# Build Your Own CapeOpen Unit In 15 Minutes

**Introduction:**

CasterUnitBase is a .Net Wrapper for CAPEOPEN, it aims to make developer can concern less about CAPEOPEN standard but to think in the way of .Net and focus on the core calculation. You can forget what you don't concern, and can always customize those you need.

I try my best to make it clear for developer which kind of parameters should be passed. In most time, you don't need to pass strings which is easy to make mistake, like phase or unit category, although sometime string is still necessary, like property name or CapeCollection index.

Some features is listed below:

Build-in GUI, nothing is needed to do if you don't want to write one.

Build-in Save and Load, just put Serializable label on your own class.

Auto register for COM interop.

Material object can support both 1.0 and 1.1, unit port support material, energy and information.

Support Real, Int, Option and Boolean parameters (Array can’t work for now).

CapeOpen standard Exceptions (not all of them), start with ECape.

There are plenty errors and inappropriate things in this base block, if you find an error or have a better idea or you write a new Exception or anything useful, please do let me know, thanks. Email is [liuboqian2015@outlook.com](mailto:liuboqian2015@outlook.com).

**ShortVersion for experienced developers:**

1 Add references CasterUnitBase.dll and CAPE-OPENv1-1-0.

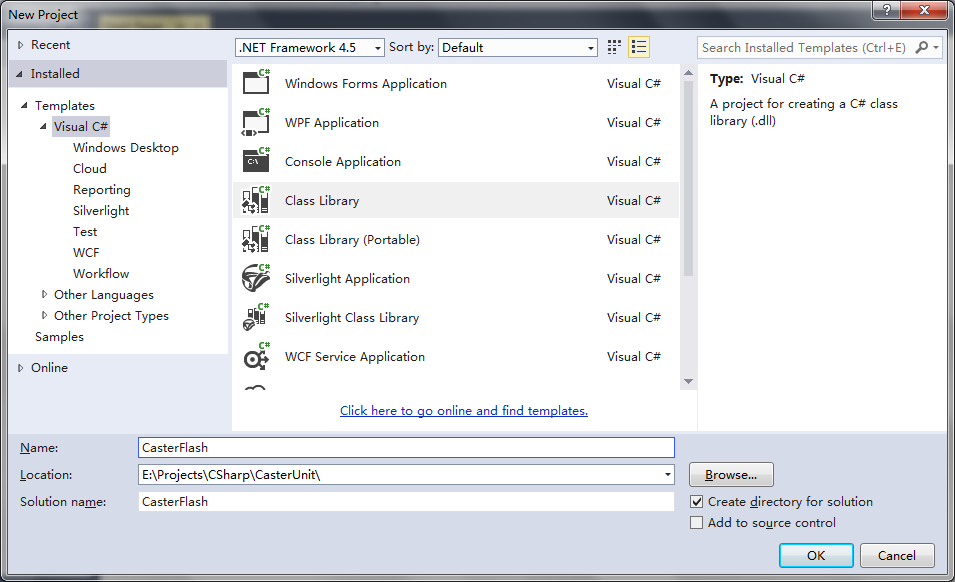
2 Add a unit class inherit CasterUnitOperationBase, implement constructor, InitParameters, InitPorts and InitResults. Add nessary attributes to this class.

3 Add a new calculator class inherit Calculator, implement BeforeCalculate, Calculate and OutputResult.

4 Debug.

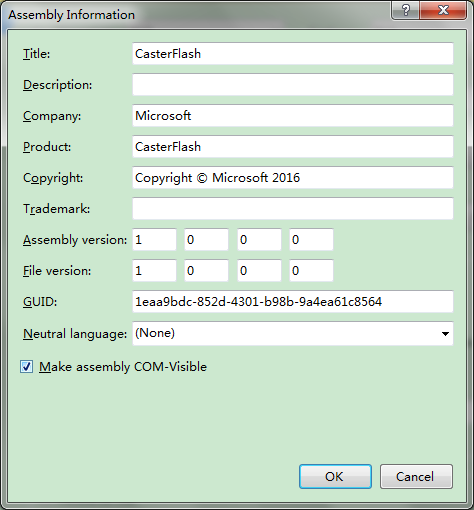
**Details:**

**Step 1**. Create a Class Library (or WPF project which must change the Output type to Class Library later).

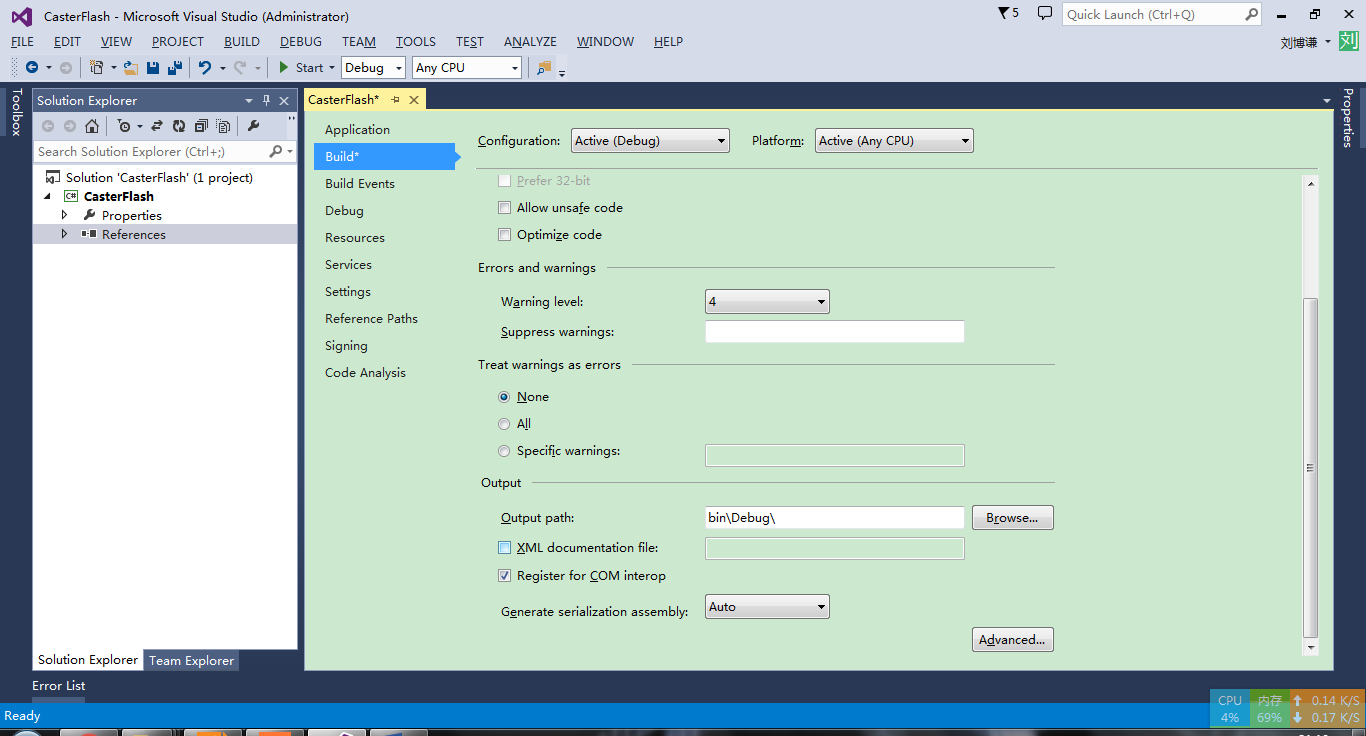


**Step 2.** Open Properties

In Application tab, Click Assembly Information, there is a checkbox named “Make assembly COM-Visible”, select it. If GUID is empty, create a new GUID, use Tools—Create GUID.



In Build tab, select Register for COM interop.



**Step 3.** Add References, add CasterUnitBase.dll and CAPE-OPENv1-1-0 to your project. CasterUnitBase is strong-named, so you can put it into GAC.

**Step 4.** Create calculator interface

To make this unit callable to other unit, it’s a good habit to declare a unique interface describing available calculation types. If you want this calculator expose to COM, remember to add ComVisible and Guid attributes.

**Step 5.** Create calculator class

Add a new class, inherit Calculator, and the interface you just defined. Just leave it here, we’ll deal with this class later.

**Step 6.** Create unit operation class

Add a new class, I like to make it the same name with this project.

1. Inherit CasterUnitOperationBase
2. Make this class public
3. Add some Attributes, the most import is three Attributes

[ComVisible(true)]

[Guid("8FFE8B95-FFAB-4032-B652-181AB5206E36")]

[CapeName("CasterFlash")]

Guid and CapeName here should be unique, use Tools—Create GUID to create one.

And some other attributes are also available, like CapeDescription, they all start with Cape, enter Cape to see other options, CapeCategory is not necessary, but useful if you want to put your unit into a particular category.

1. Implement constructor, must be non-parameter constructor for COM compatibility. In most case, just need to call the base constructor. Instantiate(new) the calculator we just made as the first parameter.
2. Three method you must implement, InitParameters, InitPorts, InitResults. Inside these methods, you should add parameters and ports to fields Parameters, Ports and Results. Normally, Parameters contains parameters which mode is CAPE\_INPUT or CAPE\_INPUT\_OUTPUT, and Results contains CAPE\_OUTPUT.
3. There are some other action you may want to customize.

Open your own window : OpenEditWindow

Change Ports in runtime : ports

Validate parameters : Validate

The other method works fine in most cases, like Save or Initialize, but you can always override them for customization

**Step 7.** Complete Calculator

1. Implement BeforeCalculate, in this method, you should get parameters and materials from Unit.Parameters and Unit.Ports, according to CAPEOPEN, the input material port must be duplicated before calculate. You may want to add some fields to make it easier to access parameters.
2. Implement Calculate, that’s where your intelligence should take over.
3. Implement OutputResult, put the result into Unit.Results, and set material to Unit.Ports. Then the calculation is done.

If you have some confusion, you can have a look at the example code CasterFlash.

**Deployment**

To deploy the unit operation, you need to run regasm xxx.dll, then run

regasm xxx.dll /tlb xxx.tlb /codebase

**Something you might concern:**

1 If your block support multi ports, under AspenPlus, you can open the parameter window and return, the port will be refreshed. If your unit has multi ports which is loaded automatically, Aspen can’t load the auto-added port, but it goes normal under another environment.

2 Please use Phases enum to represent phase, because different softwares may have different names. If the phase you need is not in the enum, you can create a Phases instance like this : new Phases("Liquid"). Besides Phases, there are plenty enums you can use, like UnitCategoryEnum, PropertyBasis and PropertyCategory, use them can save your time and reduce errors. CapeOptionParameter can use enum to initialize, and can compare or assign to enum.

3 If you don't override OpenWditWindow, there will be a default parameter window. The unit is obtained in Units class, in Parameter tab, when you change the unit, CAPE\_INPUT parameters won't change their values, CAPE\_INPUT\_OUTPUT parameters will change with unit with their SI value remains the same.

4 If save or load throw an exception, check if Parameters, Ports and Results contains something without [Serializable] or something is generic type which is not allowed both in COM and persist.

5 CAPEOPEN collection start with 1, but in C# we normally use 0, so if you are going to use raw ICapeCollection interface, remember to add 1 on index, but in CapeCollection methods, don't add 1. In normal situations, you should use methods in CapeCollection and start with 0.