Debugging ABL Code

Suppose you have a distributed application that contains client-side code, shared code used by both the client and server, and server-side code. You can use the debugger to step through all parts of the code to determine why the application is not executing correctly.

[proenv: prodebugenable -enable-all]
To use the debugger, you must first enable OpenEdge for debugging. This is something that you only do once. Here we open a proenv window as administrator and execute the prodebugenable utility. Now our development environment supports debugging.

[TestClient.cls line 215]
You debug an application by setting one or more breakpoints at selected locations in your code. You can easily set a breakpoint at a specific line by double-clicking the line number for the statement where you want the debugger to pause.

When you see this blue dot to the left of an ABL statement, it means that a breakpoint has been set and is enabled.

Now, we start the application by running the RunClient procedure in debug mode. When you run a procedure in debug mode, all procedures it calls and the class instances it accesses, will run in the debugger, provided a breakpoint has been set.

Here we see that the application executes code before the breakpoint we have set. We enter values to retrieve data for customers 1 through 5.

The debugger has recognized our breakpoint. Here we see that Developer Studio requests our permission to open the Debug perspective. We specify that we want Developer Studio to remember this decision and the Debug perspective opens.

The Debug perspective looks a little different from the OpenEdge Editor perspective. It contains views that are useful for your debug session.

There are three main areas you will use for debugging. The first is the editor area where you see the ABL code you are stepping through.

The second area is the program execution area that shows the processes running in the debugger and their respective stack information.

The third area is the program execution monitoring area where you can examine values of variables, properties, and a number of ABL objects.

When you click on a particular stack location in the program execution area, you can view its current runtime values in the variables tab.

Notice that in the editor, the current line where the debugger has paused is highlighted in green.

When the debugger is paused, you can examine data in the variables tab. Here we scroll down and see the various types of variables and properties and their values.

CustomerBE is a class instance. We can drill down into this instance and view its values. For example the Adapter property of this class is an instance of the GenericServiceAdapter class.

The service interface procedure handle represents an ABL procedure. If you do not want to drill down into an object, you can look at the object details in a separate window.

You can also view information about temp-table buffers and database buffers.

The implicit temp-tables here represent the default buffers for these temp-tables.

In the variables and variables details panes, you can also modify runtime values. Here we change the range of customers to retrieve to 1 through 10.

Now let's step through the code. First we set another breakpoint so that we can examine the method's values before it returns.

When stepping through code, you can step into the procedures or methods that are called. Here, we click the **step into** icon and enter the GetData method of the CustomerBE class instance. Notice that the stack information has been added to the program execution pane.

We click the **step over** icon to step over the empty dataset statement.

The next statement runs the procedure in the application server. We cannot step into code running in the application server because we have not yet set it up for debugging so we step over it.

When you are in a procedure or method, you can click the return icon to return from the current stack location.

Here we are back in the TestClient class instance method for handling the GetCustomersButton click.

If we click the **resume** icon, the debugger pauses at the next breakpoint.

In the breakpoints tab, we see that we have only these 2 breakpoints defined. From this view, you can delete breakpoints or disable them.

If you disable a breakpoint, the blue dot is replaced by an open dot. The debugger only pauses at enabled breakpoints.

Let's resume execution of the debugger.

Notice that the client has retrieved data for customers 1 through 10, as we modified this range of values in the debugger.

We specify retrieval of customer data again. We are back in the debug perspective at our first breakpoint.

When you are done with your debug session, you end it by clicking the terminate icon. Notice that the stack trace for the client debug session is automatically removed from the program execution view.

In the breakpoints tab, you can manage breakpoints. You can define a breakpoint that will stop the process when an error occurs. We create this type of breakpoint specifying that we want to break on any error.

If we stop the application server, an error will be raised when the client attempts to connect. We stop the application server.

Now we run the client in the debugger. A window appears telling us that an error has occurred.

We accept the warning and the debugger is paused at the line of code after the error was raised. This type of breakpoint is useful when you are trying to ensure that all possible error conditions are being handled.

We terminate the debug session.

So far, you have seen how to run the debugger in Developer Studio. If you want to run the debugger that attaches to code running in the application server, you need to configure the application server for debugging. To configure the application server for debugging, you must open an OpenEdge Explorer window.

Here we navigate to and select the configuration area for asbroker1. Here we can modify this application server's configuration for debugging.

We click Edit to make the configuration changes.

For the broker, we specify in the advanced features area that debugging is enabled through the broker port.

Then in the agent tab, we specify in the advanced features area that the 4GL debugger is enabled.

We save this configuration. You only need to make this change to the application server's configuration once.

To debug code running in the application server, you set at least one breakpoint in server-side code and then you start the application server in debug mode.
You have now seen how set up your Developer Studio environment for debugging and how to step through code in the debugger. Then you saw how to set up the classic AppServer for debugging and start it in debug mode.

To learn about developing ABL applications, take the course, Developing a Progress OpenEdge ABL Application.