Visual COBOL for Visual Studio Getting Started with Solutions, Projects and the COBOL CALL Statement

I) Overview

1. Solutions and Projects

In Visual COBOL for Visual Studio, the main unit of work is called a solution. Solutions can contain multiple projects. These projects can be managed code COBOL projects or native code COBOL projects or can be C# projects or VB.NET projects, etc. Visual COBOL projects can contain only COBOL programs or classes but these programs and classes can interact with the programs or classes contained within projects written in a different language like C#.

There are two basic types of projects, Application projects and Library projects. Normally, a solution would contain a main Application project like a Windows Forms Application, WPF Application or a Console Application. Application projects generate an output file with the .EXE extension and contain the main entry point of an application. Library projects, like a Class Library or a Link Library typically contain programs and classes that are called by the main application project. Library projects generate an output file with the .DLL extension.

Each project can contain one or more source programs or class programs. In managed code, each project is compiled into a single output file called an assembly. In native code COBOL Application and Library projects you can also select to have multiple output files. In this case, each individual program within the project will be compiled into its own .EXE or .DLL.

2. Problems with Calling Programs Located in Different Projects

Each project specifies an output folder into which its generated output files will be stored. The default name of this folder varies depending on the project CPU settings and which build type you are using such as DEBUG or RELEASE. The default location is in a subfolder which is relative to the projects main folder, i.e., .\bin\x86\debug. This default name of the output folder is configurable under the COBOL tab of the Project Properties page.

There are two issues that need to be addressed when a program in one project calls a program in another project.
1. Programs that are called cannot be found.
When an application is started in Visual Studio the output folder in which the main application resides will become the current folder. Programs that are called must either be placed in this startup folder or all programs must be placed in a different folder or they must reside in a folder that is locatable via environment variable PATH.
2. Entry points that are called that are different from the name of the .DLL cannot be found.
When the name of the program in the call statement matches the name of the .DLL on disk then it will be found as long as the conditions in 1 above are true. But if calling an entry point which is the name of another program within the .DLL or the name of an entry point specified in an ENTRY statement within a program in the .DLL, the .DLL containing the program to be called must be preloaded in order to make its entry points visible to the run-time system. This can be done using one of the following methods.

- set proc-pointer to entry “dllname”
- Micro Focus Entry Name Mapper (MFENTMAP)
- Interop Preload section of app.config file (managed code only)

All of these scenarios will be covered in the tutorials that follow.

II) Working with Native Code Calling Managed Code (COM Callable Wrapper CCW)

The .NET Framework provides for several techniques and classes that allow managed code to interact with native code and vice versa. These techniques and classes are known as Interop classes.

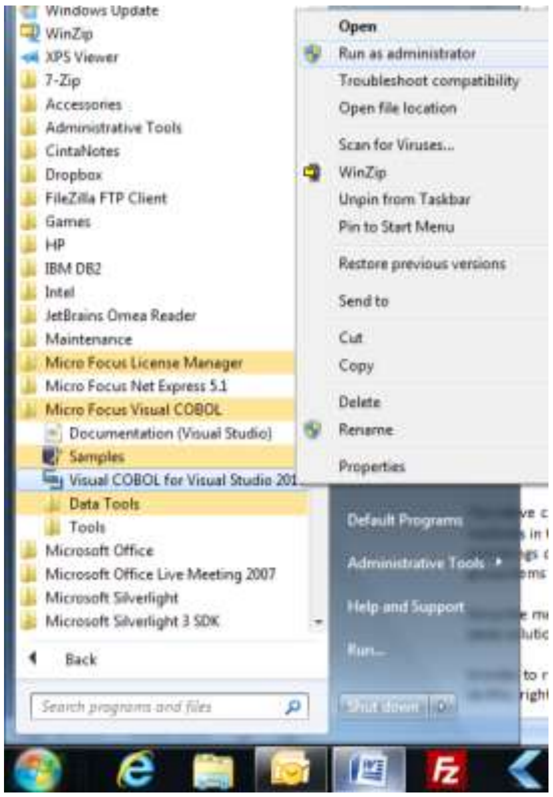
In this tutorial you will be shown how to setup and use a Visual COBOL solution containing a native code Application project that calls a class in a managed code Class Library project. The technology that allows managed code classes to be called from native code is called COM Callable Wrapper or CCW for short.

Visual COBOL does a lot of work under the covers to make this somewhat complicated technology seem quite simple. The managed code assembly is registered as a COM object which makes all of its public methods available to COM client programs.

The native code COBOL programs use the COM client support provided by Visual COBOL to invoke the methods in the managed classes as if they were standard COM methods. Native COM data types like integers and strings can be passed between native COBOL and managed COBOL using these techniques. Even COBOL group items can be passed using these same techniques.

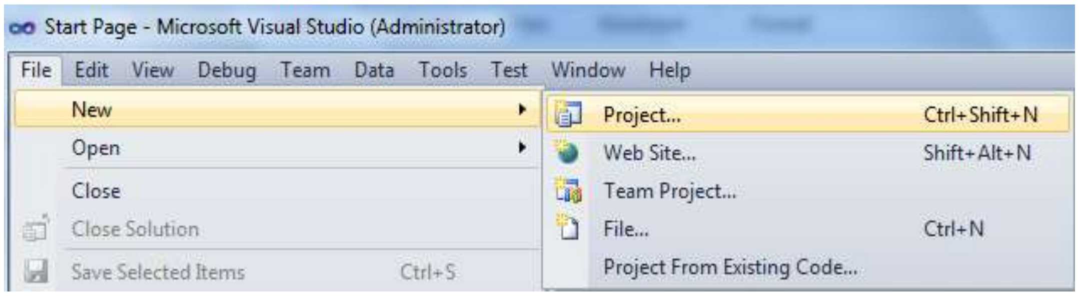
Since the managed code class is registered as COM, the native program that calls it is not required to be in the same solution, although in this tutorial the two projects are in the same solution for ease of use.

In order to register the managed assembly for COM Interop, Visual COBOL must be run as Administrator. To do this, navigate to the Windows Start Menu→All Programs→Micro Focus Visual COBOL group and right click on Visual COBOL for Visual Studio 2010 and select the option Run as administrator as shown below:

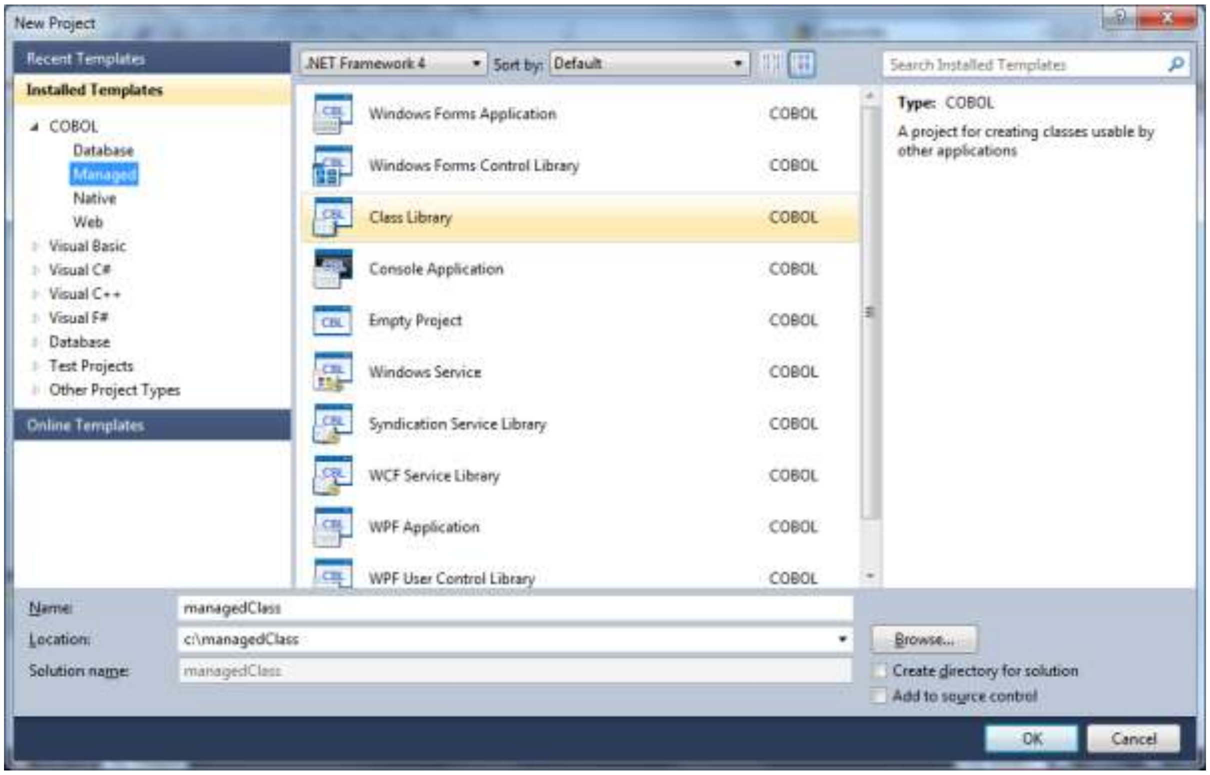


If prompted by UAC to allow this click “Yes”.

When Visual COBOL starts, from the main menu select New→Project as shown below:

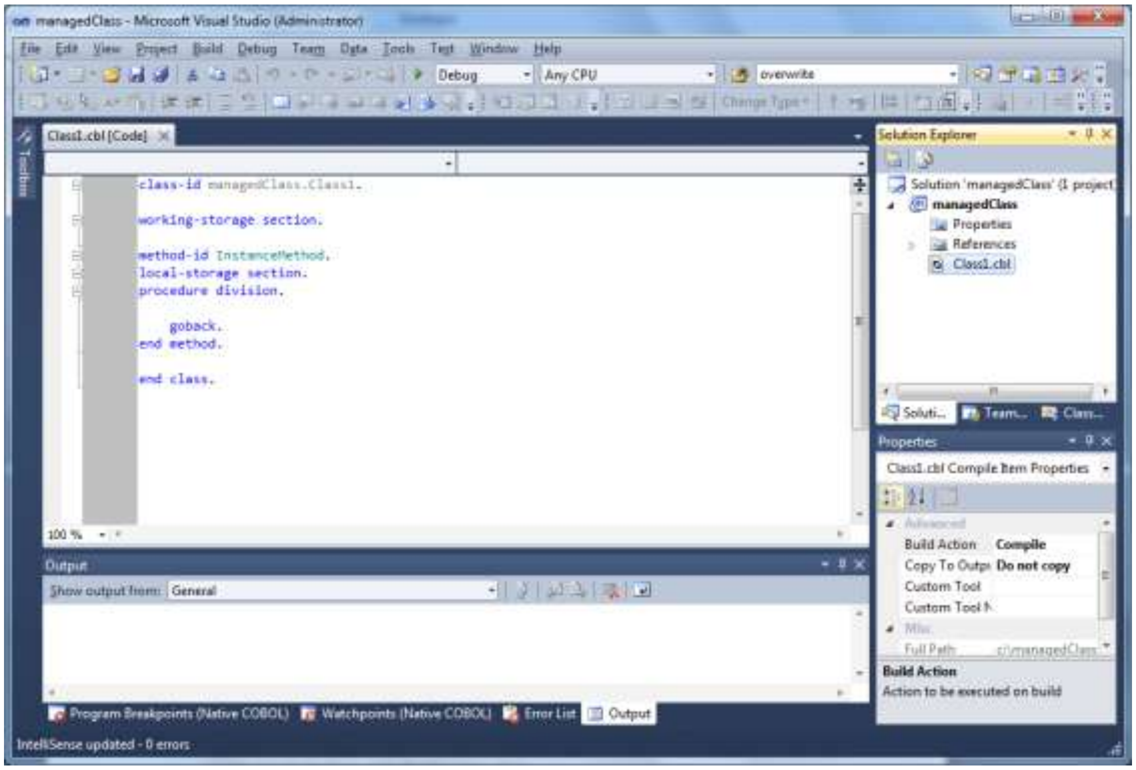

Start Page - Microsoft Visual Studio (Administrator)
File Edit View Debug Team Data Tools Test Window Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Team Project...
File... Ctrl+N
Project From Existing Code...

On the New Project Dialog select Managed under COBOL, highlight Class Library and then change the Project Name and Location to managedClass and C:\managedClasse respectively. Also uncheck the option for Create Directory for Solution so that your project will have the same folder structure as shown in this tutorial. Click OK to create the new project.



Name: managedClass
Location: c:\managedClass
Solution name: managedClass

Visual COBOL will automatically create a solution with the same name as your project file and will add a new Class1.cbl file to the project. If you do not see the Solution Explorer Window or the Properties Window you can select to display them under the View menu item.


managedClass - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Test Window Help
Debug
Any CPU
overwrite
Class1.cbl [Code]
Solution Explorer
Solution 'managedClass' (1 project)
managedClass
Properties
References
Class1.cbl
Properties
Class1.cbl Compile Item Properties
Output
Show output from: General
Program Breakpoints (Native COBOL)
Watchpoints (Native COBOL)
Error List
Output
IntelliSense updated - 0 errors

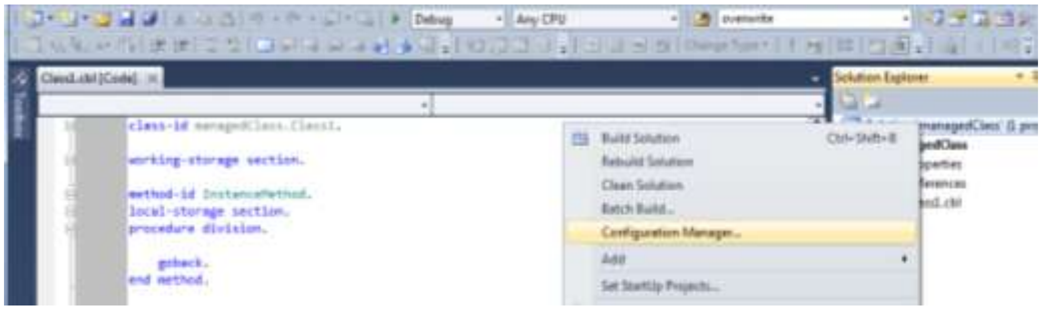
```
class-id managedClass.Class1.

working-storage section.

method-id InstanceMethod.
local-storage section.
procedure division.

    goback.
end method.

end class.
```

Build Action	Compile
Copy To Output	Do not copy
Custom Tool	
Custom Tool N	
Full Path	c:\managedClass

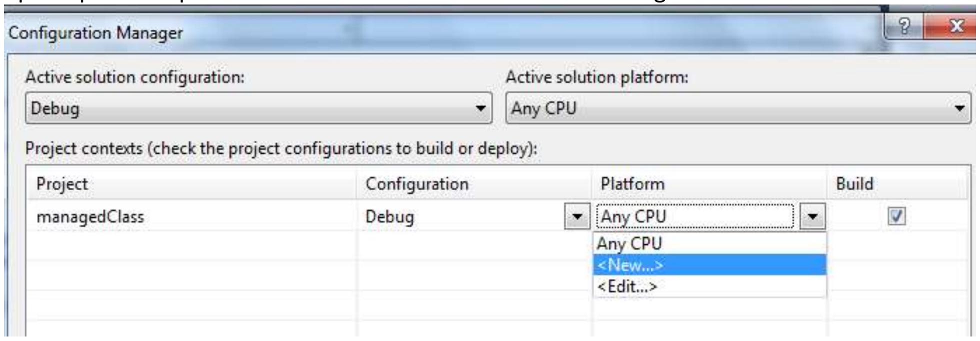
Build Action
Action to be executed on build

To simplify this demo we will create the client and server projects both as x86 (32-bit) so we must change the CPU type of the managed project from anyCPU to x86.

Start the Configuration Manager by right clicking on the Solution name in Solution Explorer, (first item) and selecting Configuration Manager from the list.



Open up the drop down list underneath the Platform heading and select <New...>


Configuration Manager

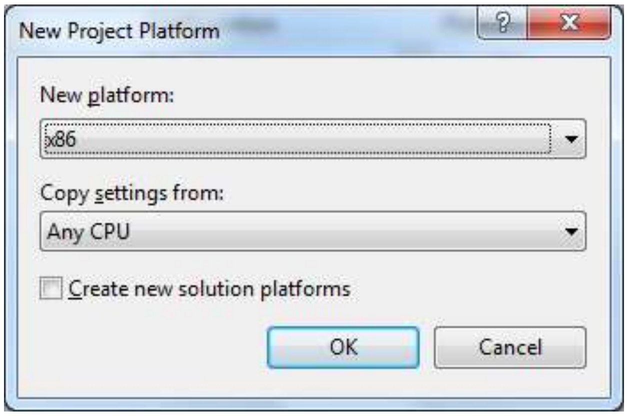
Active solution configuration: Debug

Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
managedClass	Debug	Any CPU	✓

Select x86 from the list so it looks like below and then click on OK.



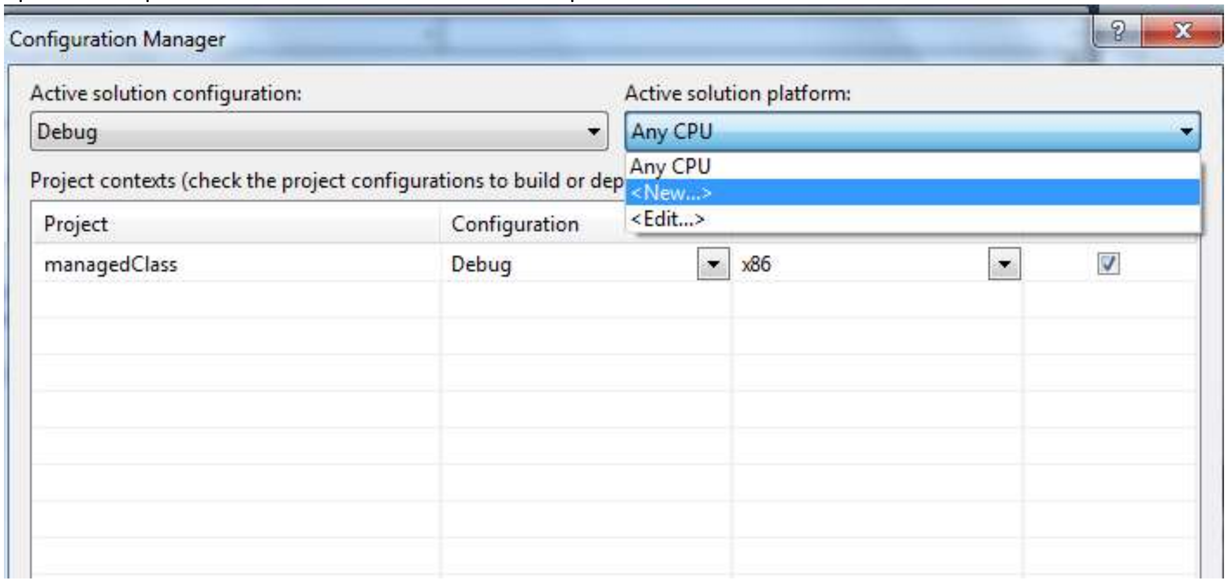
New Project Platform

New platform: x86

Copy settings from: Any CPU

☐ Create new solution platforms

Open the drop down list underneath Active solution platform and select <New...>



Configuration Manager

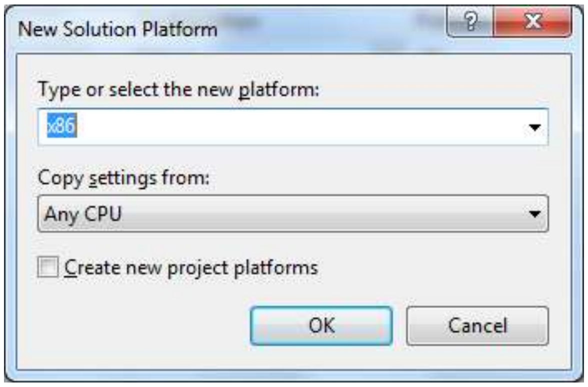
Active solution configuration: Debug

Active solution platform: Any CPU

Project contexts (check the project configurations to build or dep

Project	Configuration		
managedClass	Debug	x86	☑

Select x86 from the list so it looks like below and then click on OK.



New Solution Platform

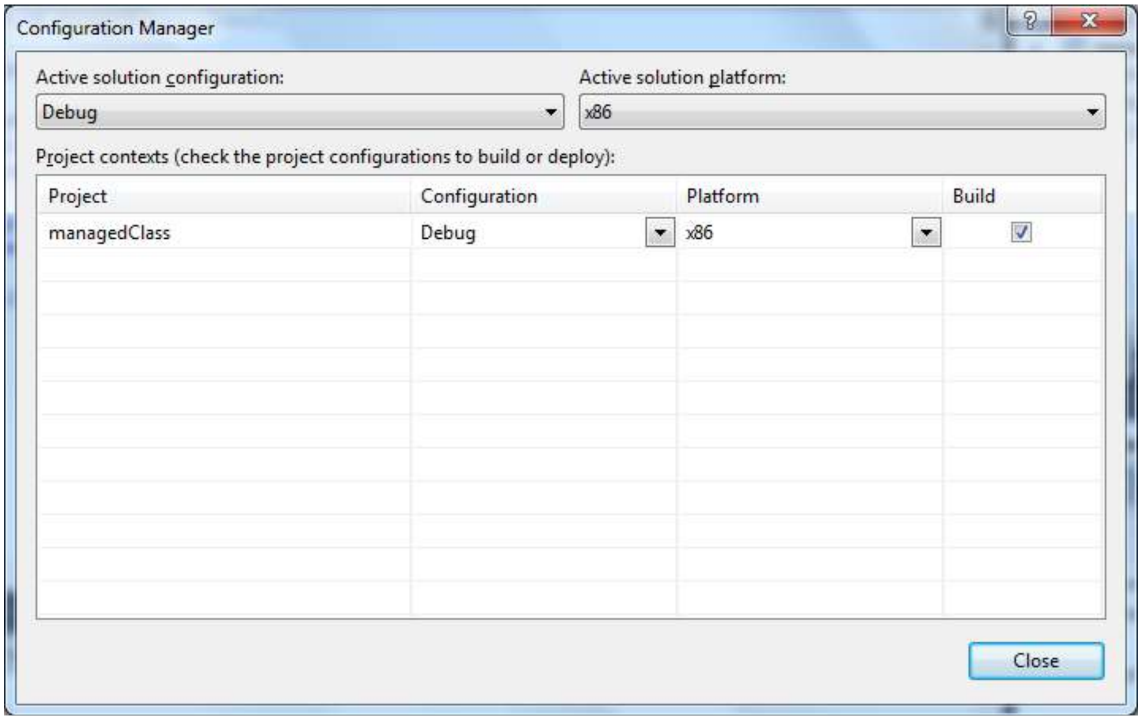
Type or select the new platform: x86

Copy settings from: Any CPU

☐ Create new project platforms

The Configuration Manager dialog should now look like below:

Click Close to finish.


Configuration Manager

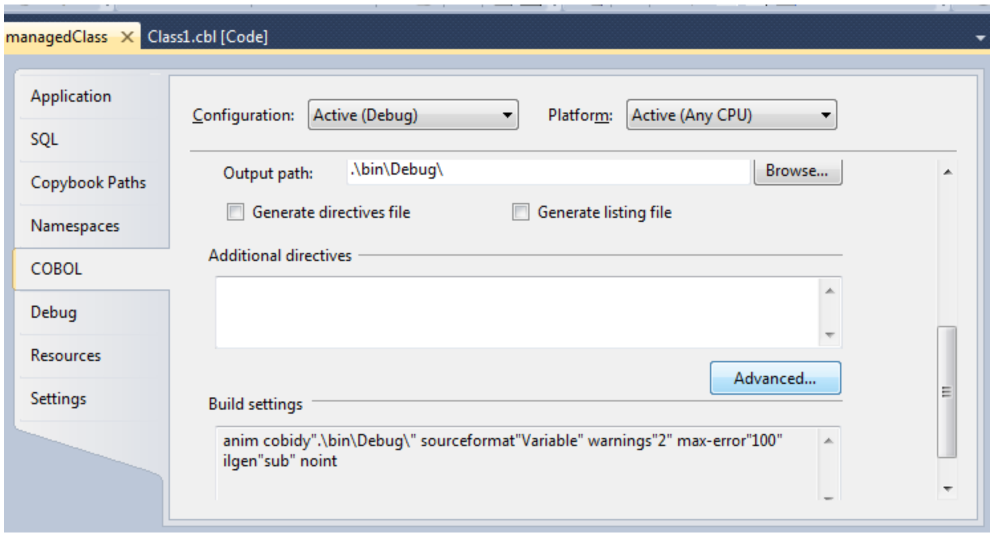
Active solution configuration: Debug

Active solution platform: x86

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
managedClass	Debug	x86	✓

We must now modify the project properties so that this project will generate the necessary resources to allow it to register its assembly as a COM server. Double click on the Properties header under the managedClass project in Solution Explorer and select the COBOL tab. Under the Additional Directives field click the Advanced button.



Configuration: Active (Debug)

Platform: Active (Any CPU)

Output path: .\bin\Debug\

Generate directives file

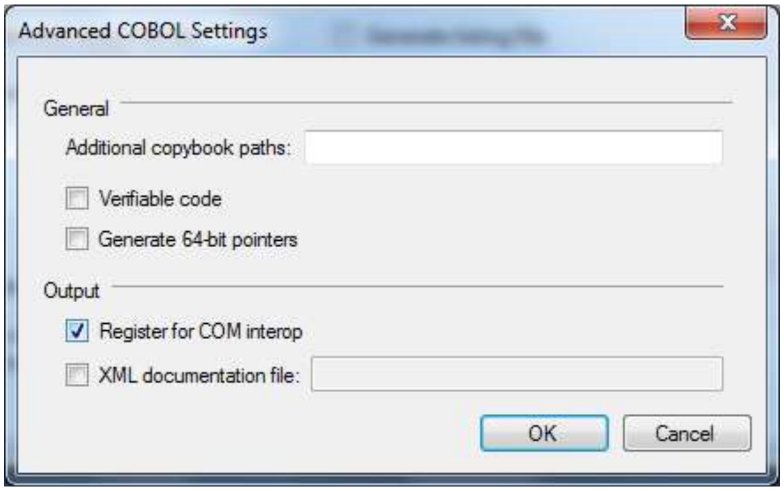
Generate listing file

Additional directives

Build settings

```
anim cobidy".\bin\Debug\" sourceformat"Variable" warnings"2" max-error"100"
ilgen"sub" noint
```

On the Advanced COBOL Settings dialog check the box next to Register for COM interop and click OK.



Click the save all icon on the Visual Studio toolbar and then close the properties page by clicking the X on the editor tab next to its name.

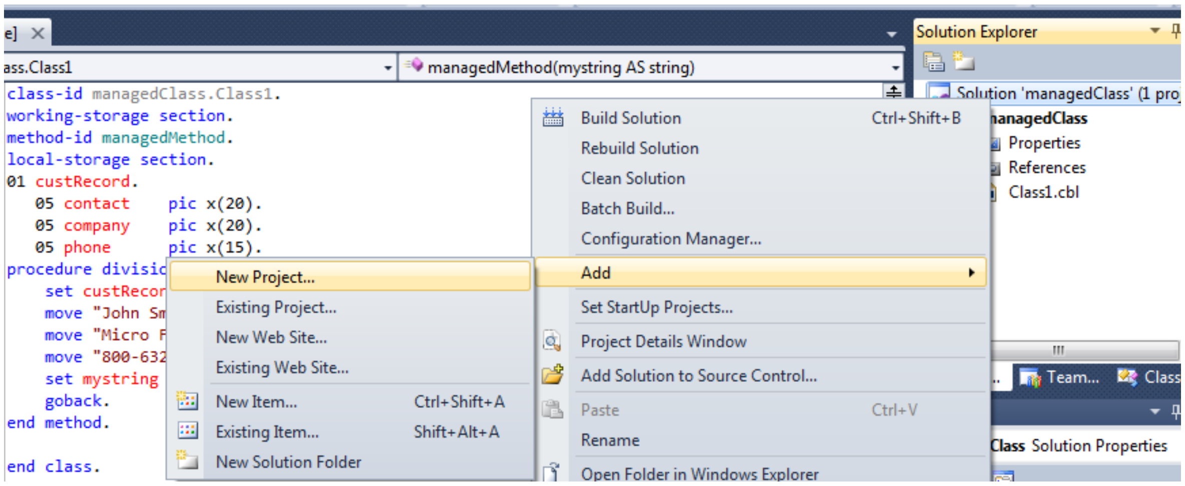
Edit Class1.cbl so that it looks like below.

```
class-id managedClass.Class1.
working-storage section.
method-id managedMethod.
local-storage section.
01 custRecord.
   05 contact    pic x(20).
   05 company    pic x(20).
   05 phone      pic x(15).
procedure division using mystring as string.
    set custRecord to mystring
    move "John Smith" to contact
    move "Micro Focus" to company
    move "800-632-6265" to phone
    set mystring to custRecord
    goback.
end method.

end class.
```

Build the Solution by selecting “Build Solution” from the Visual Studio Build menu item. If you get an error that the class cannot be registered then you probably are not running Visual COBOL as Administrator and you will have to restart Visual COBOL by right-clicking and selecting Run As administrator.

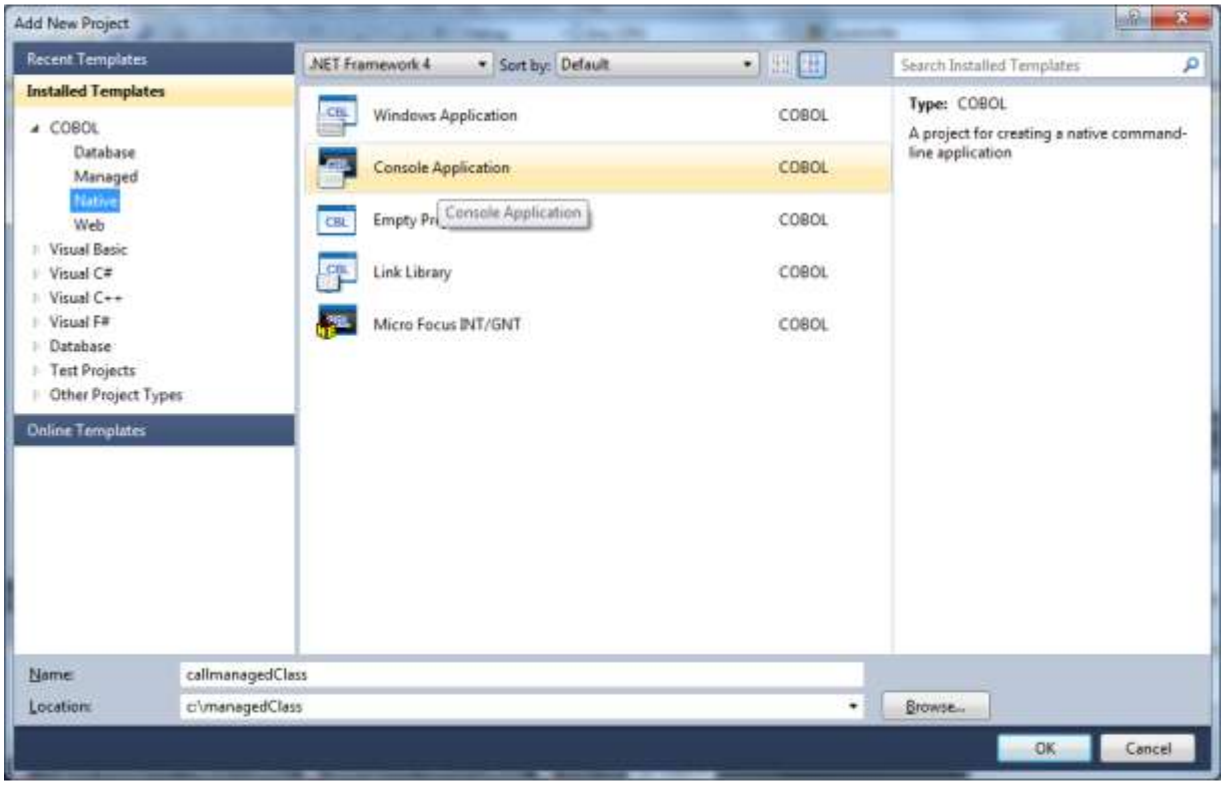
Now we will create the native COBOL program that will call our managedClass through a COM interface.

Right-click on the Solution name (top entry) in Solution Explorer and select Add→New Project.



```
class-id managedClass.Class1.
working-storage section.
method-id managedMethod.
local-storage section.
01 custRecord.
   05 contact     pic x(20).
   05 company     pic x(20).
   05 phone       pic x(15).
procedure divisic
    set custRecor
    move "John Sm
    move "Micro F
    move "800-632
    set mystring
    goback.
end method.

end class.
```

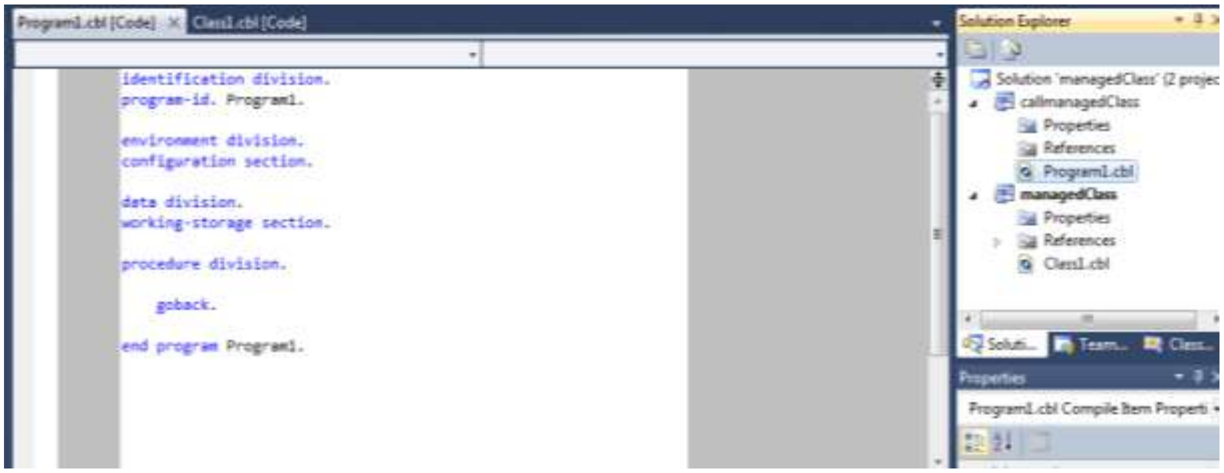
Select Native under COBOL and then Console Application as the project type. Change the name of the project to callmanagedClass and leave the Location set to c:\managedClass so that the new project will be in a subfolder of the main solution. Click Add to add the new project to the solution as shown below:



Type: COBOL
A project for creating a native command-line application

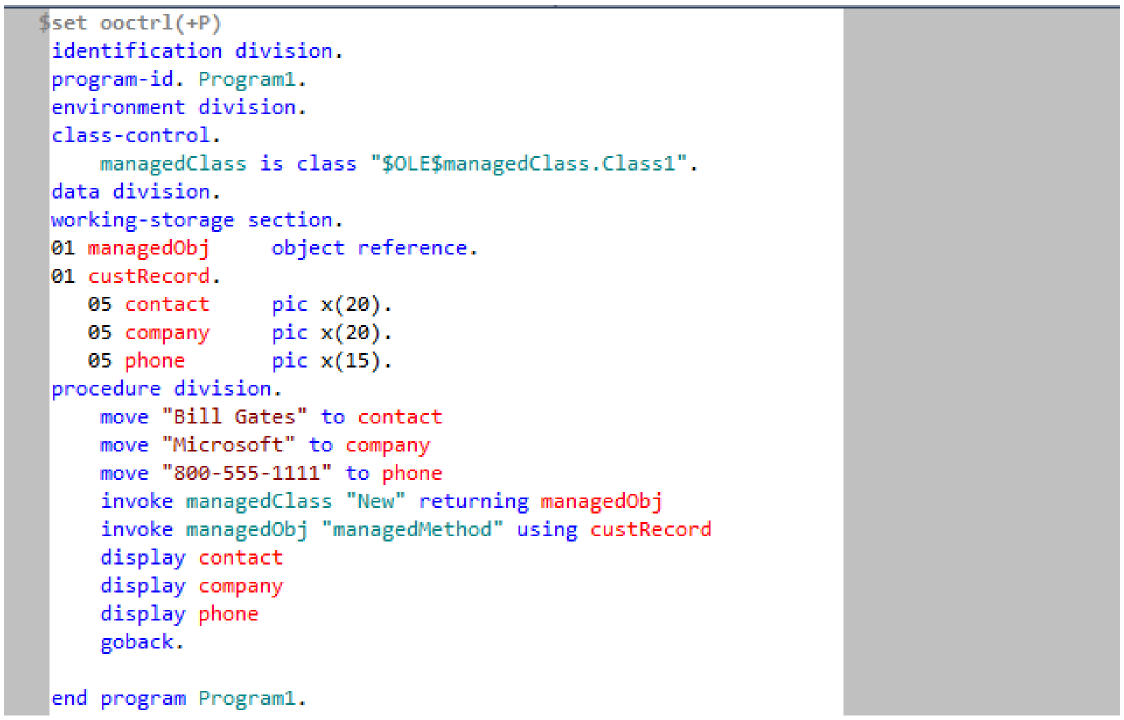
Name: callmanagedClass
Location: c:\managedClass

The new project callmanagedClass will be created and the default program Program1.cbl will be added to it.



```
identification division.
program-id. Program1.

environment division.
configuration section.

data division.
working-storage section.

procedure division.

    goback.

end program Program1.
```

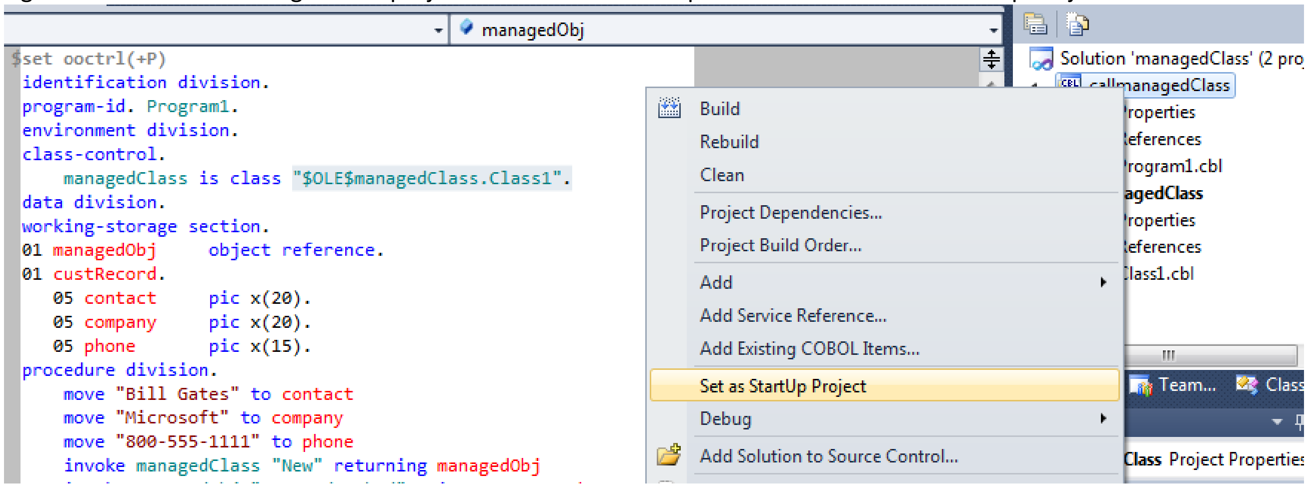
Modify the source code to Program1.cbl in the editor so that it looks exactly like the image below:



```
$set ooctrl(+P)
identification division.
program-id. Program1.
environment division.
class-control.
    managedClass is class "$OLE$managedClass.Class1".
data division.
working-storage section.
01 managedObj     object reference.
01 custRecord.
   05 contact     pic x(20).
   05 company     pic x(20).
   05 phone       pic x(15).
procedure division.
    move "Bill Gates" to contact
    move "Microsoft" to company
    move "800-555-1111" to phone
    invoke managedClass "New" returning managedObj
    invoke managedObj "managedMethod" using custRecord
    display contact
    display company
    display phone
    goback.

end program Program1.
```

Notice that this native program uses object-oriented syntax but it is different than the OO syntax used in managed code program. The native syntax is the old style OO syntax that Visual COBOL inherited from Net Express. In the class name "$OLE$managedClass.Class1" the $OLE$ tells the run-time that this is a COM server and managedClass.Class1 is the name of the class which is derived from the program-id of the COB class we are calling.

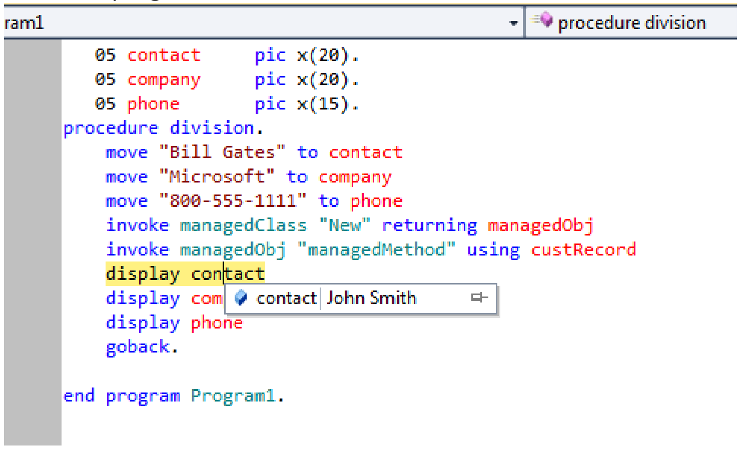
To debug this program we must change the native project callmanagedClass to be the startup program. Right-click on the callmanagedClass project name in Solution Explorer and select Set as Startup Project.



```
$set ooctrl(+P)
identification division.
program-id. Program1.
environment division.
class-control.
    managedClass is class "$OLE$managedClass.Class1".
data division.
working-storage section.
01 managedObj      object reference.
01 custRecord.
   05 contact      pic x(20).
   05 company      pic x(20).
   05 phone        pic x(15).
procedure division.
    move "Bill Gates" to contact
    move "Microsoft" to company
    move "800-555-1111" to phone
    invoke managedClass "New" returning managedObj
```

Press F11 to start stepping through the callManagedClass program. After you step the line invoking managedMethod, control returns to the next statement instead of debugging the source code to the managedClass program. This is due to the restriction that you can debug a native program or a managed program in a single session but you cannot debug both simultaneously.

Check the values of the data parameters returned from the invoke to show that they have been modified by the called program.

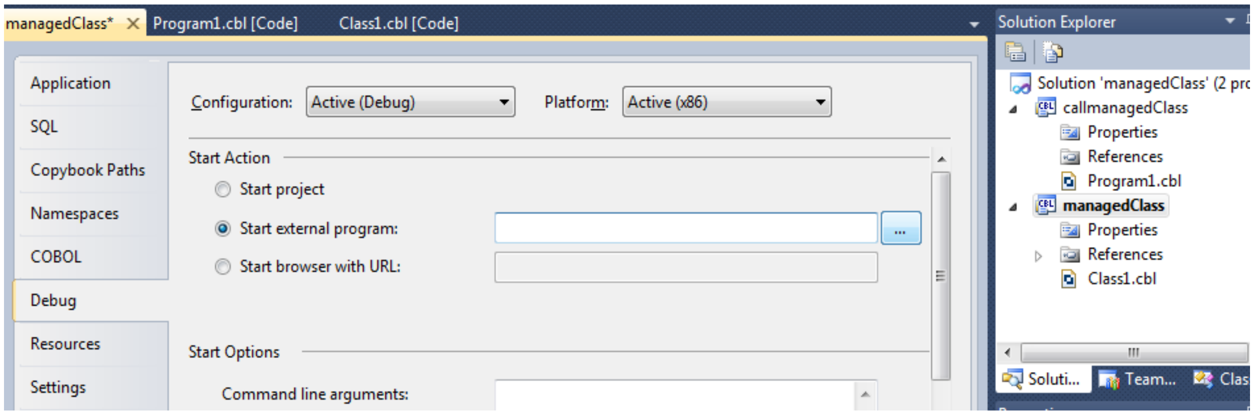


```
    05 contact      pic x(20).
    05 company      pic x(20).
    05 phone        pic x(15).
procedure division.
    move "Bill Gates" to contact
    move "Microsoft" to company
    move "800-555-1111" to phone
    invoke managedClass "New" returning managedObj
    invoke managedObj "managedMethod" using custRecord
    display contact
    display com
    display phone
    goback.

end program Program1.
```

Finish stepping through the program to completion. To debug the managed code side we need to make a couple of changes to the project properties of managedClass.

First right-click on the managedClass project name in Solution Explorer and select Set as Startup Project.

Open up the project properties for the managedClass project by double-clicking on Properties under the managedClass project header in Solution Explorer.



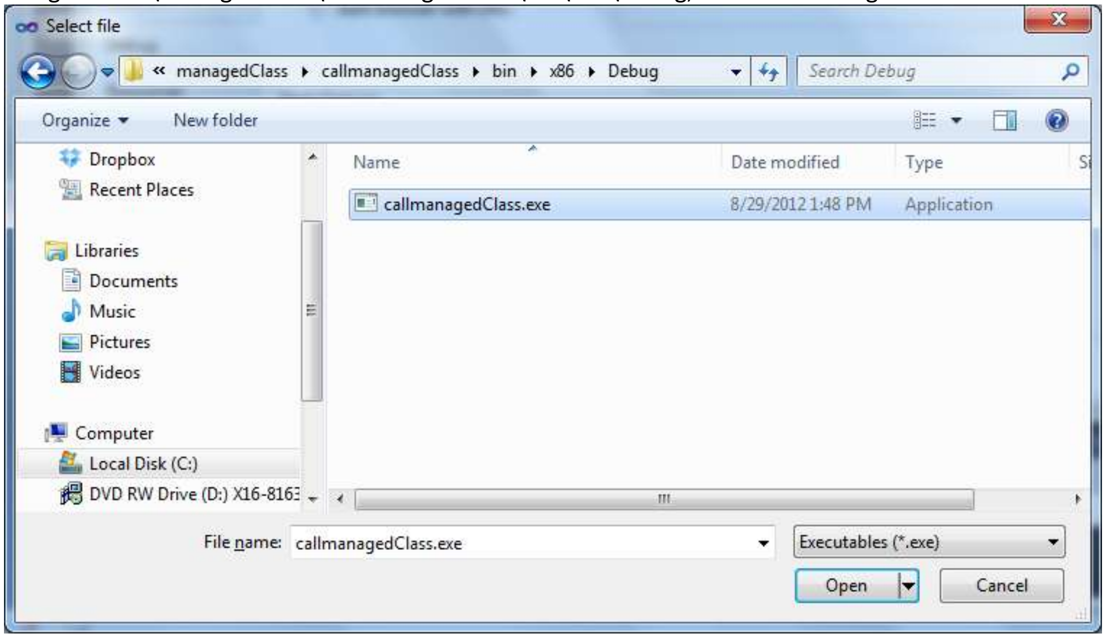
Configuration: Active (Debug) Platform: Active (x86)

Start Action
Start project
[✓ Start external program:]
Start browser with URL:

Start Options
Command line arguments:

Click on the ellipsis (…) to the right of the Start external program field so that we can select the program to start when we begin debugging. This will be the native code program callmanagedClass.exe that calls the managed program.

Navigate to c:\managedClass\callmanagedClass\bin\x86\debug, select callmanagedClass.exe and click Open.

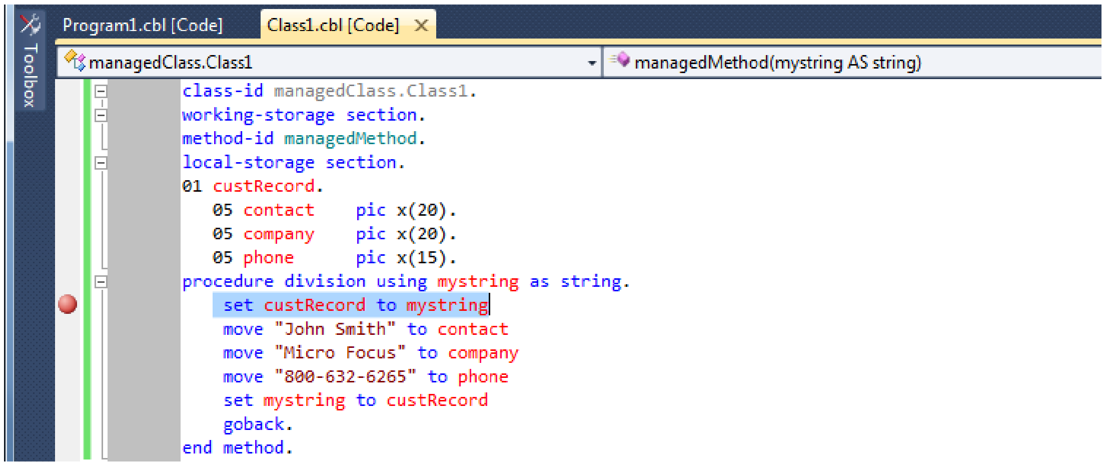


Name	Date modified	Type
callmanagedClass.exe	8/29/2012 1:48 PM	Application

File name: callmanagedClass.exe Executables (*.exe)

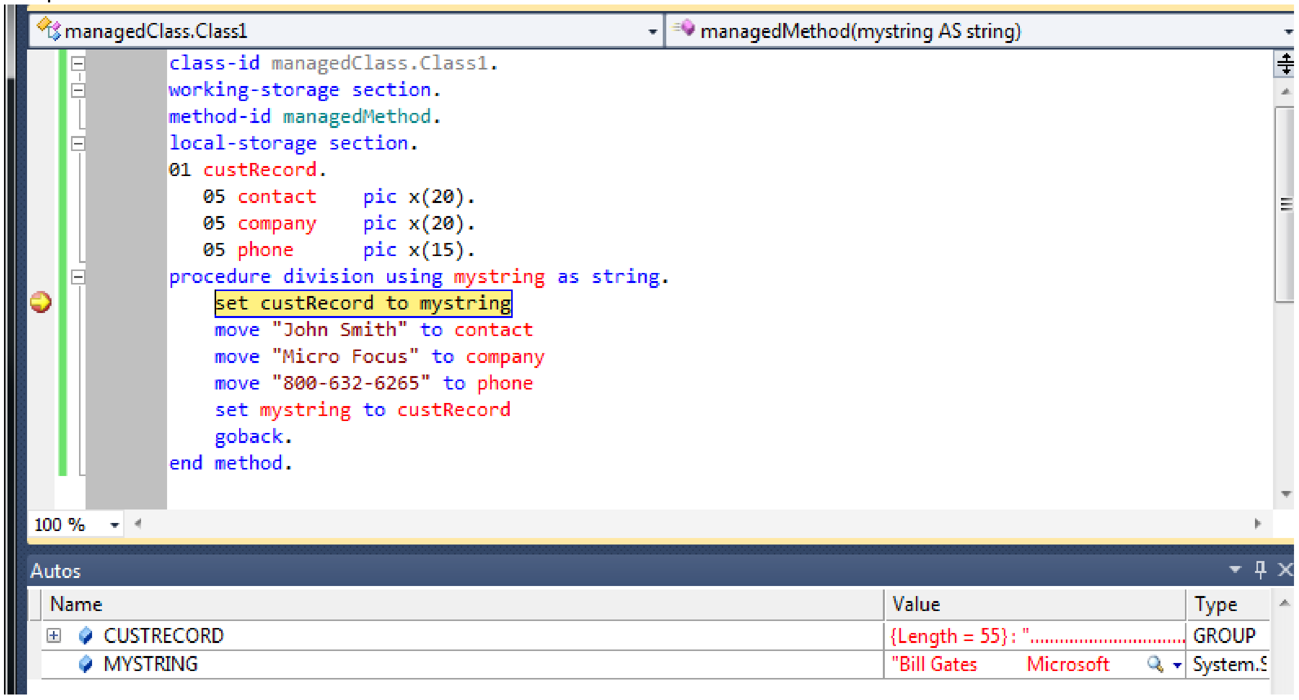
Click on the Visual Studio save icon to save the change and close the property page

Open up Class1.cbl in the editor and set a breakpoint on the first statement in the procedure division by clicking the mouse in the first position of the line. Click where the red stop icon is displayed in the image below.



```
class-id managedClass.Class1.
working-storage section.
method-id managedMethod.
local-storage section.
01 custRecord.
   05 contact    pic x(20).
   05 company    pic x(20).
   05 phone      pic x(15).
procedure division using mystring as string.
    set custRecord to mystring
    move "John Smith" to contact
    move "Micro Focus" to company
    move "800-632-6265" to phone
    set mystring to custRecord
    goback.
end method.
```

Press F11 to start debugging and the callmanagedClass.exe program will be run at full speed and the current line in Class1.cbl will be highlighted and will be ready to debug. Step though the rest of the code until complete.



```
class-id managedClass.Class1.
working-storage section.
method-id managedMethod.
local-storage section.
01 custRecord.
   05 contact    pic x(20).
   05 company    pic x(20).
   05 phone      pic x(15).
procedure division using mystring as string.
    set custRecord to mystring
    move "John Smith" to contact
    move "Micro Focus" to company
    move "800-632-6265" to phone
    set mystring to custRecord
    goback.
end method.
```

Name	Value	Type
CUSTRECORD	{Length = 55} : "...............................	GROUP
MYSTRING	"Bill Gates Microsoft	System.S

This completes the section on calling managed code from native code using CCW.

III) Summary

We have covered a number of different scenarios here in the preceding tutorials, some of which may or may not be applicable to your particular application.

The chart below summarizes the techniques that we covered in these pages and outlines under which scenarios each can be used.

	All Native	INT/GNT	All Managed	Managed to Native P/Invoke	Native to Managed CCW
Common Output Folder	X	X	X	X	NA
PATH in app.config	X		X	X	NA
COBPATH in app.config		X			NA
MFENTMAP	X	X	X	X	NA
Cobconfig required for MFENTMAP	X	X			NA
Preload section in app.config			X		NA
Add reference to projects			X		NA
Add reference to .dlls			X	X	NA
Multiple Output Projects	X	X		X	NA
SET PROC-POINTER TO ENTRY	X	X	X	X	NA