Chapter 9
Accessing and Updating Data

Objective

This lesson demonstrates how to access data from controls, data servers, and data windows. It is presented in narrative format and is designed as a reference on the subject of accessing data.

Overview

So far, you have seen that data windows, with attached data servers, provide mechanisms for updating the database and controls automatically. In this case, the programmer has nothing to code.

In many instances (for example, when creating event handlers), you will want to access data from your source code. There are many ways to access data—through methods, functions, data servers, data windows, and controls.

In this lesson, you will see how to retrieve and set data in databases and controls. You will also see that there are different ways to get at the same data.

The information in this lesson is essential for a good understanding of how data is processed in CA-Visual Objects.

Narrative

Xbase Compatibility

A subset of the CA-Visual Objects language is derived from Xbase. All of the usual commands work to access and assign your data. For example:

```
// Access the Amount field
nAmount := Invoice->Amount

// Assign 5 to the Amount field
Invoice ->Amount := 5

// Assign 5 to the Amount field
REPLACE Invoice->Amount WITH 5
```

In CA-Clipper, you also have related functions to allow you to access and assign database fields. For example:

```
SELECT (nInvoiceArea)

// Access the Amount field
nAmount := FieldGet(nAmountPosition)

// Assign 5 to the Amount field
FieldPut(nAmountPosition, 5)
```

In CA-Visual Objects, the above, as well as many new functions, allow database manipulation.

See the CA-Visual Objects online Help for more information about database functions. From the Contents topic, choose: Functions, By Category, Database.

Access and Assign Methods

Access and assign methods are special methods of classes. They are executed automatically each time you access data from or assign data to a named instance variable. The South Seas application contains about 500 access and assign methods. They are generated automatically by the IDE editors (Window, DB Server, and SQL).

You can define four types of instance variables in a CLASS declaration, including EXPORT, INSTANCE, HIDDEN, and PROTECT. All of these, except EXPORT, are not directly accessible externally (that is, outside of the class). The EXPORT instance variable is accessible outside of the class definition.

To access or assign to a nonexported instance variable from outside of the class, you must use a method. Indeed, this is the purpose of not exporting the variable.

The syntax for referencing a method is obviously different from that of referencing a variable. This violates encapsulation since users of the class must be aware of how a property of the class is implemented in order to know whether to use function style or variable style reference.

Using Protected Variables

In the following example, the Size instance variable is a PROTECT type variable. It therefore cannot be accessed or assigned from outside of the class.

```
CLASS Square
      PROTECT Size

METHOD Init(oSize) CLASS Square

      // Assign an instance variable (of any type)
      Size := Dimension{oSize}

RETURN SELF
```

Using a Method

You can also create a method (all methods are PUBLIC-visible outside of the class) to set the size, as in:

```
METHOD SetSize(oSize) CLASS Square

       IF oSize != NIL
              Size := oSize
              RETURN Size
       ELSE
              RETURN Size
       ENDIF
```

To access or assign the size of the square from outside of the class, you would then code:

```
oSquare := Square{100}

// Print the size of the square
? oSquare:SetSize()

// Change the size of the square
oSquare:SetSize(200)
```

Using Access and Assign Methods

A better way to do this would be through the use of access and assign methods. Access and assign methods allow you to reference your nonexported variable while maintaining the syntax established for instance variables. The Size access and assign methods are shown below:

```
ACCESS Size CLASS Square
RETURN Size

ASSIGN Size(uValue) CLASS Square
       Size := uValue
RETURN Size
```

From outside of the class, you can now code:

```
oSquare := Square{100}

// Print the size of the square
? oSquare:Size

// Change the size of the square
oSquare:Size := 200
```

EXPORT instance variables are accessible outside of the class definition. Why then, should you ever bother with access and assign methods?

Access and assign methods provide a layer above the implementation of the variables in question. Thus, if the implementation changes at the lower level, the program interface can remain intact. When using the variables directly, you take a risk in assuming the implementation will never change. If a change does occur, you may have to change each and every instance of the variable within your program.

For more information on access and assign methods, see "Objects, Classes, and Methods" in the *Programmer's Guide, Volume III*.

Generated Data Server Classes

When you save a data server, the DBServer Editor (or the SQL Editor) creates access and assign methods for each field in the table, as in:

```
ACCESS Amount CLASS Invoice
RETURN SELF:FieldGet(#Amount)

ASSIGN Amount(uValue) CLASS Invoice
RETURN SELF:FieldPut(#Amount, uValue)
```

You can therefore, access and assign data from the table by coding:

```
// Access Amount field
nAmount := oServer:Amount

// Assign 5 to the Amount field
oServer:Amount := 5.00
```

You can also access and assign fields of the data server by using the data server methods FieldGet() and FieldPut() directly, as the access and assign methods did above:

```
// Access Amount field
nAmount := oServer:FieldGet(#Amount)

// Assign 5 to the Amount field
oServer:FieldPut(#Amount,5)
```

Both coding styles yield identical results; however, using the access and assign methods make your code more readable.

Base DBServer and SQL Classes

When using the SQLSelect or DBServer classes directly, no access and assign methods exist for the fields in the database. You might therefore, expect an error to be generated with the following code:

```
// Create a data server
oConn := SQLConnection{"Accounting","UserID",;
       "Password"}
oServer := SQLSelect{"SELECT * FROM Invoice",oConn}
```

or

```
// Create a data server
oServer := DBServer{"Invoice"}
```

Then,

```
// Access Amount field
nAmount := oServer:Amount

// Assign 5 to the Amount field
oServer:Amount := 5.00
```

No error results because of an error interception mechanism provided by CA-Visual Objects. This mechanism is designed specifically to handle references to nonexistent instance variables. When you make a reference to an instance variable (in any class) that does not exist, a method call, either NoIVarGet() or NoIVarPut(), is made to the class in question.

The NoIVarGet() method is invoked when you attempt to retrieve a value from a nonexistent variable. The method is passed a parameter indicating the name of the variable to be retrieved. The following code:

```
? oAnyClass:NonExistantVariable
```

produces

```
? oAnyClass:NoIVarGet(#NonExistantVariable)
```

The NoIVarPut() method is invoked when you attempt to assign a value to a nonexistent variable. The method is passed through two parameters: the first indicates the name of the variable and the second indicates the value to assign. The following code:

```
oAnyClass:NonExistantVariable := 5
```

produces

```
oAnyClass:NoIVarPut(#NonExistantVariable,5)
```

The base data server classes use this feature to allow you to access the fields of the table to which they are associated. The NoIVarGet() method tests to see if a field in the table corresponds to the passed symbol name, and if so it performs a SELF:FieldGet(<*symName*>). Similarly, the NoIVarPut() method tests to see if a field in the table corresponds to the passed symbol name, and if so it performs a SELF:FieldPut(<*symName*>, <*Value*>).

Data Windows

This section applies to both DataWindow objects and subform controls. The subform control object is actually a data window and therefore, can be used in the same way as other data windows.

When you save a data window from the Window Editor, access and assign methods are created for each ***data*** control on the window that ***could be*** linked to a data server field.

```
ACCESS Amount() CLASS PaymentSubForm
RETURN SELF:FieldGet(#Amount)
```

```
ASSIGN Amount(uValue) CLASS PaymentSubForm
SELF:FieldPut(#Amount,uValue)
RETURN Amount := uValue
```

A data control is one that can be linked to a field in an attached data server, such as an edit control or check box. The data window uses name-based linkages when deciding whether to use data from a control or from the data server. That is, it attempts to establish a linkage between the name of the control and the name of a field in the data server. Consider the following:

```
// Access Amount field
nAmount := oDW:Amount

// Assign 5 to the Amount field
oDW:Amount := 5.00
```

If the Amount field exists in the data server, the FieldGet() and FieldPut() methods retrieve and set the data server field, respectively. Otherwise, they retrieve and set the window's control value.

You can also use the FieldGet() and FieldPut() methods directly, as follows:

```
// Access Amount field
nAmount := oDW:FieldGet(#Amount)

// Assign 5 to the Amount field
oDW:FieldPut(#Amount, 5)
```

Both coding styles yield identical results; however, using the access and assign methods make your code more readable.

Data Servers Attached to Data Windows

Data servers attached to data windows are accessible by using the DataWindow:Server instance variable. They can be used as described in the Data Server Classes section above. For example:

```
// Access Amount field
nAmount := oDW:Server:Amount

// Assign 5 to the Amount field
oDW:Server:Amount := 5
```

or

```
// Access Amount field
nAmount := oDW:Server:FieldGet(#Amount)

// Assign 5 to the Amount field
oDW:Server:FieldPut(#Amount, 5)
```

Controls

Controls can be placed on data windows and dialog windows. Controls hold data that the user inputs in a buffer. In many cases, you want to manipulate these values directly.

As you have seen, data windows automate the process of transferring data between an attached server and its controls, provided the names of the controls have corresponding fields in the data server.

If a control has no corresponding field in a data server, the control simply acts as a buffer and holds the value that the user inputs. The data window essentially leaves it alone. In this case, you have to write code to access the control's value if you want to use it.

Consider appending records to a database. In database applications, it is standard practice to buffer the user input prior to appending records. If the user chooses to save (for example, by clicking the OK button), then and only then, do you append the record. You would then update the new record manually.

For dialog windows, you may want to assign initial values to controls. For example, in a Print Report dialog, you may want the default destination to be the printer. In this case, the Printer radio button should be selected.

Whatever the reason, you will undoubtedly be required to directly access controls somewhere along the development trail. Earlier, it was stated that the Window Editor creates access and assign methods for each of its controls that ***could*** be linked to the connected data server (such as edit controls and check boxes). That includes those controls that are not linked to a data server field as well.

For example, examine the access and assign methods for the nAmount data control of the NewPaymentWindow window (open the Payment:Forms module). The only difference from the code shown earlier for the linked control (Amount) is that the buffered control name (mAmount) is used.

If a control on a data window does not have a corresponding field in the attached data server or if the data window does not have a data server attached, the data window can still help you access and assign the values in the control.

The CA-Visual Objects control classes allow you to manipulate window controls. When you save a window, the Window Editor generates a class for your new window. The class entity includes instance variables for each data control on the window. The instance variables for data controls are prefixed with oDC (for Data Controls), such as:

```
PROTECT  oDCFirst_Name
```

After instantiation, the instance variables are set to an object of the class that represents the control (for example, oDCFirst_Name is a SingleLineEdit object).

Each control class has a Value and TextValue access and assign method.

Value Access and Assign Methods

The Value instance variable represents the value held in the control, in whatever data type the control holds. The data type is determined by the field specification attached to a control. The field specification can be specified or assumed if it is attached to a data server field.

```
// Prints "N" for numeric
? ValType(oDCAmountControl:Value)

// Access the control
nAmount := oDCAmountControl:Value

// Assigns 5 to the control
oDCAmountControl:Value := 5
```

TextValue Access and Assign Methods

For edit controls and scroll bars, the TextValue instance variable represents the value held in the control as a string. The value is formatted according to the field specification attached to the control. The field specification can be specified or assumed if it is attached to a data server field.

```
// Prints "C" for character
? ValType(oDCAmountControl:TextValue)

// Access the control
cAmount := oDCAmountControl:TextValue

// Assigns "5" to the control
oDCAmountControl:TextValue := "5"
```

For other data controls—like check boxes, combo boxes, list boxes, and radio button groups—TextValue may contain different information than Value. See the Classes and Methods topic in CA-Visual Objects online Help for further information.

Summary

In this lesson, you learned about the various ways of accessing and updating data from within your program. You have also learned how to manipulate data using methods, functions, data servers, data windows, and controls.

Proceed to the next lesson to learn about using window event handlers—giving your application the ability to respond to an event initiated by the user (such as a push button click or an edit control modification).