opens the files in the appropriate order

For the field specifications, the Init() methods define hyperlabels for each field specification and assign values to the various properties.

Now that you have imported the Customer data server and set up the Orders data server, the OE Data Servers library is complete. There are only two steps left to complete this lesson: building the OE Data Servers library and associating it with the Order Entry application. Before continuing, close the Source Code Editor if you are using it to view the generated code, and choose No if prompted to save changes.

Building the OE Data Servers Library

Anytime you import a new library or make changes to an existing one, it is necessary to build the library to make sure that there are no errors in the source code and to make it available in compiled form to any applications that use it.

You've already gone through the process of building the Standard Program in the previous lesson—building a library is no different. Simply click on the Build button when the library has focus, and you're done.

Adding the Library to Order Entry's Search Path

After it is built, you need to add this library to the search path of the Order Entry application. In CA-Visual Objects, an application's properties include its *name*, *type* (that is, application, library, or DLL), and its associated *libraries*, among other things.

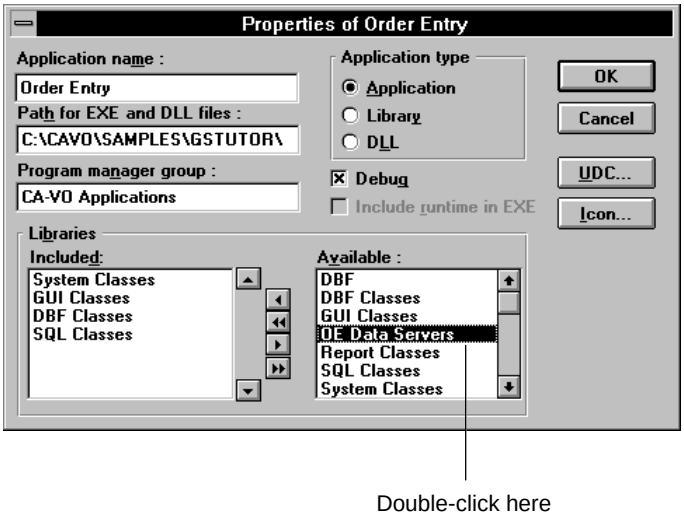
Recall that in the first lesson of this tutorial, you initially specified these items for the application when you created it. However, you can change an application's properties at any time using the Application Properties menu command.

Let's use this command now to add the OE Data Servers library to the Order Entry application's path:

1. If you haven't already done so, build the library by clicking the Build button.
2. Close the Orders Entity Browser and the OE Data Servers Module Browser.

   You are returned to the Application Browser.
3. Click the Order Entry application button to select it.
4. Choose the Application Properties menu command.

5. When the Properties dialog box is displayed, add the OE Data Servers library to your application by double-clicking on it in the Available list box:



   OE Data Servers is then added to the Included list box.

6. Choose OK.

Summary

That concludes the lesson in setting up data servers. You've taken a big step in customizing the Order Entry application, but you won't be able to see the results of what you've done until you set up a data window that uses these data servers. Therefore, let's move on to the next lesson and do just that.