developmentor

Visual Studio 2010

Estimated time for completion: 45 minutes

Overview:
In this lab you will investigate Code Contracts and Intellitrace

Goals:
- Learn how Code Contracts can aid building libraries
- Investigate historical debugging with Intellitrace

Lab Notes:
N/A

Part 1: Code Contracts

In the first part of the lab take an existing library use Code Contracts to annotate the methods specify the requirements of the library

1. You will be working on the YieldCurveInterpolation library. Turn on Code Contract runtime and static checking in the Code Contracts section of the project settings. Check all of the checkboxes other than `Only Public Surface Contracts`
2. Compile the project and you will notice that you have compiler information messages suggesting contract requires options. Add the suggested options and recompile
3. If you missed one or two you will receive more suggestions (you may have to wait briefly as these checks are performed asynchronously. Add any outstanding and again recompile
4. Eventually you will get to the point of no more suggestions but only warnings about assumptions that are not verifiable. To fix these you will need a combination of new Requires settings and also Asserts. Some places the static analysis will not be able to work out that a condition has been met in the code. Add a `Contract.Assert` statement for these (such as ones that state that an index must be >= 0)
5. You will see that you get a warning about not being able to prove `sortedSource` is not `null` in the `TwoItemInterpolation` base class. Add this assertion in and recompile. You will notice that you now have an error that sortedSource is less visible than the method. The idea here is that there is no way for a caller to satisfy this directly as it cannot touch the member. This is really a shortcoming of the static analysis that is has not recognized that the constructor has already ensured this condition is met. To work around this an a public readonly property to return the `sortedList`. This obviously breaks encapsulation and shows some of the fragility of the current implementation
6. Fix up the code to use the new public property and recompile.

7. Continue this iteration trying to clear all the warnings. You can eventually get to a small number if warnings but issues with the current static analysis will prevent all of them being eliminated
8. Try calling the library in a number of invalid ways and see if the error gets picked up at compile time or runtime

---

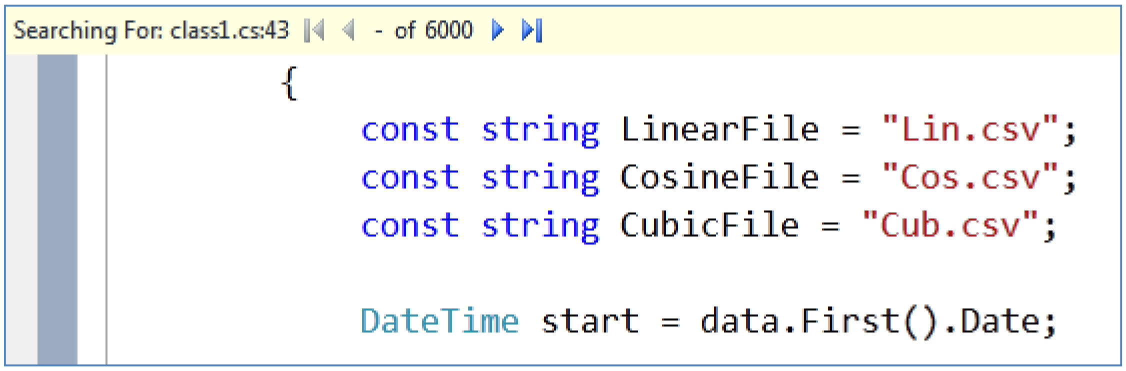
Part 2: Experimenting with Intellitrace

In the second part of the lab you will debug an application using Intellitrace. You will run the trace from the command line and then debug the generated file inside Visual Studio. Be aware that this is beta software and parts of this exercise will be slow

1. Open a Visual Studio Command Prompt and go to the directory for the VizualizerTest debug project **before\VizualizerTest\Bin\Debug**
2. Launch the application under the control of the **Intellitrace.exe** Intellitrace executable. Set the log file to **dbgtest.itrace**. Depending how you have your path set up you may need to fully qualify the path to **Intellitrace.exe** (**C:\Program Files\Microsoft Visual Studio 10.0\Team Tools\TraceDebugger Tools** by default). Use the collection plan from **CollectionPlan.xml** in the same directory as the executable being debugged

```
"c:\Program Files\Microsoft Visual Studio 10.0\Team Tools\TraceDebugger
   Tools\Intellitrace.exe" launch /f:dbgtest.itrace /cp:CollectionPlan.xml
   VisualizerTest.exe
```

3. When the process has exited (there is an exception), open the generated **dbgtest.itrace** file in Visual Studio
4. Double click on the first thread shown. This should take you to the last line of program.cs in the VizualizerTest project
5. Press Ctrl-Shift-F11 to step back to the call to visualize.Visualize then press F11 to step into this method
6. Press F10 to step through the method a couple of times. You will notice that this will be a very impractical mechanism for stepping through the loop which has 3000 iterations
7. Right click on the call to `GetRate` on the `CubicInterpolation` and select *Search for this line in Intellitrace*. You will notice a counter incrementing in the top left hand croner of the code window



```
{
    const string LinearFile = "Lin.csv";
    const string CosineFile = "Cos.csv";
    const string CubicFile = "Cub.csv";

    DateTime start = data.First().Date;
```

8. Click on the right-most button next to the counter to take you to the last instance of this line being hit. This will take a short time.
9. It would also be useful to be able to navigate more freely around the trace file. In the Intellitrace window (tabbed with the Solution Explorer by default) select *Switch to Calls View*
10. Scroll down the calls and click on the last call to `ShowData`. Press F11 to debug into this call
11. Hover the mouse over the `file` parameter and notice this doesn't match the value of the `fileroot` that is passed to the `CubicInterpolation`. This is the root of the problem: the data has been saved in a file with a different name to that which we are trying to open.
12. Press Shift-F5 to stop debugging. Fix the instantiation of the `CubicInterpolation` to take the correct file path and run the application. You should see all three graphs rendered correctly in Excel.

Solutions

after\vs2010.sln