Chapter 14 Debugging Your Application

Objective

This lesson shows you how to use the Error Browser and Debugger to debug your CA-Visual Objects applications. During this lesson, you will use the Error Browser to quickly fix compiler errors and the CA-Visual Objects Debugger to track down, then fix logic and runtime errors.

Error Browser Exercise

The Error Browser is the primary tool for locating compiler errors and is automatically displayed after any build, if an error or warning occurs during compilation. The Error Browser displays the error information, such as type and location of error, and provides easy access to the line in question through the Source Code Editor. By using the Error Browser, you will find that locating and fixing your errors could not be easier.

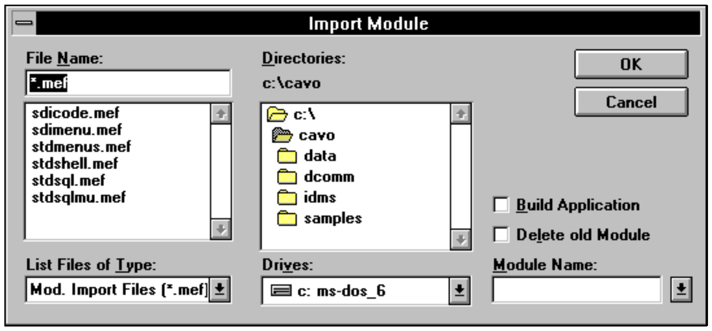
Important! *The errors which occur in this exercise involve reserved words and undeclared variables. Before beginning this exercise, choose the Compiler Options command from the Application menu and make sure the Undeclared Variables check box is* ***not*** *selected. (If it is checked, uncheck it.) To close this dialog box, choose OK.*

Importing a Module with Errors

To explore the Error Browser, let's import and compile a module with several intentional errors.

1. Open the South Seas Adventures application by double-clicking its button in the Application Browser.
2. Select the Import command from the File menu.

   The Import Module dialog box is displayed:

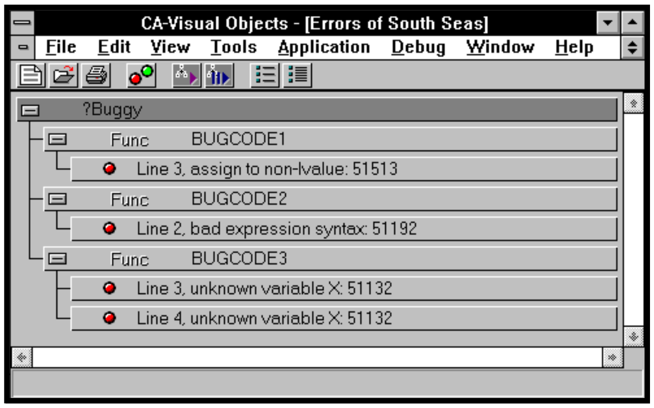
   

3. Select the file _BUGGY.MEF, which is in your CA-Visual Objects SAMPLES\SSATUTOR\FILES subdirectory, and choose OK.

   The ?Buggy module button appears as the first module in your Module Browser. Note that the question mark in the module name was replaced with an underscore when it was exported as an .MEF file, to conform to DOS file naming rules.

4. Choose the Build toolbar button to build the application.

   Since there are errors in the code, you are presented with the Error Browser. The compiler errors are displayed in a tree-like structure that can be expanded or collapsed:


CA-Visual Objects - [Errors of South Seas]
File Edit View Tools Application Debug Window Help

```
?Buggy
  Func   BUGCODE1
    Line 3, assign to non-lvalue: 51513
  Func   BUGCODE2
    Line 2, bad expression syntax: 51192
  Func   BUGCODE3
    Line 3, unknown variable X: 51132
    Line 4, unknown variable X: 51132
```

   It is similar to the application's Entity Browser, Menu Editor, and Class Browser. In this structure, the first level indicates the module, the second level indicates the entity, and the third (and final) level indicates the line number and error message.

Resolving the Errors

Now, use the Error Browser to resolve the errors in this code.

1. Double-click on the first error, which appears as follows:

```
Line 3, assign to non-lvalue: 51513
```

   This brings you to the Source Code Editor with the cursor located on the line where the error occurred:

```
LOCAL Date := Today() AS DATE
```

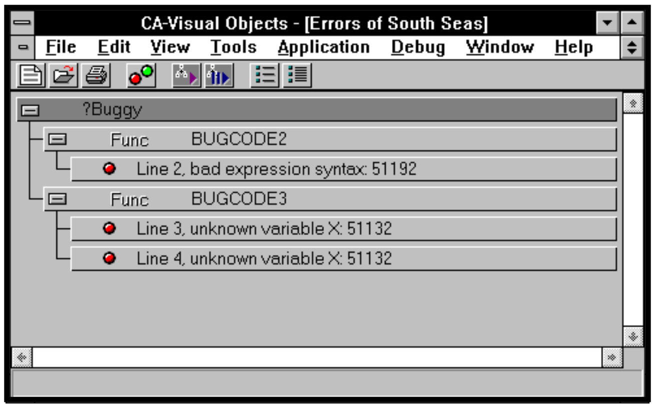
   The error occurs because DATE is a reserved word and, therefore, cannot be used as a variable.

2. Fix this code by changing the variable Date to **dDate** in the last two lines of code, as follows:

```
LOCAL dDate := Today() AS DATE
RETURN (dDate)
```

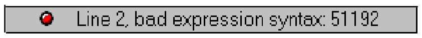
3. Close the Source Code Editor by double-clicking on its system menu and answering Yes when prompted to save your changes.

   You are returned to the Error Browser and, since you have resolved the first error, it no longer appears:


CA-Visual Objects - [Errors of South Seas]
File Edit View Tools Application Debug Window Help

```
?Buggy
  Func    BUGCODE2
    ● Line 2, bad expression syntax: 51192
  Func    BUGCODE3
    ● Line 3, unknown variable X: 51132
    ● Line 4, unknown variable X: 51132
```

4. Double-click on the next error:



● Line 2, bad expression syntax: 51192

   Again, you are brought directly into the Source Code Editor with the cursor located on the line where the error occurs:

```
nValue := nValue +
```

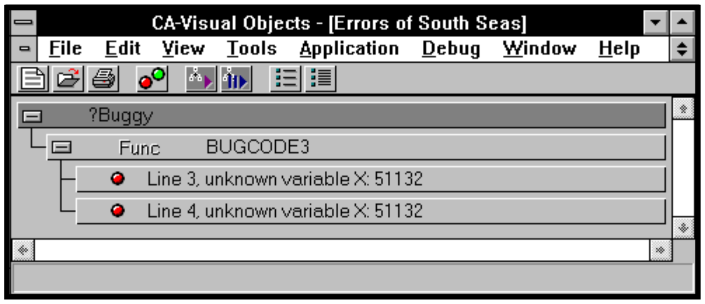
   The error message indicates that this line contains bad expression syntax and it indeed has an incomplete statement.

5. Correct the statement by adding **10** to the end of the line, as follows:

```
nValue := nValue + 10
```

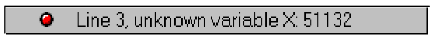
6. Close the Source Code Editor by double-clicking on its system menu and answering Yes when prompted to save your changes.

   Once again, you are in the Error Browser—now only two errors are left, both regarding the same variable:


CA-Visual Objects - [Errors of South Seas]
File Edit View Tools Application Debug Window Help

?Buggy
Func BUGCODE3
Line 3, unknown variable X: 51132
Line 4, unknown variable X: 51132

7. Double-click on the next error:



Line 3, unknown variable X: 51132

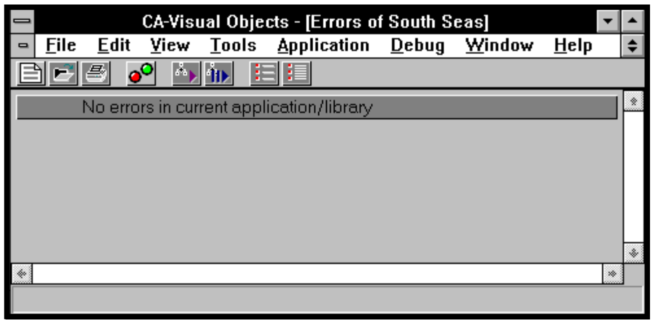
   This error message indicates that the variable *X* is unknown to the compiler, which means that it is not declared.

8. Remove the comment indicators (//) from the LOCAL statement, just above the line you are currently on, to correct the problem:

```
LOCAL X AS INT
```

9. Close the Source Code Editor by double-clicking on its system menu and answering Yes when prompted to save your changes.

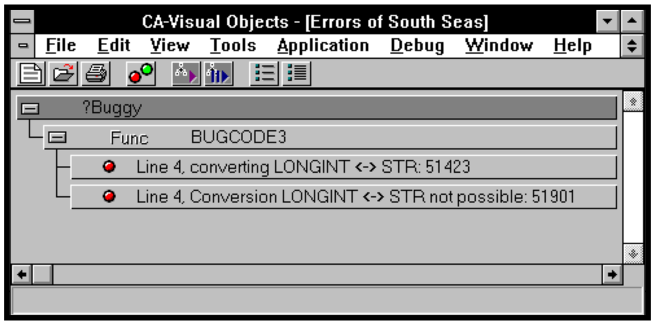
   This last correction resolved both remaining errors, so the Error Browser now states “No errors in current application/library”:


CA-Visual Objects - [Errors of South Seas]
File Edit View Tools Application Debug Window Help

No errors in current application/library

10. Choose the Build toolbar button to rebuild the application.

    The Error Browser now appears with another error:



CA-Visual Objects - [Errors of South Seas]
File Edit View Tools Application Debug Window Help
?Buggy
Func BUGCODE3

```
Line 4, converting LONGINT <-> STR: 51423
Line 4, Conversion LONGINT <-> STR not possible: 51901
```

    The previous correction causes the compiler to discover a new error. This error involves a type conversion that is not possible.

11. Double-click on the error to see the line in question.

    By examining this line you can see that the function, FUNCTION BugCode3(), returns an INT, although it was declared to return type STRING. This is a good example of why strong typing is beneficial at the development stage—an error of this nature, although very obvious at compile time, might easily elude you at runtime.

12. Fix the error by converting the return value of the function to a string, as follows:

    ```
    RETURN Str(X)
    ```

13. Close the Source Code Editor by double-clicking on its system menu and answering Yes when prompted to save your changes.

14. Choose the Build toolbar button to rebuild the application, which should now compile without errors.

15. Close the Error Browser by double-clicking on its system menu and return to the Module Browser.

Debugger Exercise

The CA-Visual Objects Debugger is the primary work space in the IDE for tracking and correcting errors that occur at runtime. To access the debugger from the various browsers and editors, use either the Debug Run menu command or the Trace Expression toolbar button.

By using the Debugger you can:

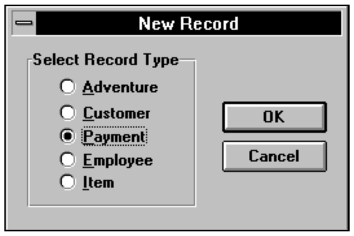
- Control the execution of your application while viewing the source code in the Debug Source Code window
- Execute any part of your application using one of several execution modes, including a mode in which you step through the code one line at a time
- Conditionally stop program execution using breakpoints or by pressing Ctrl+Alt+SysRq at any time
- Monitor watch expressions in a separate window
- Evaluate expressions on-the-fly
- View and modify variables of all storage classes
- View database, index, and other work area information in a separate window
- View and modify system settings

The most common type of errors that you will correct using the Debugger are known as *logic errors*. Logic errors occur when the application does not perform as intended, but does not crash or give any other immediate indication of flaw (such as a runtime error message). These types of errors are often the hardest to locate and the area in which the Debugger excels.

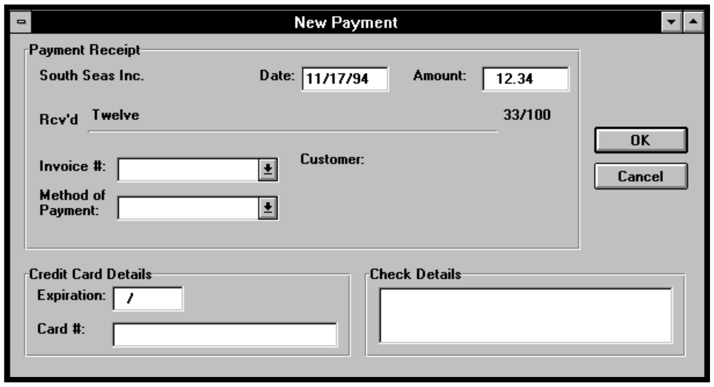
Viewing the Error

In this exercise, use the Debugger to track down a logic error that is currently in the South Seas Adventures application:

1. Provided the application was successfully built at the end of the previous exercise, you can run it now by choosing the Execute toolbar button.
2. Log in to the application as usual (Name: **User**, Password: **Trainee**).
3. Choose the New command from the File menu.
4. Select the Payment radio button from the New Record dialog box, as follows:



5. Choose OK, to display the New Payment window.
6. Enter **12.34** in the Amount edit control, as follows:



South Seas Inc. Date: 11/17/94 Amount: 12.34

Rcv'd Twelve 33/100

Expiration: /

As you enter an amount on this form, the dollars portion is printed in word format in the Rcv'd line. The cents portion is also displayed as nn/100.

Notice that the cents portion of the Rcv'd line shows 33/100 instead of 34/100. Entering several different values for the cents portion, in the Amount edit control, gives you the following results:

12.30	OK	30/100
12.31	OK	31/100
12.32	OK	32/100
12.33	OK	33/100
12.34	bad	33/100
12.35	bad	34/100
12.36	bad	35/100
12.37	bad	36/100
12.38	OK	38/100
12.39	OK	39/100

Obviously, there is something wrong with the routine used to strip out the cents portion of the Amount value. Let’s try to locate this error.

7. Close South Seas Adventures by double-clicking on its system menu and choosing Yes when prompted.

Setting Debugging On

The CA-Visual Objects Debugger can be set at three levels:

- Application level – provides debugging information for the entire application. This is done in the application properties window.
- Module level – provides debugging information for every entity in the current module.
- Entity level – provides debugging information only for the current entity.

Setting Debugging at the Module Level

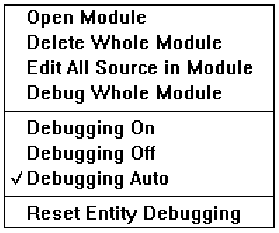
The South Seas Adventures application currently does not have debugging activated; therefore, you need to attach debugging information to the entities or modules that you feel are affected. To save you some time searching through code that you are unfamiliar with, here are a few clues:

- The routines that convert a numeric to its long string representation are in the module called App:Misc.
- The fixed text controls are assigned in the EditChange() method of the class NewPaymentWindow, which is in the Payment:Methods module.

Now, attach debugging information to the following module:

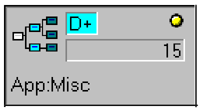
1. From the Module Browser, select the App:Misc module and click the right mouse button.

   The following local pop-up menu is displayed:

2. Check Debugging On, by clicking on it with the left mouse button.

   The App:Misc module now appears as follows:

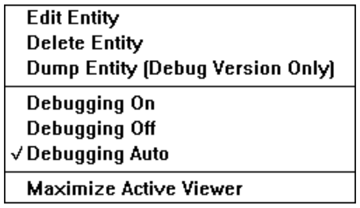
   Notice that the module button LED now shows a D+ indicator, telling you that debugging is turned on for this module. Also, the LED indicator changes from green to yellow, indicating that the application needs to be recompiled.

Setting Debugging at the Entity Level

Next, set Debugging On for the EditChange() method of the NewPaymentWindow class:

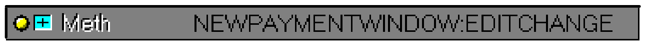
1. Locate the Payment:Forms module in the Module Browser and double-click on it to open its Entity Browser.
2. Locate the NewPaymentWindow:EditChange() method and click the right mouse button.

   The following local pop-up menu appears:

3. Check Debugging On by clicking on it with the left mouse button.

   The NewPaymentWindow:EditChange() method now appears as follows:

   Notice that the entity LED now shows a + indicator, telling you that debugging is turned on for this entity. Also, the LED indicator has changed from green to yellow, indicating that the application needs to be recompiled.

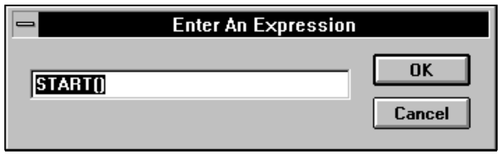
4. Choose the Build toolbar button to rebuild the application.

Running the Application Using the Debugger

Now you are ready to debug the application. First, clear some space on your Windows desktop so that you can see the Debugger while the application is running.

1. Size and position the CA-Visual Objects window so that it takes up only the top half of your screen.
2. Select the Trace Expression toolbar button.

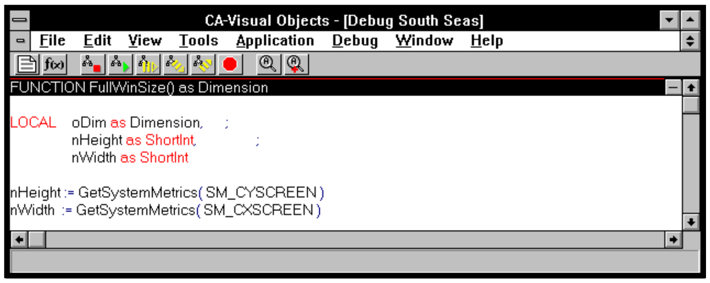
   The Enter An Expression dialog box appears:



3. Choose OK to run the application, beginning with the Start() entity.

   After choosing OK in the opening dialog box, the Debugger is invoked automatically when the application hits the first entity that has debugging set on.

   The code for the FullWinSize() function is visible in the Debug Source Code window:


CA-Visual Objects - [Debug South Seas]
File Edit View Tools Application Debug Window Help

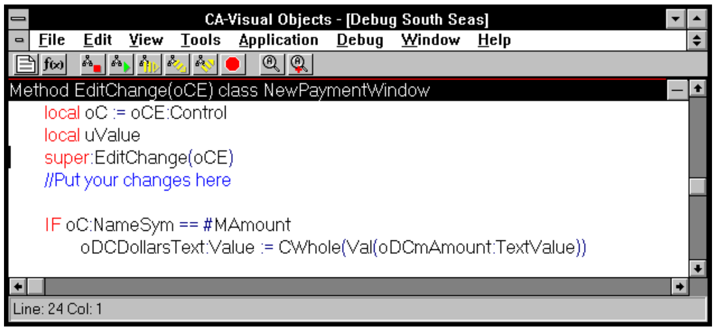
```
FUNCTION FullWinSize() as Dimension

LOCAL   oDim as Dimension,   ;
        nHeight as ShortInt,     ;
        nWidth as ShortInt

nHeight := GetSystemMetrics( SM_CYSCREEN )
nWidth  := GetSystemMetrics( SM_CXSCREEN )
```

   Any code that has debugging enabled can be seen in the Debug Source Code window.

4. To load the source code for the EditChange() method, choose Modules of South Seas from the Window menu and click on the Payment:Forms module.

5. From the Entity Browser, double-click on the NewPaymentWindow:EditChange() method.

   The code is brought into the Debug Source Code window:


CA-Visual Objects - [Debug South Seas]
File Edit View Tools Application Debug Window Help

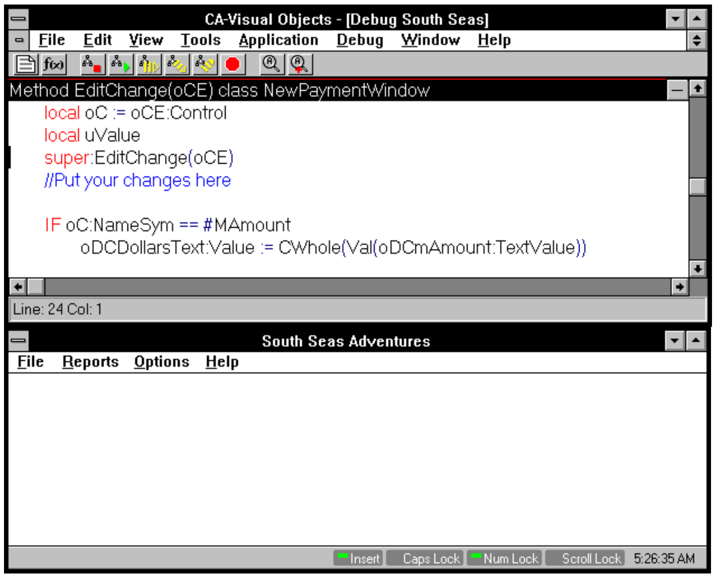
```
Method EditChange(oCE) class NewPaymentWindow
    local oC := oCE:Control
    local uValue
    super:EditChange(oCE)
    //Put your changes here

    IF oC:NameSym == #MAmount
        oDCDollarsText:Value := CWhole(Val(oDCmAmount:TextValue))
```

Line: 24 Col: 1

6. Choose the Execute toolbar button, to run the application.

7. Log in to the application as usual (Name: **User**, Password: **Trainee**).

8. Size the South Seas Adventures application window, so that it takes up the bottom half of your screen, as follows:


CA-Visual Objects - [Debug South Seas]
File Edit View Tools Application Debug Window Help
South Seas Adventures
File Reports Options Help
Insert
Caps Lock
Num Lock
Scroll Lock
5:26:35 AM

```
Method EditChange(oCE) class NewPaymentWindow
    local oC := oCE:Control
    local uValue
    super:EditChange(oCE)
    //Put your changes here

    IF oC:NameSym == #MAmount
        oDCDollarsText:Value := CWhole(Val(oDCmAmount:TextValue))
```

Line: 24 Col: 1

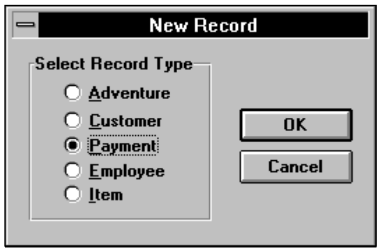
You are now ready to debug the application.

Locating the Bug

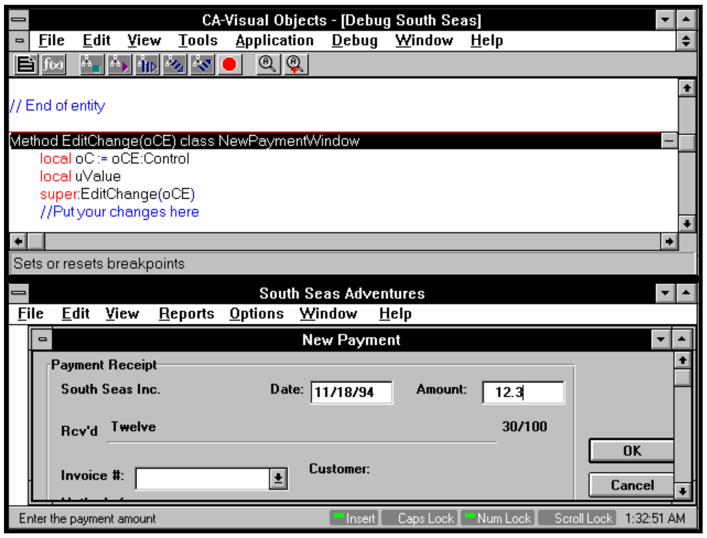
At this point, you should have the South Seas Adventures application active and positioned on the lower half of your screen and CA-Visual Objects should be positioned on the top half of your screen.

1. From the South Seas Adventures main window, select the New command from the File menu.

2. Select the Payment radio button from the New Record dialog box and choose OK:



3. In the Amount edit control , enter **12.3**:


CA-Visual Objects - [Debug South Seas]
File Edit View Tools Application Debug Window Help
South Seas Adventures
File Edit View Reports Options Window Help
New Payment
OK
Cancel
Insert
Caps Lock
Num Lock
Scroll Lock

```
// End of entity

Method EditChange(oCE) class NewPaymentWindow
	local oC := oCE:Control
	local uValue
	super:EditChange(oCE)
	//Put your changes here
```

Sets or resets breakpoints

Payment Receipt
South Seas Inc. Date: 11/18/94 Amount: 12.3
Rcv'd Twelve 30/100
Invoice #: Customer:

Enter the payment amount 1:32:51 AM

Setting a Breakpoint

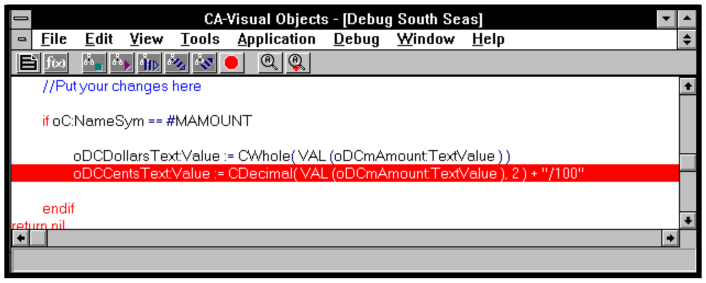
You are now ready to invoke the Debugger and to inspect the variables to determine what is happening. To help with your analysis, let's set a breakpoint in the EditChange() method:

1. Switch focus to the Debug Source Code window.
2. Select the following line of code, located in the EditChange() method:

   ```
   oDCCentsText:Value := CDecimal(...) + "/100"
   ```

3. Choose the Set/Reset Breakpoint toolbar button.

   The line of source code changes colors to indicate that a breakpoint has been set:

   
   CA-Visual Objects - [Debug South Seas]
File Edit View Tools Application Debug Window Help

   ```
       //Put your changes here

       if oC:NameSym == #MAMOUNT

           oDCDollarsText:Value := CWhole( VAL (oDCmAmount:TextValue ) )
           oDCCentsText:Value := CDecimal( VAL (oDCmAmount:TextValue ), 2 ) + "/100"

       endif
   return nil
   ```

4. Select the Amount edit control in the South Seas Adventures application window and enter **4** after the 12.3 that you entered earlier (the amount should now appear as 12.34).

   The Debugger has stopped on the line that you set as the breakpoint.

Evaluating Expressions

Now you can inspect the variable values.

1. Choose the Evaluate toolbar button.

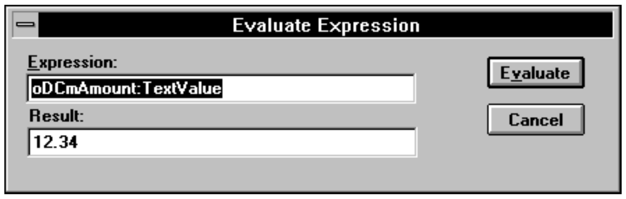
   The Evaluate Expression dialog box appears.

2. Type **oDCmAmount:TextValue** in the Expression edit control.

   This represents the character value of the Amount edit control where you are entering data.

3. Choose Evaluate.

   The evaluation produces the correct result of 12.34 in the Result edit control:



Expression: oDCmAmount:TextValue
Result: 12.34

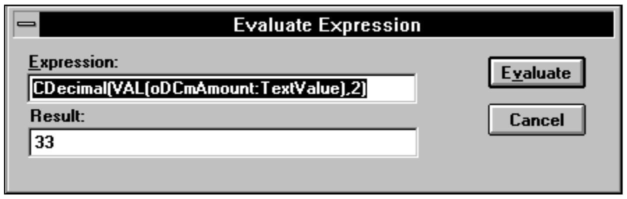
4. The error must occur deeper in the code, therefore, type the following in the Expression edit control:

   `CDecimal(VAL(oDCmAmount:TextValue),2)`

   This represents the return value of the call to the CDecimal() function.

5. Choose Evaluate.

   The Result edit control now produces the incorrect result of 33 instead of 34:



Expression: CDecimal(VAL(oDCmAmount:TextValue),2)
Result: 33

You have now determined that the problem can be traced to the CDecimal() function.

6. Choose Cancel to close the Evaluate Expression dialog box.

Tracing into a Module

Let's load the CDecimal() function and inspect it.

1. Choose the Trace Entity toolbar button to enter into the entity on the line that is currently executing.

   The CDecimal() function is loaded into the Debug Source Code window under the EditChange() method. If you scroll up, you can see that there are now three entities in the Debug Source Code window.

   After analyzing this function, you see that it has only one line of code:

   ```
   nCents := (10^nDec)*(nAmount - Integer(nAmount))
   ```

2. To start the function call, click on the Execute Next Line toolbar button (or select the Step command from the Debug menu).

   Let's look at all the local variables now.

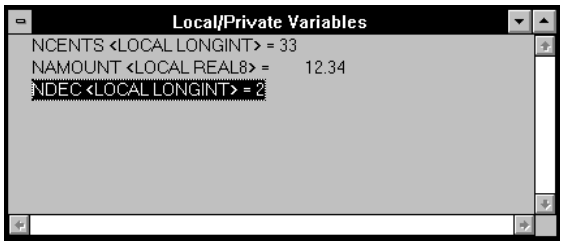
3. Select the View Locals command from the Debug menu.

   The Local/Private Variables window displays, indicating:

   ```
   NCENTS = 0
   NAMOUNT = 12.34
   NDEC = 2
   ```

4. Select the Step command from the Debug menu to execute the line that calculates nCents.

   You can see that the value of nCents is 33, which is incorrect:



```
NCENTS <LOCAL LONGINT> = 33
NAMOUNT <LOCAL REAL8> =       12.34
NDEC <LOCAL LONGINT> = 2
```

The line is executed as follows:

```
nCents := (10^2)*(12.34 - Integer(12.34))
   := (100) *(12.34 - 12)
   := (100) *(.339999999...)
   := 33.99999999
nCents -> 33 since nCents is type INT
```

The error occurs because of the way some decimal numbers are stored in binary format. Without presenting a discussion on numerical analysis for computers, suffice to say that adding **.5** to the end of the expression corrects the problem. Change the line to:

`nCents := (10^nDec)*(nAmount-Integer(nAmount))+.5`

5. To verify this expression, choose the Evaluate Expression command from the Debug menu.

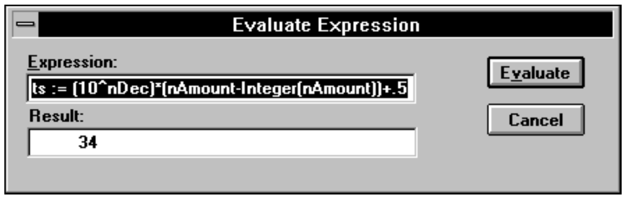
   The Evaluate Expression dialog box appears.

6. Enter the corrected expression as follows:

   `nCents:=(10^nDec)*(nAmount-Integer(nAmount))+.5`

7. Choose Evaluate.

   The Result edit control now produces the correct result of 34:



Expression:
ts := (10^nDec)*(nAmount-Integer(nAmount))+.5
Result:
34

8. Choose Cancel to close the dialog box.

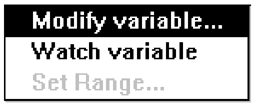
Modifying Variables

The test shows that the new expression works for *nAmount* equal to 12.34. However, from the earlier testing, we know that the program fails for other values. To check out the validity of the new expression, for other nAmount values, use the Local/Private Variables window to change nAmount to a couple of different values and test the results each time.

Perform the following steps to validate the new expression:

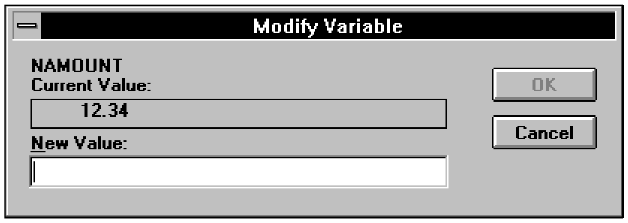
1. Highlight the nAmount variable and click the right mouse button.

   The following local pop-up menu appears:

2. Choose Modify variable by clicking on it with the left mouse button.

   The Modify Variable dialog box appears:

3. Enter **12.35** and choose OK.

   The Local/Private Variables window is updated with the new amount.

4. Choose the Expressions command from the Debug menu.

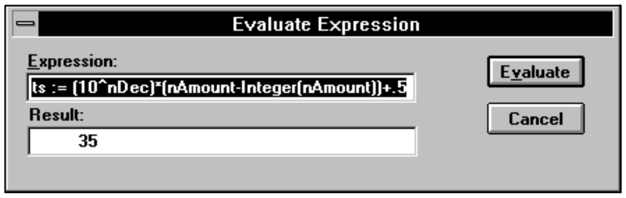
   The Evaluate Expression dialog box appears.

5. Enter the corrected expression, in the Expression edit control, as follows:

   ```
   nCents:=(10^nDec)*(nAmount-Integer(nAmount))+.5
   ```

6. Choose Evaluate.

The Result edit control now displays 35, which is correct:



Expression: ts := (10^nDec)*(nAmount-Integer(nAmount))+.5

Result: 35

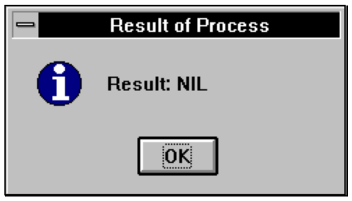
7. If you like, you may repeat steps 1 through 6, using an nAmount value of **12.36**.
8. When you are finished testing, and are confident that the new expression is valid, choose Cancel to close the Evaluate Expression dialog box.
9. Close the Local/Private Variables window by double-clicking on its system menu.

Correcting the Error

Now that you have found the bug, you can close down the South Seas Adventures application and the Debugger and fix the source code for the CDecimal() function.

1. Choose the Execute toolbar button to return to the application.
2. Double-click on its system menu to close the South Seas Adventures application and choose OK to confirm.

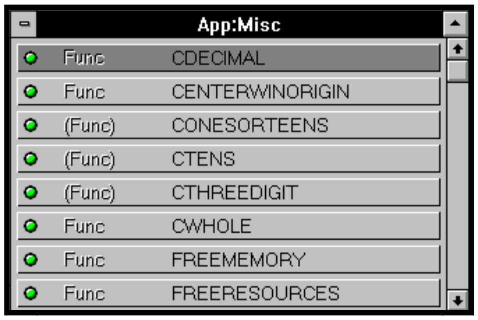
   The Result of Process dialog box, which indicates that the debugger is about to be closed, appears:



Result: NIL

3. Choose OK to close the Debugger and return to the Payment:Forms Entity Browser.
4. Close this Entity Browser by double-clicking on its system menu.
5. Maximize your CA-Visual Objects desktop.

6. Open the App:Misc module by double-clicking on its module button.
7. To make the correction, double-click on the CDecimal() entity to load it into the Source Code Editor.

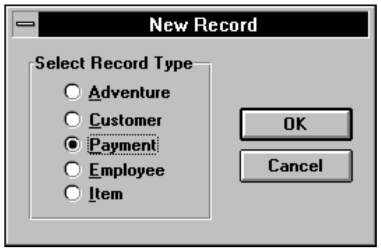


8. Correct the line of code so that it reads:

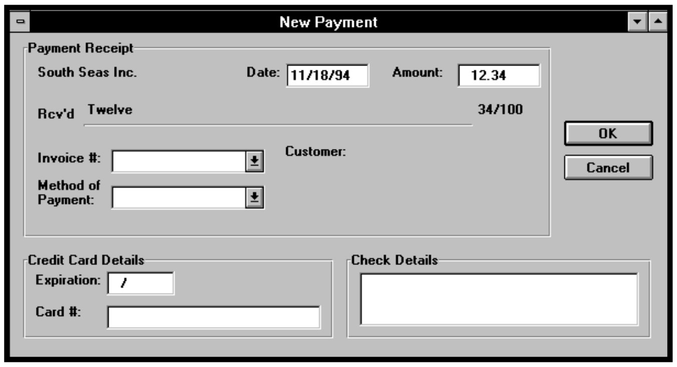
```
nCents:=(10^nDec)*(nAmount-Integer(nAmount))+.5
```

9. Close the Source Code Editor by double-clicking on its system menu and choosing Yes when prompted to save the changes.
10. Rebuild the application using the Build toolbar button.
11. Run the application to test the new results using the Execute toolbar button.
12. Log in to the application as usual (Name: **User**, Password: **Trainee**).
13. Select the New command from File menu.

    The New Record dialog box appears:



14. Select the Payment radio button, then choose OK.
15. Enter **12.34** in the Amount edit control of the New Payment window, as follows:



South Seas Inc.

Date: 11/18/94

Amount: 12.34

Rcv'd Twelve 34/100

Invoice #:

Customer:

Method of Payment:

Expiration: /

Card #:

Notice that the cents portion of the Rcv'd line is now correct.

16. If you like, you can experiment with other amount values.
17. When you are finished, close the South Seas Adventures application by double-clicking on its system menu and choosing Yes when prompted.

Summary

This lesson provided you with an understanding of some of the key features of the error detection and debugging facilities provided in CA-Visual Objects. You used the Error Browser to diagnose and fix compiler errors and the Debugger to resolve a runtime error. For additional information, see "Debugging Your Applications" in the *IDE User Guide*.

The next lesson shows how to implement an online help system for your CA-Visual Objects applications.