# CAPE-OPEN for .Net Class Library

The Cape-Open for .Net class library is a collection of classes that implement the Cape-Open v.1.0 interfaces in the .Net framework. This is a tool to aid process modeling component (PMC) developers in producing CAPE-OPEN compliant objects using the latest version of the Visual Studio integrated development environment.

The installation package will install the class library assembly on your computer, register it with Visual Studio, and install a class documentation help file that can be access from the Start menu (click the "Start" button, select "All Programs," select the "CapeOpen.Net" folder and run the "Documentation.chm" file.). The documentation file current contains class documentation. It will be expanded to include a tutorial, which is presented below.

This class library is being distributed for testing and evaluation purposes only. If you have any comment, please contact Bill Barrett at barrett.williamm@epa.gov.

**Disclaimer of Liability:**

With respect to the Cape-Open.Net software and documentation, neither the United States Government nor any of their employees, assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed. Furthermore, software and documentation are supplied "as-is" without guarantee or warranty, expressed or implied, including without limitation, any warranty of merchantability or fitness for a specific purpose.

**Disclaimer of Endorsement:**

Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United Sates Government. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government, and shall not be used for advertising or product endorsement purposes.

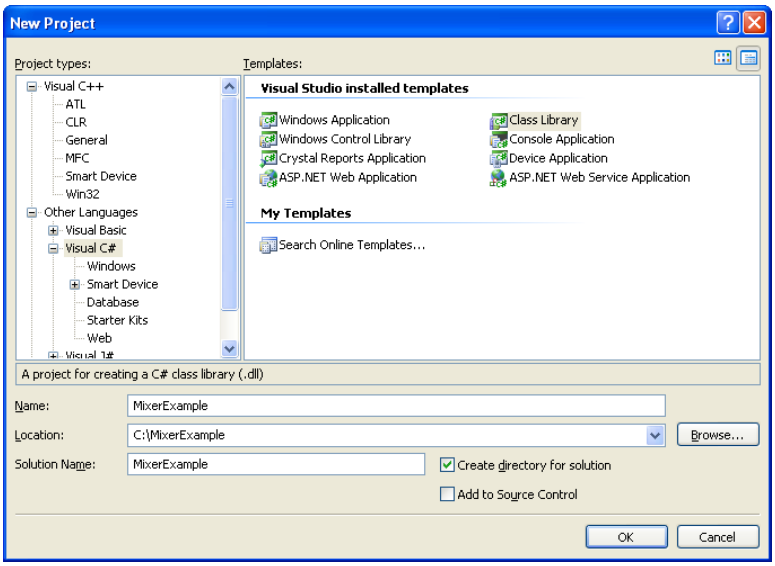
**Copyright Status:**

The U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the documents and software provided. The software and documentation may be freely distributed and may be used for testing and evaluation purposes.

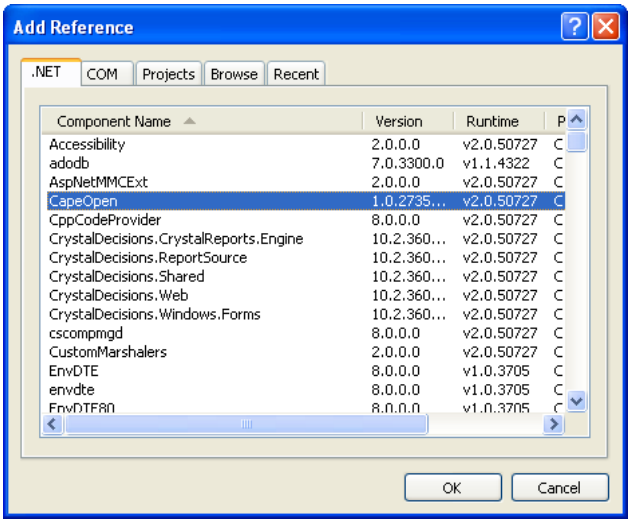
# Creation of a Mixer class in C-Sharp

This example will demonstrate the use the Cape-Open.Net class library to create a Mixer unit operation using the C-Sharp language in Visual Studio.

**Step 1:** Create the Project. This is a standard process in Visual Studio. From the file menu, you select "New->Project…:" and complete the "New Project" dialog as shown below. Click the "OK" button to create the project.

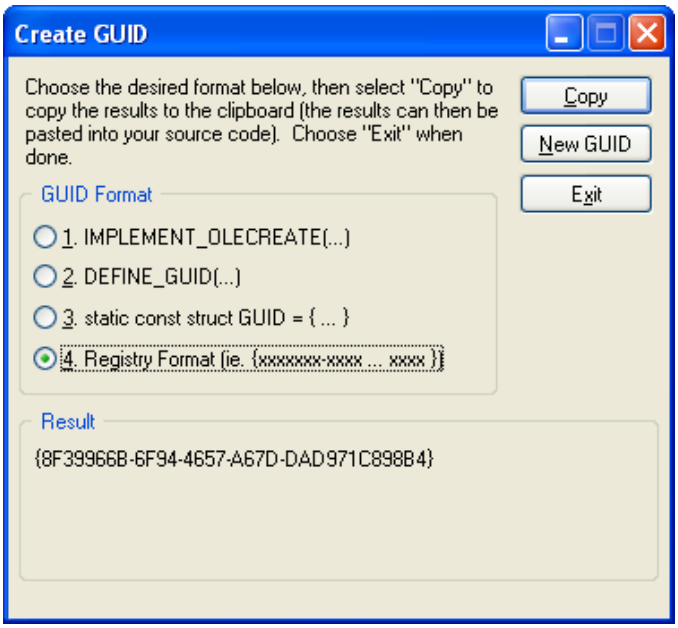



**Step 2:** Add a reference to the Cape-Open.Net class library to the project. This can be initiated by either clicking the "Project->Add Reference…" menu item or right clicking the "References" item in the Solution Explorer and selecting "Add Reference…" from the context menu. This will bring up the "Add References" dialog. Select CapeOpen from the list and click the "OK" button.



**Step 3:** The unit operation needs to derive from the *CapeOpen.CUnitBase* base class and needs to be marked with the *Serializable* attribute. The base class implements all required Cape-Open interfaces for a unit operations except the *ICapeUnit.Calculate()* method, which will be implemented below. The *Serializable* attribute allows the unit to use .Net-based serialization for persistence (saving to disk). The ComVisible(true) and the GUID attributes are added for registering the class for COM interop.

Please note, the GUID for each class is a unique identifier. You will need to create a new GUID using the CreateGUID tool in Visual Studio, as shown below:




Click the "Copy" button to copy the GUID in registry format, replace the GUID in the GUID attribute (it will be surrounded by {}, which need removed). The class library also provides custom attributes for exposing the CapeDescription registry keys, as shown below. Once completed, the class will look like this:

```
[Serializable]
[CapeOpen.CapeDescriptionAttribute("Mixer example class written in C#.")]
[CapeOpen.CapeVersionAttribute("1.0")]
[CapeOpen.CapeVendorURLAttribute("http:\\www.epa.gov")]
[CapeOpen.CapeHelpURLAttribute("http:\\www.epa.gov")]
[CapeOpen.CapeAboutAttribute("US Environmental Protection Agency\nCincinnati,
Ohio")]
[System.Runtime.InteropServices.ComVisible(true)]
[System.Runtime.InteropServices.Guid("8F39966B-6F94-4657-A67D-DAD971C898B4")]
    public class Class1: CapeOpen.CUnitBase
    {
    }
}
```

And building the class should provide the following error:

```
C:\MixerExample\MixerExample\MixerExample\Class1.cs(8,18): error CS0534: 'MixerExample.Class1'
does not implement inherited abstract member 'CapeOpen.CUnitBase.Calculate()'
c:\Program Files\USEPA\CapeOpen\CapeOpen.dll: (Related file)
```

**Step 4:** Implement the unit operation constructor and/or initialize method. The parameter and port collections are created in the *CapeOpen.CUnitBase* class constructor, which is called prior to the derived class's constructor during object creation. However, the CAPE-OPEN specification states that ports and parameters collections be created in the *ICapeUtilities.Initialize()* method and requires that the process modeling environment calls the unit operation's *ICapeUtilities.Initialize()* method prior to using the PMC. The initialize method is implemented as a virtual function that does nothing in the *CapeOpen.CUnitBase* class. You may choose to add ports and parameters in either the Class constructor or the *ICapeUtilities.Initialize()* method. Below shows adding the *Initialize()* method to the unit operation. The code will look like this:

```
[Serializable]
    [Serializable]
    [CapeOpen.CapeNameAttribute("MixerExample")]
    [CapeOpen.CapeDescriptionAttribute("Mixer example class written in C#.")]
    [CapeOpen.CapeVersionAttribute("1.0")]
    [CapeOpen.CapeVendorURLAttribute("http:\\www.epa.gov")]
    [CapeOpen.CapeHelpURLAttribute("http:\\www.epa.gov")]
    [CapeOpen.CapeAboutAttribute("US Environmental Protection
Agency\nCincinnati, Ohio")]
    [System.Runtime.InteropServices.ComVisible(true)]
    [System.Runtime.InteropServices.Guid("8F39966B-6F94-4657-A67D-
DAD971C898B4")]
public class Class1: CapeOpen.CUnitBase
{
   public Class1()
   {
      this.ComponentName = "MixerExample";
      this.ComponentDescription = "Mixer easmple class written in C#";
   }

   public override void Initialize()
   {
      this.Ports.Add(new CapeOpen.CUnitPort("Inlet Port1",
             "Test Inlet Port1", CapeOpen.CapePortDirection.CAPE_INLET,
             CapeOpen.CapePortType.CAPE_MATERIAL));
      this.Ports->Add(new CapeOpen.CUnitPort("Inlet Port2",
             "Test Inlet Port2", CapeOpen.CapePortDirection.CAPE_INLET,
             CapeOpen.CapePortType.CAPE_MATERIAL));
      this.Ports->Add(new CapeOpen.CUnitPort("Outlet Port",
             "Test Outlet Port", CapeOpen.CapePortDirection.CAPE_OUTLET,
             CapeOpen.CapePortType.CAPE_MATERIAL));
      this.Parameters->Add(new CapeOpen.CRealParameter("PressureDrop",
             "Drop in pressure between the outlet from the mixer and the
             pressure of the lower pressure inlet.", 0.0, 0.0, 0.0,
             100000000.0, CapeOpen.CapeParamModeCAPE_INPUT, "Pa"));
      this.Parameters->Add(new CapeOpen.CIntParameter("Parameter2",
             12, CapeOpen.CapeParamMode.CAPE_INPUT_OUTPUT));
      this.Parameters->Add(new CapeOpen.CBoolParameter("Parameter3",
             false, CapeOpen.CapeParamMode.CAPE_INPUT_OUTPUT));
      String[] options = { "Test Value", "Another Value" };
      this.Parameters->Add(new CapeOpen.COptionParameter("Parameter4",
```

```
            "OptionParameter", "Test Value", "Test Value", options, true,
            CapeOpen.CapeParamMode.CAPE_INPUT_OUTPUT));
        this.AvailableReports->Add("Report 2");
    }
}
```

The constructor sets the unit operation's *ICapeIdentification.ComponentName* and *ICapeIdenfication.ComponentDescription* properties. The *Initialize()* method overrides the base class method and adds ports and parameters to their respective collections.

Creating Ports: Ports are implemented by the CUnitPort class. This class implements the *ICapeUnitPort* and *ICapeIdentification* interfaces. This class has a single contructor, which takes the following parameters:

String ComponentName (sets the *ICapeIdentification.ComponentName* property)
String ComponentDescription (sets the *ICapeIdentification.ComponentDescription* property)
CapePortDirection direction (sets the Port Direction property)
CapePortType type (sets the port type property)

Once you have created the port, it can be added to the port collection using the *Add()* method of the PortCollection class.

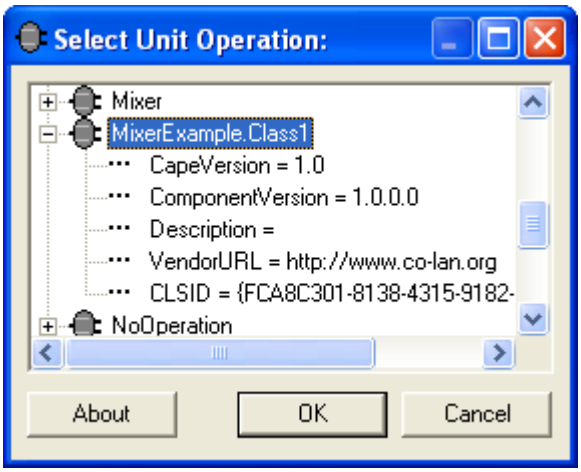
Creating Parameters: There are four (4) types of parameters available, *CRealParameter*, *CIntParameter*, *CBoolParameter*, and *COptionParameter*. Each of these implement the *ICapeIdentification*, *ICapeParameter*, and *ICapeParameterSpec* interfaces, and the appropriate specification interfaces. The constructors are overloaded (multiple versions) and descriptions of each constructor is provided in the above described documentation file.

Once you have created the parameter, it can be added to the port collection using the *Add()* method of the ParameterCollection class.

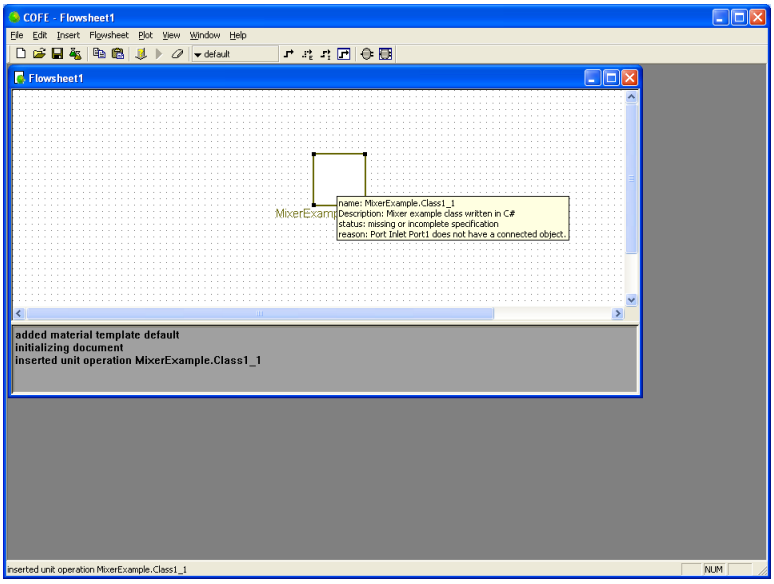
**Step 5:** Implement the unit operation's "Calculate" method. See the attached "MixerExample.cs" file for the implementation of a mixer. This example obtains the component flows, pressures, and enthalpies of the input streams using both the *CapeOpen.ICapeThermoMaterialObect* interface and the *MaterialObjectWrapper* class. The *MaterialObjectWrapper* class wraps a material object and converts the variants (objects) to type proper array type for convenience. The example provided uses the *MessageBox.Show* method to provide feedback if an exception is encountered, and requires a reference to the "System.Windows.Forms" assembly be added to your

**Step 6:** Set the IDE to register the build output. This is done by opening the project properties page, selecting the "Build" tab and selecting the "Register for COM interop" checkbox. At this point, the mixer unit operation is available for use in your process simulation application, such as COFE. It should be noted that by checking the "Register for COM interop" box, the component is only registered locally. If you would like to distribute the component to other machines, it will need to be registered with regasm.exe or by creating an installation package.

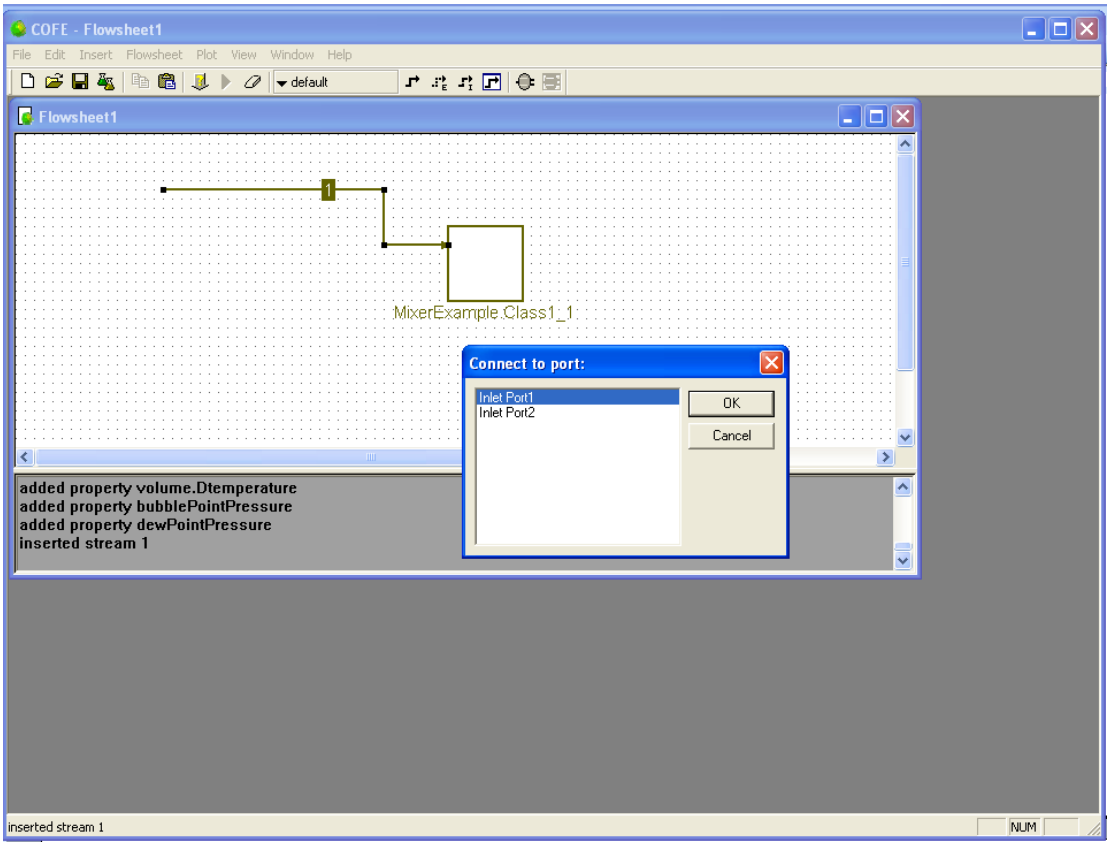
Open COFE, and select the add unit operation button, and you will see your MixerExample.Class1 in the "Select Unit Operation" dialog, as shown below.



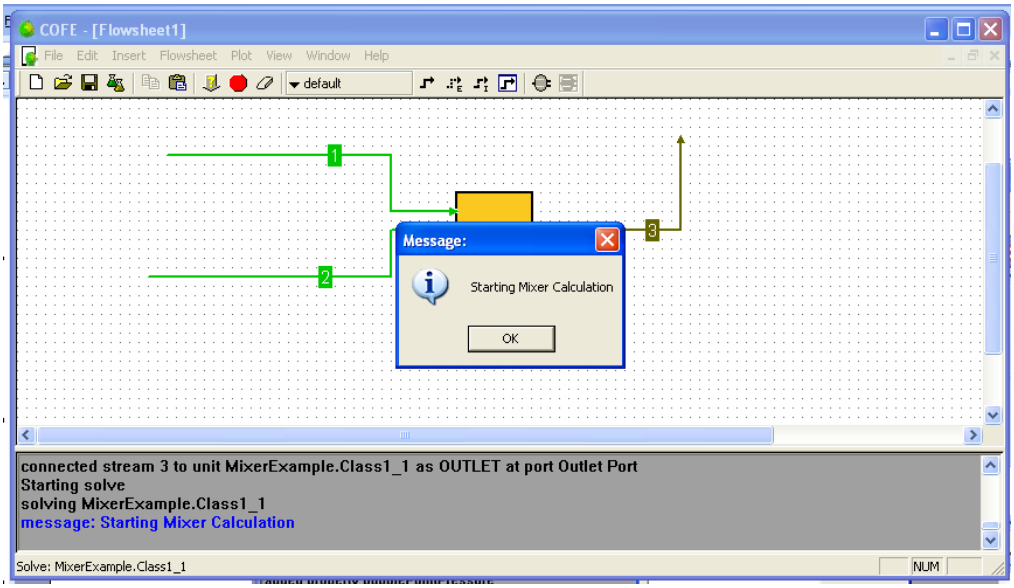

Select OK, and add the unit operation to the flowsheet:



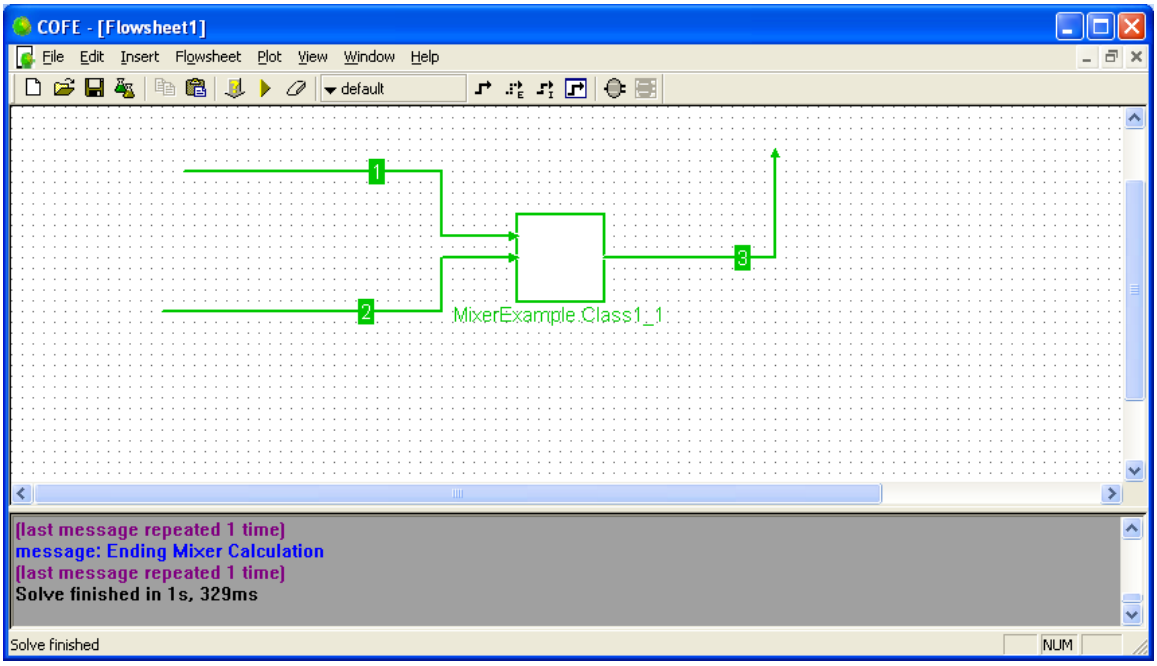
Then select a property package and attach materials to the ports.




Configure the inlet material streams and solve the flow sheet. On starting the unit operation's Calculate() method, the *ICapeDiagnostic.LogMessage* and *ICapeDiagnostic.PopUpMessage* methods are clled, as can be see below:

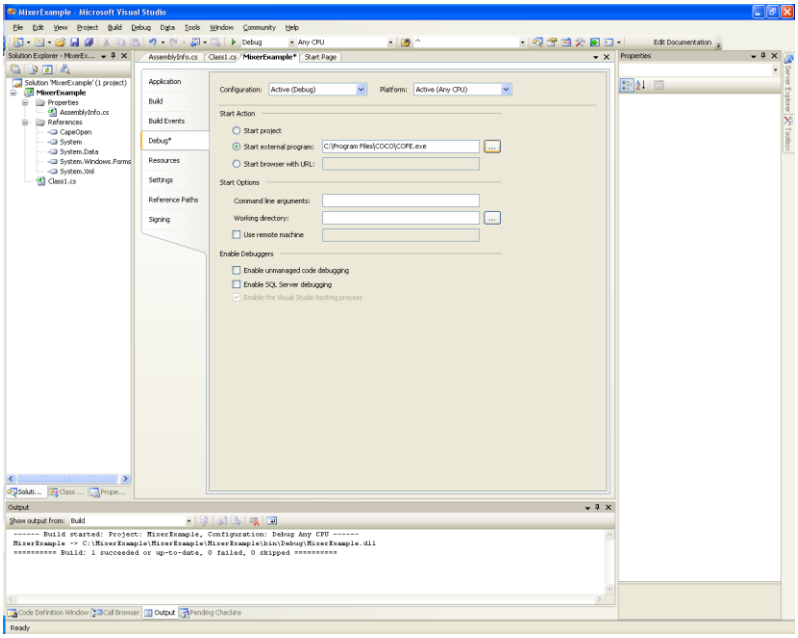


Ad the unit will calculate:



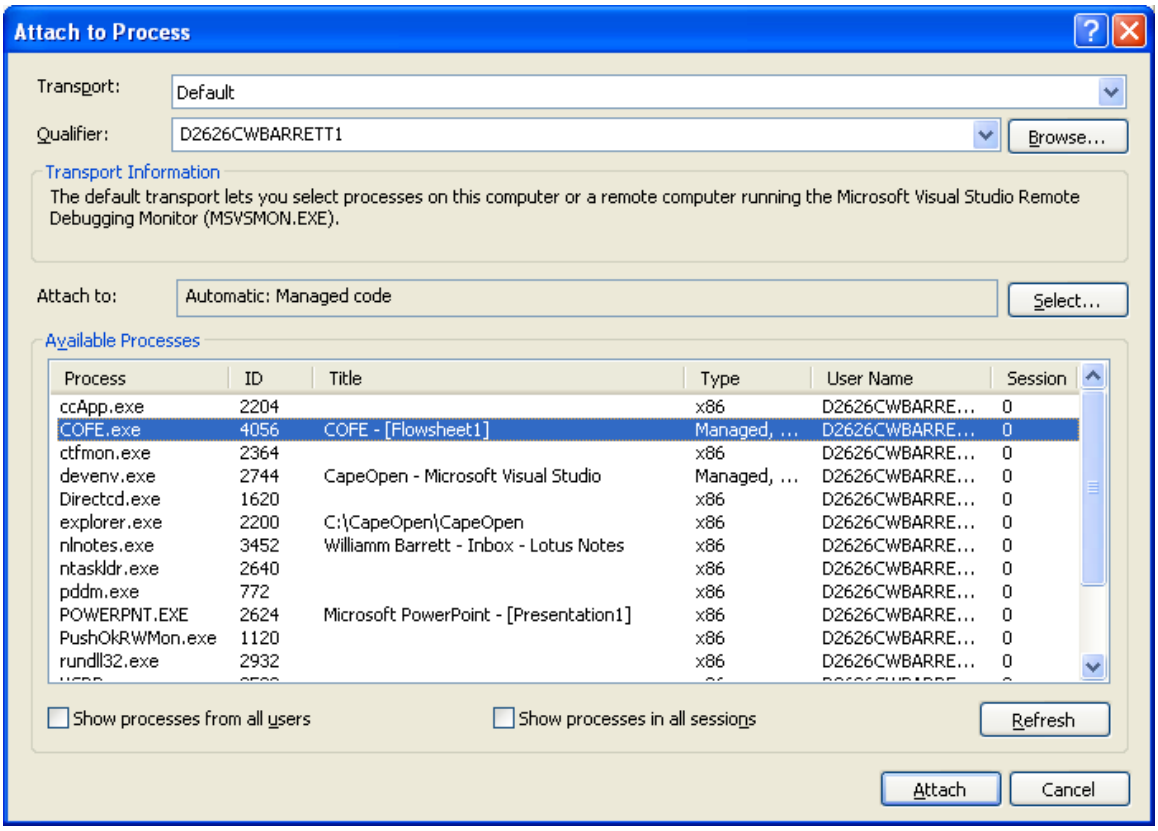

## DEBUGGING

There are two ways to access the debugger. First, you could select your desired simulation application as the external program to start on the Debugger property page, as shown:



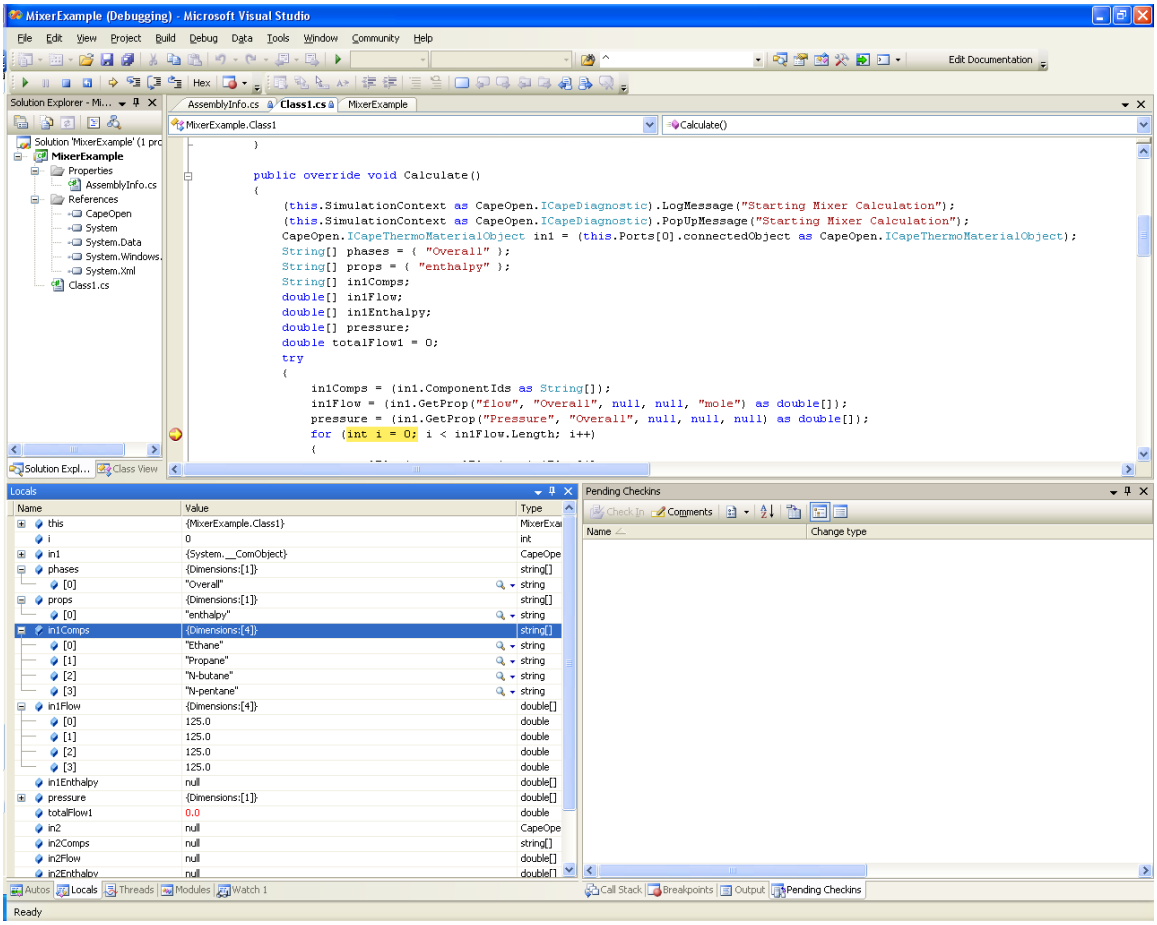
And the selected application will start as a host when you select the “Debug->Start Debugging” menu item.

Alternatively, your can attach to a running process by selecting the “Debug->Attach to process” menu item, which brings up this dialog box shown below:




Select the desired application and click the “Attach” button.

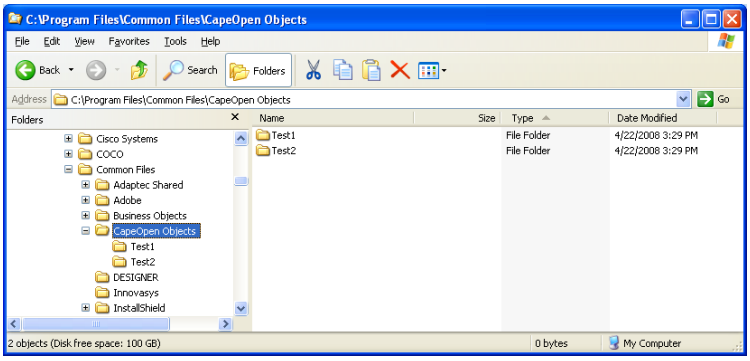
The class library will then stop at all breakpoints hit during execution, as shown below:




## Unit Operations Manager

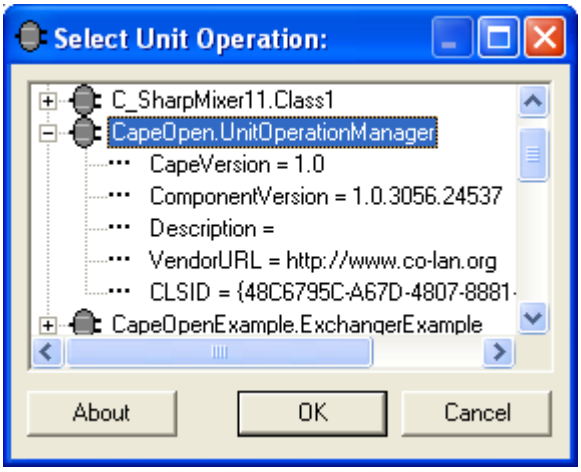
This is an analogous system to the ThermoSystem provided in the Thermodynamics package. It is intended to allow developers to create unit operations classes that can be developed, tested and installed on client machines without the need for administrative credentials, required for COM-based development.

On installation of this package, a directory will be installed in the %program files% (typically, the “C:\Program Files” directory) named “CapeOpen Objects” (see below).

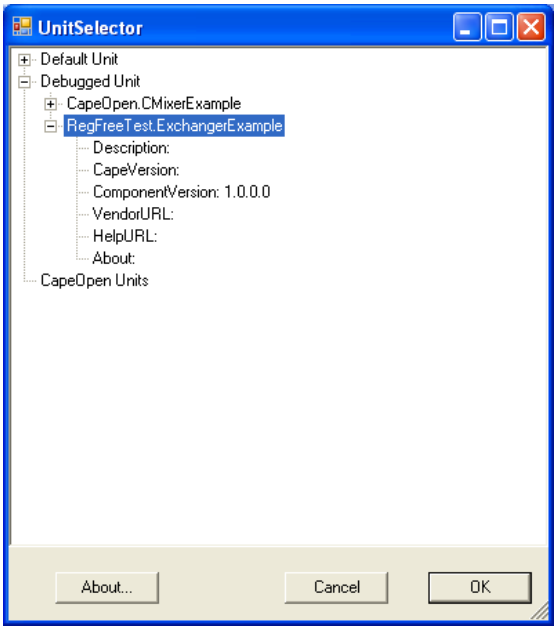


The unit operation manager will inspect “CapeOpen Objects” directory and all subdirectories for assemblies that contain Cape-Open unit operations built from this assembly. Further, if you are debugging a test unit, the unit operation manager will detect the debugger and locate any unit operations within the assembly being debugged.

To use the unit operation manager, install this package into the target computer. If you develop a unit operation as shown above, when you debug it, select the UnitOperationManager from the flowsheeting environment (as shown below):




Click OK, and the unit operation selector will be shown:




You can then select the desired unit and it will be created and inserted into the flowsheet for your use.