Chapter 3
Working with Data Servers

Objective

This lesson introduces you to the basic concepts of data servers. You will:

- Create a Customer data server based on the traditional Xbase model of a database file (.DBF) and two SQL data servers
- Use the data servers in data windows
- Learn about server notification
- Explore programming techniques used to implement data servers

Overview

Data servers are the object-oriented means by which your applications communicate with databases. CA-Visual Objects applications can communicate with DBF databases and SQL (ODBC-compliant) databases. These database formats are very different and, in other programming languages, require drastically different approaches when writing applications.

CA-Visual Objects overcomes this difficulty through data servers. The DataServer class is the base class from which all data servers are derived. It defines a common set of methods and properties, based on a common database paradigm, which all server objects use. This means that you no longer have to code specifically for a particular data model.

The following table displays the data server methods that are designed to be compatible with the Xbase DML (data manipulation language):

Xbase (CA-Clipper)	Data Server Classes

`USE Customer ALIAS Cust NEW`
`DO WHILE CUST->(!EOF())`
`  IF Cust->Sex == "M"`
`  Cust->(DBDelete())`
`  ELSE`
`  Cust->Salary += ;`
`  Raise(Cust_Name)`
`  ENDIF`
`  Cust->(DBSkip())`
`ENDDO`
`USE`
	`oCust := Customer{}`
	`DO WHILE !oCust:EOF`
	`  IF oCust:Sex == "M"`
	`  oCust:Delete()`
	`  ELSE`
	`  oCust:Salary +=;`
	`Raise(oCust:Name)`
	`  ENDIF`
	`  oCust:Skip()`
	`ENDDO`
	`oCustServer:Close()`

Two editors, the DB Server Editor and the SQL Editor, are used to create data servers. The DB Server Editor creates a class (and associated entities) which is derived from the DBServer class. This class is used to access DBF-type databases. The SQL Editor creates a class (and associated entities) which is derived from the SQLTable class. This class is used to access SQL databases via ODBC.

The main difference between these classes is in the way you instantiate them. Also, each data server contains methods and properties specific to the type of database they serve.

For an in-depth discussion about data servers, see “Data Server Classes” in the *Programmer’s Guide, Volume II*.

Exercise

In this lesson, you will create the data servers used by the South Seas Adventures application to access customer and invoice data.

Creating a Customer Data Server

At one time, South Seas Inc. relied on a customer-tracking system written in CA-Clipper for the DOS environment. Now, they wish to incorporate the data from that system into the new South Seas Adventures application in Windows.

Invoking the DB Server Editor

You are now going to create the data server from an existing .DBF file, using the DB Server Editor.

1. Open the South Seas Adventures application by double-clicking on its button in the Application Browser:

2. To create the data server in a new module, choose the New Module toolbar button.

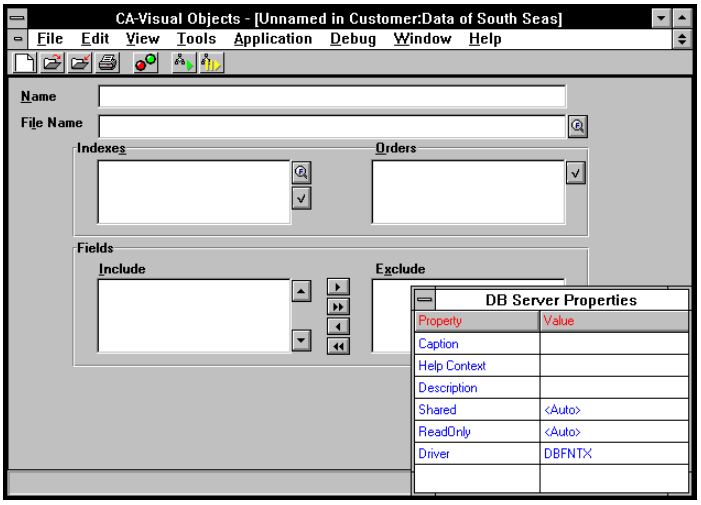
   The Create Module dialog box appears.

3. In the Enter module name edit control, type **Customer:Data** and choose OK.

   This adds the following entry to the South Seas Module Browser:

4. Select the DB Server Editor command from the Tools menu (or click on the Open Entity toolbar button and choose DB Server Editor from a local pop-up menu).

The DB Server Editor is displayed. It consists of a workspace and a floating Properties window:

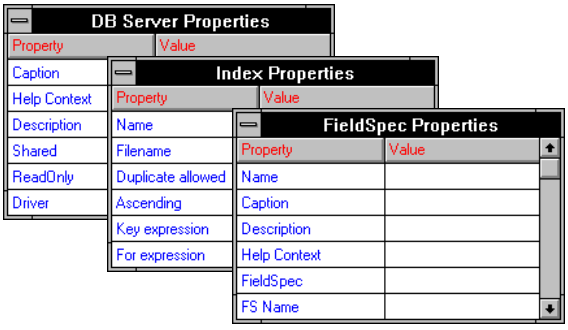


Initially, the Properties window displays properties associated with the server as a whole, as indicated by its title, “DB Server Properties.”

The Properties window changes depending on which edit control is active in the editor workspace. For example, when the active control is the Indexes list box, the Properties window displays Index properties. If the active control is a field in the Include list box, the Properties window displays field specification properties.

Each of these windows is displayed below:

Floating Properties Windows

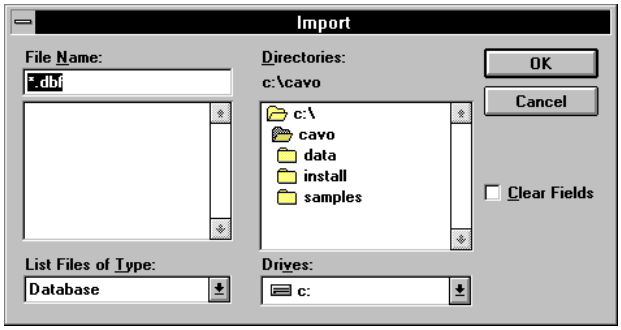


Importing a .DBF File

The DB Server Editor allows you to create a data server from scratch, or from an existing .DBF file. The CUSTOMER.DBF file already exists, so let's use the Import feature to read in its structure.

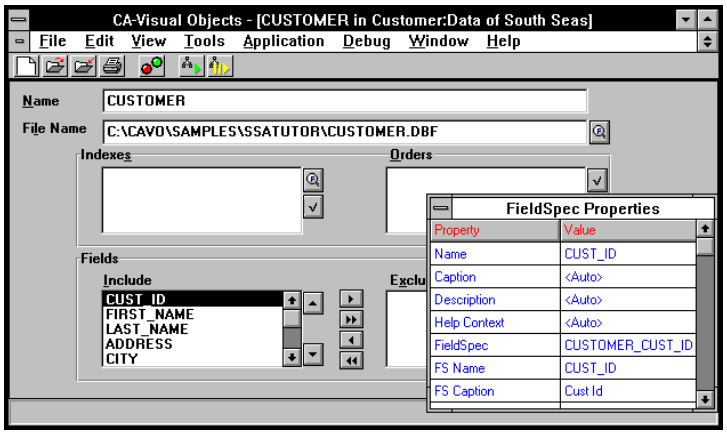
1. Choose the Import command from the File menu (or click on the Find button located to the right of the File Name edit control).

   The Import dialog box is displayed:



2. Select the CUSTOMER.DBF file located in the CA-Visual Objects SAMPLES\SSATUTOR subdirectory and choose OK.

   The Import feature fills in the Name and File Name edit controls, as well as the Include list box control in the Fields group box:



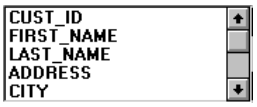
   The file name is placed in the Name edit control. This name serves as the data server name and the class name for the data server. It acts as a prefix to the classes created for each field in the database.

The physical DOS file name, including the full path, is placed in the File Name edit control.

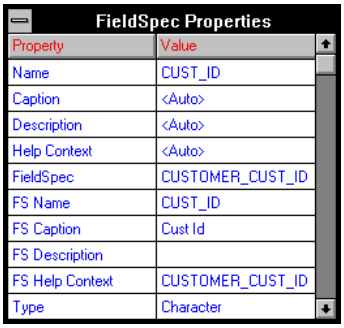
Important! *If you choose to leave the full path as is, the resulting application attempts to find the Customer database in that directory. This limits the installation options for your application (refer to Appendix A, "Creating a Path-Independent Application," for information on how to modify this path at runtime).*

3. Remove the drive and directory information from the File Name edit control, leaving only CUSTOMER.DBF.

4. The Include list box is populated with the names of the fields in the Customer database. The fields are listed in the order in which they are defined in the database. Select the CUST_ID field from the Include list box:



The Properties window now displays the properties for the Cust_ID field:



Each field of a data server has a field specification which is made up of many properties.

Tip: Scroll down through the FieldSpec Properties window to see the various properties.

The Include and Exclude list boxes allow you to control which fields are accessible through this data server. Fields in the Exclude list box are inaccessible when using this server.

Importing an Index

Now, let's select the index files for use in the South Seas Adventures application.

1. Click the Find button located to the right of the Indexes list box.

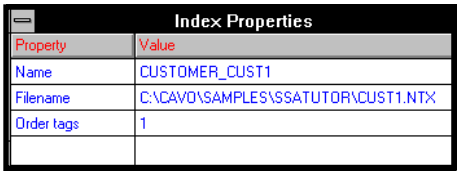
   The Browse dialog box is displayed.

2. Select the CUST1.NTX file located in the CA-Visual Objects SAMPLES\SSATUTOR directory and choose OK.

   CUST1.NTX now appears in the Indexes list box:

   The check mark indicates that this is the *controlling index*, which means that it contains the order used to control the logical order in which the database file will be processed.

   Notice that the properties window now displays the Index Properties:



   These properties specify the index file name and the number of orders within the index.

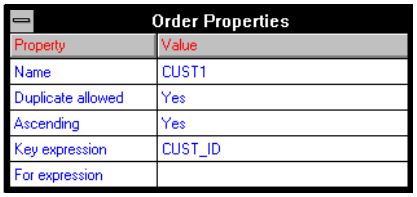
3. Click on the Filename property and remove the drive and directory information, leaving only CUST1.NTX.

   In order to support multiple index files (such as .CDX and .MDX), the index orders are displayed in the Orders list box, with the *controlling order* also indicated by a check mark:

   Since .NTX files can only support one order, no more than one order can be added to this list.

4. Click the CUST1 order.

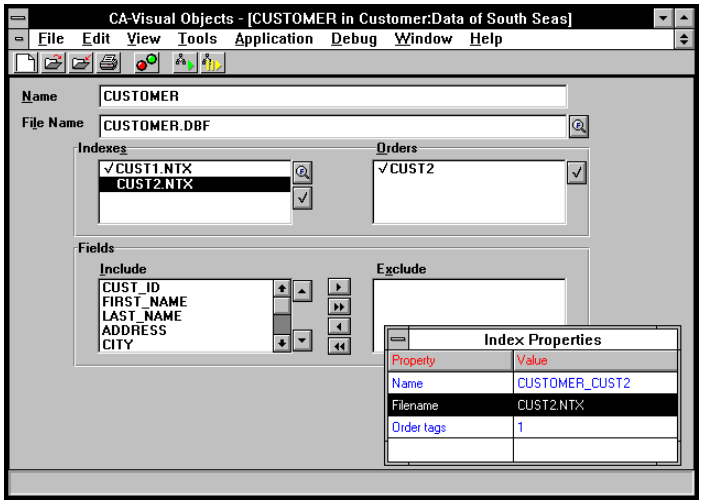
   The properties window now displays the Order Properties of the CUST1.NTX index.



   This is where you define the properties of the index order. The CUST1.NTX index will be in ascending order by the contents of the Cust_ID field.

5. Repeat steps 1 through 3 for the CUST2.NTX index file.

   When all these steps are completed, the DB Server window should look as follows:



Saving the Data Server

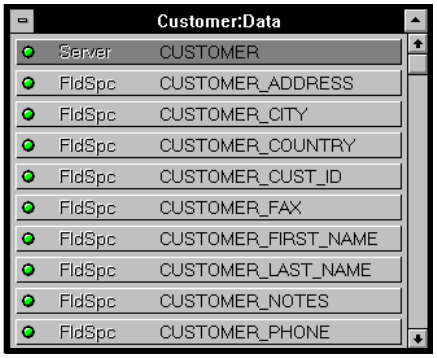
1. Save the data server by clicking on the Save toolbar button.

   CA-Visual Objects now creates the entities required for your program. Watch the DB Server Editor's status bar as it creates these entities.

2. Close the DB Server Editor by double-clicking on its system menu.

3. To see the entities created, open the Customer:Data module by double-clicking on its button.

The Entity Browser displays the entities in the Customer:Data module:



4. Scroll through the entities in the Customer:Data module to see what was created when you saved your work.

   There are five entities for each field and three entities relating to the data server. Each field has a FldSpc entity, a FieldSpec class, an Init() method, an access method, and an assign method. For the data server, there is a server entity, a DBServer class, and its Init() method.

Creating an SQL Server

The South Seas Adventures application serves as a front-end management and point-of-sale tool. The invoices and payments created using this system must be sent to the Accounting department.

For purposes of this tutorial, assume that the Accounting department of South Seas Inc. uses a separate system on an SQL database. This database is accessible via Open Database Connectivity (ODBC).

CA-Visual Objects provides ODBC drivers for the following major databases:

- CA-Ingres
- CA-DATACOM/DB
- CA-DATACOM/PC
- Watcom SQL
- Btrieve
- SQL Server (Sybase)
- SQLBase (Gupta)
- Informix
- Oracle
- NetWareSQL

Note: If you did not select all the components when you originally installed CA-Visual Objects, refer to the Installing ODBC Drivers section in this chapter.

To simulate access to the Accounting department's data, you will access .DBF files via the dBASE ODBC driver provided with CA-Visual Objects.

Tip: This is a great way to prototype applications which will eventually be connected to an SQL database. For development purposes, you can use the dBASE ODBC driver. When creating your .DBF files, make the DBF layout the same as the SQL accounting database. For the final release, simply modify the SQL server to use the new data source and recompile. Alternatively, you might choose to use a database created with Watcom SQL , which is also included with CA-Visual Objects.

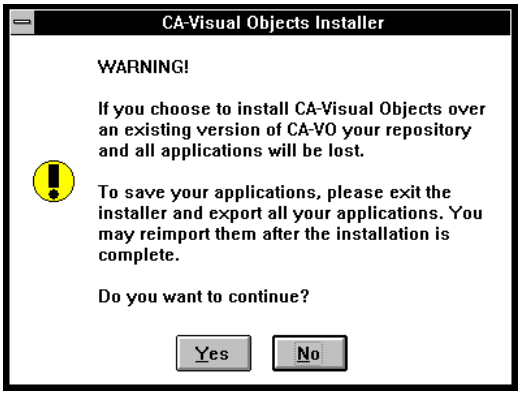
Installing ODBC Drivers

The CA-Visual Objects installation program allows you to install any number of the available ODBC drivers. For this lesson, you need to have the dBASE ODBC driver installed. If you already installed this driver, skip to the next section, The ODBC Administrator.

To install the dBASE ODBC component on your hard drive:

1. Insert the Installation diskette (disk 1) in drive A (or B).
2. From the Windows Program Manager, select the Run command from the File menu.
3. Type **A:\INSTALL** (or **B:\INSTALL**).

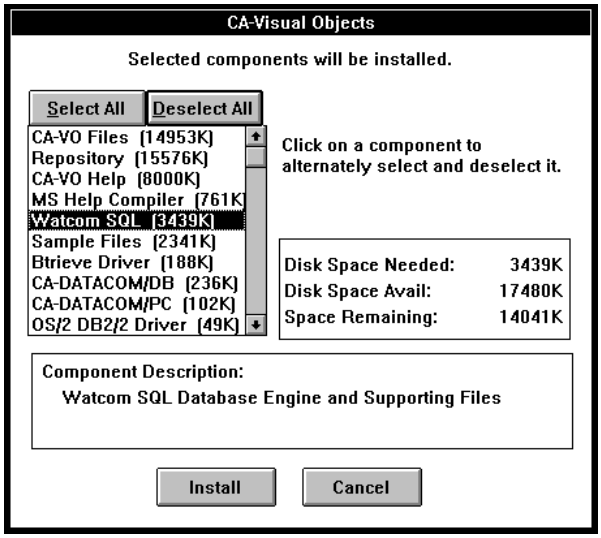
   Since you have already installed CA-Visual Objects, a warning dialog box appears:



4. Choose Yes, as you do not want to reinstall the CA-Visual Objects repository.
5. You will then be prompted to view the ReadMe file again. Choose No.
6. Next you will be asked to enter the directory path in which to install CA-Visual Objects. Enter the *same* path in which CA-Visual Objects was originally installed, and click on the Continue push button.

   This invokes the CA-Installer.

7. Click on the Deselect All push button.
8. Scroll through the available options, and click on the Watcom SQL component (the dBASE ODBC driver is installed as part of this component):



9. Click on the Install push button.
10. The CA-Installer will proceed to install the dBASE ODBC driver. Simply follow the on-screen prompts to proceed with the installation.
11. Since you do not need to modify your path, choose No in the Installation dialog box.

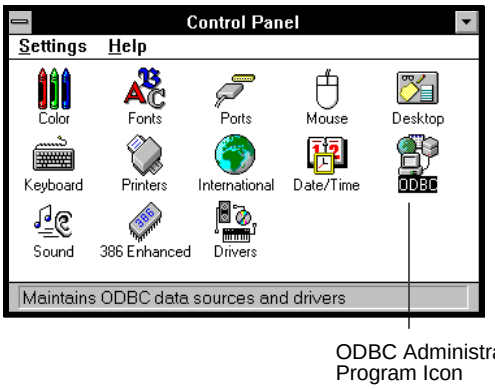
The ODBC Administrator

Before you use the SQL Editor to define a SQL table for the first time, you must define the data sources using the Windows ODBC Administrator.

The ODBC Administrator allows you to add, delete, or configure data sources. A data source is the data you want to access and the information needed to get to that data.

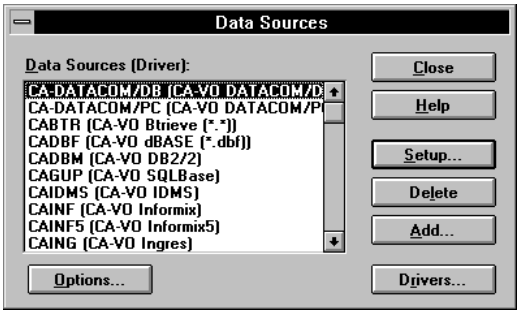
The ODBC Administrator is accessed through the Windows Control Panel, as follows:

1. Open the Windows Control Panel, which is typically found in the Main program group:



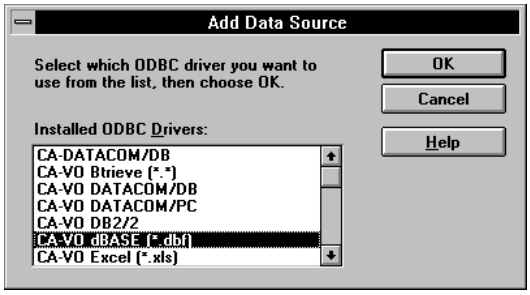
2. Double-click the ODBC program icon to launch the ODBC Administrator.

   The Data Sources dialog box appears:



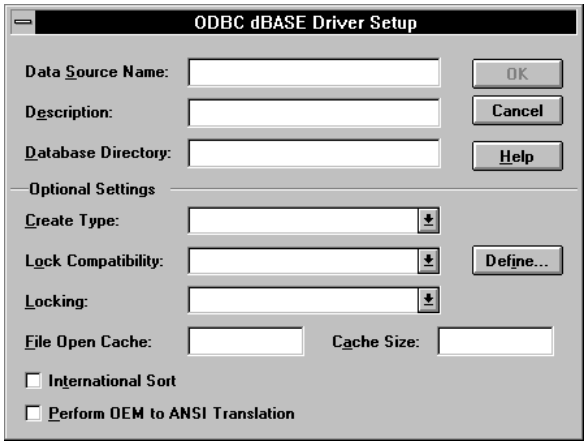
3. Add a new data source by clicking on the Add button.

   The Add Data Source dialog appears, displaying all of the drivers that you have installed:



4. Select CA-VO dBASE (*.dbf) from the Installed ODBC Drivers list box, and then choose OK.

   The ODBC dBASE Driver Setup dialog box appears:



5. In the Data Source Name edit control, type **Accounting**.

   This is the name that is searched for, when choosing a data source from inside of the SQL Editor.

6. In the Description edit control, type **Accounting Department Data**.

7. In the Database Directory edit control, type in the path to the SAMPLES\SSATUTOR subdirectory, which is located in the CA-Visual Objects installed directory (for example, C:\CAVO\SAMPLES\SSATUTOR).
8. From the Create Type drop-down list box, select Clipper.

   This ODBC driver allows access to all Xbase data.
9. Each Xbase product uses its own style of locking, therefore, choose Clipper from the Lock Compatibility drop-down list box.
10. Choose RECORD locking from the Locking drop-down list box.

    You are now finished defining the data source.
11. Choose OK.

    You should see your new entry in the Data Sources (Drivers) list box when the Data Sources dialog box reappears.
12. Choose the Close push button, and then close the Control Panel.

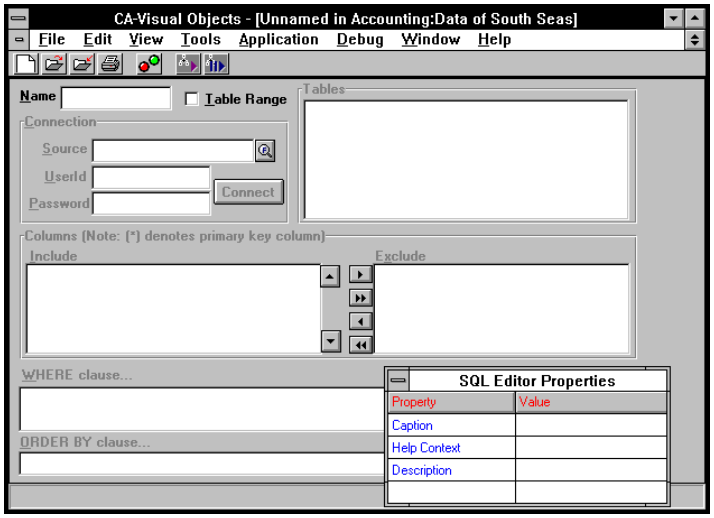
Using the SQL Editor

1. Open the South Seas Adventures application (if not already open) by double-clicking its button in the Application Browser.

2. To create your SQL server in a new module, select the New Module command from the File menu.
3. Name the new module **Accounting:Data** and choose OK.

4. Click on the Open Entity toolbar button and select SQL Editor (or select the SQL Editor command from the Tools menu).

   The SQL Editor window appears:



5. In the Name edit control, type **ACCINVC**.

   The SQL Editor uses this name to create the AccInvc class, which you are going to use later to access the data from your program.

   Now that you have a name, you may select the data source that the AccInvc server accesses.

6. Click the Find button, which is located to the right of the Source edit control.

   The SQL Data Sources dialog box is displayed.

7. Select the Accounting data source and click OK.

   The Tables list box is now populated with a list of the .DBF databases or tables that are in the directory you specified for the Accounting data source.

8. Select the ACCINVC table from the list.

   This is the destination file for the invoice information sent by the South Seas Adventures application.

   The SQL Editor fills the Include list box with the columns (fields) of your table. As with the DB Server Editor, you can choose to exclude columns from the server. Excluding columns has no affect on the actual table, they simply are not accessible by the server that excludes them.

   Field spec entities have also been created for each column. Since you are not going to use this server as part of a window, the default field specifications need no modification.

   If you require the table rows to be filtered, you can add an SQL WHERE clause in the WHERE Clause multiline edit control. You can also specify an SQL Order clause to sort the rows.

9. For the purposes of this lesson, this SQL server definition is complete. Therefore, select Save from the File menu.

10. Select the New Server toolbar button to clear the current editor.

11. Repeat steps 5 through 9 to create the AccPay server. The AccPay table also resides in the Accounting data source.

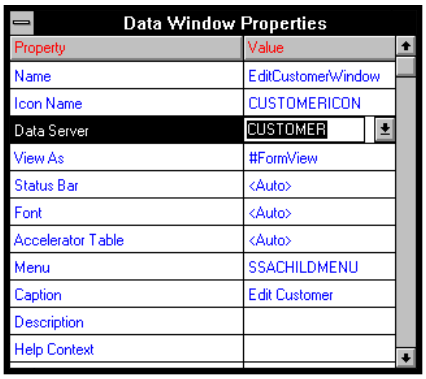
12. Once this is done, exit the SQL Editor by double-clicking on its system menu.

Attaching a Data Server to a Data Window

The data server also serves as a driver for the CA-Visual Objects DataWindow class. All derived classes of the data server, which conform to its protocol, can be used by data windows.

The Customer data server that you created is destined to be a driver for the EditCustomerWindow window, which is subclassed from DataWindow.

Attaching a data server to a DataWindow object is a simple operation. If the window you wish to attach it to was created using the Window Editor, simply set the Data Server property to the name of your server, as shown below:



Data Window Properties from the Window Editor

You can use the following code to attach a data server to a window:

```
// Instantiate a data server
oCust := Customer{}
// Instantiate the data window
oEditCustomerWindow := ;
       EditCustomerWindow{SELF,,oCust}
```

Upon instantiation, a DataWindow object can accept a data server as one of its parameters.

When a data server is attached to a data window, the data window does all the work for you. The data window automatically performs such tasks as updating your controls with data, moving the record pointer in the data server, and updating the database with data from the controls. (Refer to the "Creating and Using Windows" chapter in this guide for more information on data windows.)

Programming with Servers

You will now look at a support method that demonstrates how a data server is typically used. It actually implements the transfer of data from the South Seas Adventures application to the Accounting Department.

Importing a Support Module

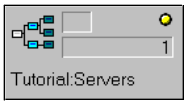
First, you must import a support module export file (.MEF), containing the predefined OptionsSubmit() method, using the following steps:

1. Select the Import command from the File menu in the Module Browser.
2. From the Import dialog box, select the TUTSERV.MEF file located in the SAMPLES\SSATUTOR\FILES subdirectory.
3. Choose OK.

   Notice that a new module called Tutorial:Servers has been added to the South Seas Adventures application.

Viewing the Server Source Code

1. Open the Tutorial:Servers module, by double-clicking on its module button.



2. Open the Source Code Editor by double-clicking on the OptionsSubmit() method of the SSAWindow class.

   Notice that the code uses the AccInvc and AccPay SQL servers in the same way that data servers based on .DBF files are used. In fact, had you not just created the servers, you would probably not know what type of server they are.

   In the OptionsSubmit() method, you will find two loops—one for processing invoices, the other for processing payments.

   The payment processing loop proceeds as follows:

   - Define and create the necessary data server objects.

     ```
     LOCAL oAccPay AS AccPay
     LOCAL oPayment AS Payment
     oAccPay := AccPay{}
     oPayment := Payment{}
     ```

   - Position the Payments data server at the first record.

     ```
     // Submit payments
     oPayment:GoTop()
     ```

   - Loop while there are records to process and the user wants to continue. This method uses a progress bar dialog box, that allows the user to cancel the process.

     ```
     DO WHILE !oPayment:EOF .AND. ;
           !oProgressDialog:CancelRequested
     ```

   - Check to see if the current payment record has already been submitted to the Accounting department.

     ```
     IF !oPayment:Submitted
     ```

- If the current record has not been submitted, add a record to the Accounting department's database and update its fields.

```
oAccPay:Append()
oAccPay:Inv_ID := oPayment:Inv_ID
oAccPay:Pay_Date := oPayment:Pay_Date
oAccPay:Tender_ID := oPayment:Tender_ID
oAccPay:Amount := oPayment:Amount
oAccPay:Details := oPayment:Details
oAccPay:Expiry := oPayment:Expiry
oAccPay:CardNo := oPayment:CardNo
```

- Force a write to the database.

```
oAccPay:Commit()
```

- Set the payment record as having been submitted.

```
oPayment:Submitted := TRUE
```

- Advance the progress bar.

```
oProgressDialog:Advance;
       ("Reviewing Payment #: "+ ;
       oPayment:Inv_ID)
```

- Go to the next record in the payment record and return to top of loop.

```
       oPayment:Skip()
ENDDO
```

- Close all files

```
oAccPay:Close()
oPayment:Close()
```

Running the Application

Now, let's see the code in action:

1. You must first build the application by clicking on the Build toolbar button.

2. Run the application by clicking on the Execute toolbar button.
3. At the Login dialog box, type **User** in the Name edit control and **Trainee** in the Password edit control, and choose OK.

4. Select the Submit Invoices and Payments command from the Options menu.

   All unsubmitted invoices and payments currently in the system will be sent, via the ODBC connection, to the Accounting Department's database. At this point, there are no invoices or payments in the database, so no data will be sent. The Submission Report dialog box informs you how many records have been submitted.

5. Close the Submission Report dialog box by clicking OK.

6. Close the South Seas Adventures application by choosing Exit from the File menu and then selecting Yes when prompted.

Event Notification

Up to this point, you have seen data servers as simple tools for programming. But they are capable of much more with minimal coding.

Client Data Windows

Data servers are aptly named. Within an application, each data server object has clients, in particular—data windows.

A data server will notify its client data windows of all operations affecting the data server. This allows the data windows to keep themselves up to date with respect to the data server (for example, updating the data display, appending records, and moving the record pointer).

To attach a data server to a window, all you need to do is perform a Use() operation on the data server.

1. Open the Employee:Forms module by double-clicking its button in the Module Browser.

2. Find the EditEmployeeWindow:Init() method and open it by double-clicking on it. The following code appears in the Source Code Editor:

```
METHOD Init(oWindow,iCtlID,oServer);
          CLASS EditEmployeeWindow
    LOCAL olServer AS OBJECT
    SUPER:Init(oWindow,ResourceID ;
          {"EditEmployeeWindow"},iCtlID)

    ...

    IF (oServer = NIL)
          SELF:Use(Employee{})
    ELSE
          SELF:Use(oServer)
    ENDIF

    ...

    SELF:ViewAs(#FormView)

    RETURN SELF
```

This method was created when the EditEmployeeWindow window was saved from within the Window Editor. This method performs a Use() on either a data server passed to the window, or the Employee data server defined for the window in the Window Editor.

SELF:Use(Employee{}) registers the window as a client of the Employee data server. When something happens, such as record-pointer movement to the EditEmployeeWindow object's attached Employee data server, the window is notified. The data window then takes the appropriate action—such as updating its controls.

What if a data server has two client data windows? Both will be notified when something happens to the data server. If data changes in one data window, the other is notified and changes are automatically displayed.

Child Servers

A data server can also be related to another data server, via a SetRelation or SetSelectiveRelation link. The server issuing the SetRelation or SetSelectiveRelation call becomes the parent, while the other becomes the child.

Any movement in the parent server automatically causes movement in the child. In addition to movement actions, the SetSelectiveRelation link limits visible records in the child server to those that match the relation as demonstrated below:

SetSelectiveRelation links are used on data windows that contain subform controls. The subform control has its own attached data server. The data server of the window is the parent, while the data server of the subform control is the child.

An example of this can be found in the Init() method of the EditAdventuresWindow class in the Adventures:Forms module. This code was generated automatically using the Master Detail option of the Auto Layout feature in the Window Editor. The EditAdventureWindow deals with two data servers. It uses them in a master-detail relationship.

In this case, the master server is related to the detail server using a SetSelectiveRelation(). Using this type of relation filters the child server so that only those records that match the relation key are visible. In the Edit AdventureWindow window, you only want to see the detail records of a particular adventure.

The EditAdventureWindow contains a subform called AdventureDetailSubform (see the Adventure:Forms module). The subform server, AdvDtl, is specified at about line 51 of its Init() method. This code was generated by the Window Editor.

The subform is created at about line 102 of the Init() method of the EditAdventureWindow. The next two lines show the subform and set the selective relationship using the #Adv_ID field. This field relates the parent server (AdvHdr) to the child server (AdvDtl).

Manual Notification

The data server classes were designed to allow you to create multiple instances of the same server without having to worry about work areas, unique aliases and SQL cursors.

When you create individual instances of the server, it is important to remember that notification will be sent only to its registered clients. If it has no clients, no notification is sent.

Consider the case where you wish to validate a key for uniqueness. You could use the following code:

```
LOCAL cCustID
cCustID := "00001"
oCust := CUSTOMER{}
IF !oCust:Seek(cCustID)
      ? "Customer is unique!"
ELSE
      ? "Customer exists!"
ENDIF
```

Using this type of code is perfectly safe and no other considerations must be made.

However, consider the case where you have a customer window on the screen, you do not have your window registered with the oCust data server and your program executes the following:

```
oCust := CUSTOMER{}
IF !oCust:Seek(cCustID)
       oCust:NAME := "NEW NAME"
ENDIF
```

Since the customer window is not a registered client of oCust, it will not be notified of the update made to its database.

If you want your customer window to be updated, you must send the notification yourself.

Broadcast Message Activation

In the South Seas Adventures application, this type of custom notification was accomplished using a notification broadcasting system from the SSAWindow class (the shell window of the application).

Here's how it works. Essentially, any code that modifies servers directly, sends a notification to the SSAWindow. The message is sent via the BroadcastMessage() method of the SSAWindow class.

Broadcasting Messages

The BroadcastMessage() method is called from several different methods, including the Notify() method for edit windows and others containing custom code for push buttons (for example, Delete, Invoice, OK, Refund, and Void). You can find these methods by using the Source Code Editor's Find dialog box, and selecting the << Advanced push button.

Open the Adventure:Methods module and double-click on the OKButton() method in the NewAdventureWindow class. Several key lines are shown below:

```
METHOD OKButton() CLASS NewAdventureWindow

       IF ValidateControls(SELF, SELF:AControls)
              SELF:Append()
              ...
              oCust := Dup_Customer{}
              oCustSeek(oDCmCustID:Value)

              SELF:Append()

              ... // Updates to Adventure servers

              SELF:Server:Commit()

              // Broadcast notification
              SELF:Owner:BroadcastMessage(SELF,;
                     #Customer)
              SELF:Owner:BroadcastMessage(SELF,;
                     #Adventure)
              SELF:EndWindow()

       ENDIF

       RETURN SELF
```

The message to be broadcast is simply the symbolic name of the affected server. The SSAWindow window, in turn, sends the notification to all its child windows that possess a ReceiveBroadcastMessage() method. The messages to be broadcast simply contain the symbolic name of the affected server.

```
METHOD BroadcastMessage(oSender,symMessage) ;
       CLASS SSAWindow

       LOCAL i AS WORD
       LOCAL oCurrentChild AS OBJECT

       FOR i := 1 TO LEN(aChildWindows)
         oCurrentChild := aChildWindows[i]

         // Do not process the sender
         IF oSender != oCurrentChild
              IF IsMethod(aChildWindows[i],;
                     #ReceiveBroadcastMessage)
                            oCurrentChild:;
                            ReceiveBroadcastMessage;
                            (symMessage)
```

```
            ENDIF
        ENDIF
    NEXT

    RETURN SELF
```

Receiving Broadcast Messages

In the South Seas Adventures application, several windows have a ReceiveBroadcastMessage() method. These include browse windows (Adventure, Employee, Item, Invoice, Payment), edit windows (Adventure, Employee, Item, and Invoice), and the NewAdventure and ViewPayment windows.

The child window that needs to receive these notifications can have its own ReceiveBroadcastMessage() method to update itself accordingly.

For example, open the Adventure:Methods module and double-click on the ReceiveBroadcastMessage() method for the AdventureBrowser window.

```
METHOD ReceiveBroadcastMessage(symMessage);
    CLASS AdventureBrowser

    IF symMessage == #Adventure
        oSFAdventureSubform:Browser:Refresh()
    ENDIF
RETURN NIL
```

In the previous example, the AdventureBrowser and the NewAdventureWindow windows each have their own instances of the Adventure data server. When a new adventure is added, the AdventureBrowser window must be notified.

Summary

You now know how to use the DB Server and SQL Server Editors to create data servers. In this lesson, you have:

- Created the Customer class (a subclass of DBServer), as well as the AccInvc and AccPay classes (subclasses of SQLTable)
- Created an ODBC data source and seen how to program using data servers
- Been introduced to the notification process that occurs between data servers and data windows

If you would like more information, refer to “Defining Data Servers and Field Specs” in the *IDE User Guide*.

In the “Creating and Using Windows” chapter of this guide, you will attach the Customer data server to the EditCustomerWindow window.