# My Experiences with the AmsterCHEM COBIA Class Wizard

**Peter Nellen**

# Introduction

Name: Peter Nellen
Country: The Netherlands
Current position: In between jobs
Worked for Shell: 1980 -2016, the last 14 yrs in Gas Processing

Interest in CAPE-OPEN:

Within Shell we developed, together with Jasper van Baten, CAPE-OPEN wrappers for our legacy Fortran programs for Gas Treating with Amines. A column model and a property package were made CAPE-OPEN compliant.

Why COBIA:

Jasper made me aware of the COBIA developments, I told him that I am an expert in finding bugs in new software. To get access to COBIA I was supposed to be a member of CO-LaN, so I became an individual associate.

# AmsterCHEM COBIA Class Wizard

- What is the AmsterCHEM COBIA Class Wizard for Visual Studio
- COBIA code generation by the wizard
- Experiences
- Remarks
- Demo COBIA Class wizard for Visual Studio (*if time permits*)

# AmsterChem COBIA Class Wizard

- The AmsterCHEM COBIA Class Wizard (COBIA  Class Wizard) is an add-in for Visual Studio, to help develop a COBIA PMC object, e.g. unit operation.
- The COBIA Class Wizard is based on the COBIA Code Generation Interface.
- The COBIA Class Wizards generates classes and the definitions for all the functions in the classes. The content of the functions is not generated.

- The COBIA Wizard does NOT generate ready to run units!

# Code generation by the COBIA wizard

**To start a project from scratch**

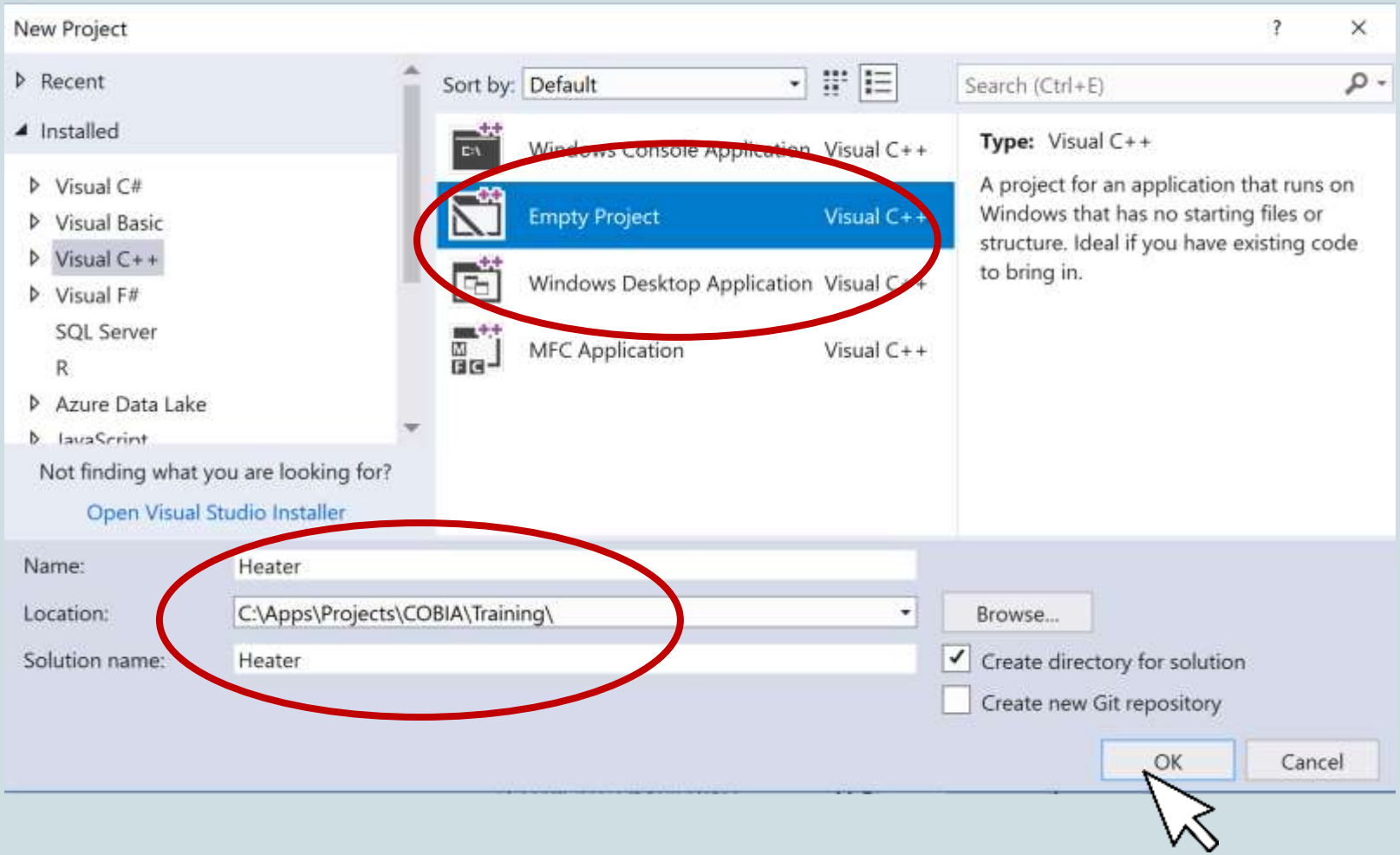
**Requirements:**

- The COBIA SDK
- Visual Studio
- COBIA Class Wizard
- Set configuration type to DLL
- add the SDK Include folder to the C/C++ Directories: $(COBIA_Include)

The COBIA Class Wizard is accompanied with documentation for a step by step walk through for a unit operation. This one was used for my experience.

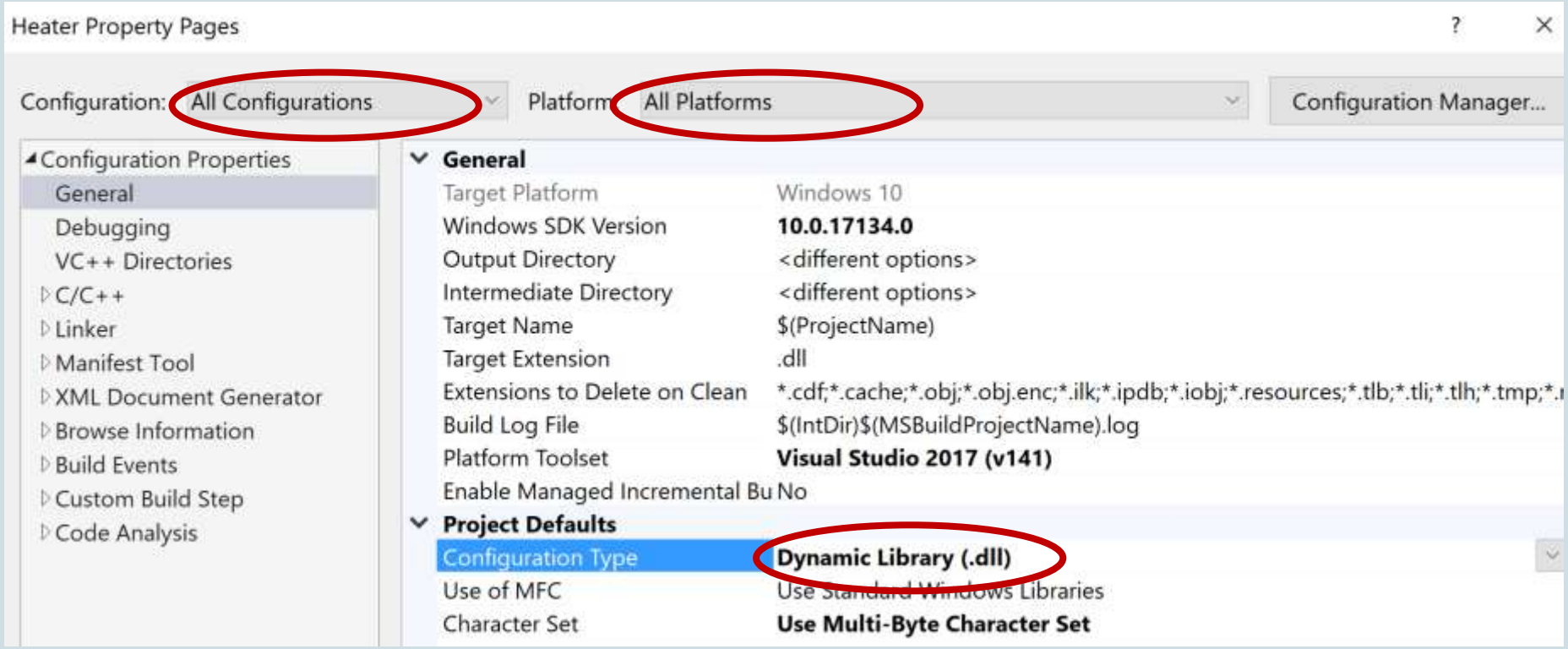
# Code generation by the COBIA wizard

## In Visual Studio start an empty C++ project



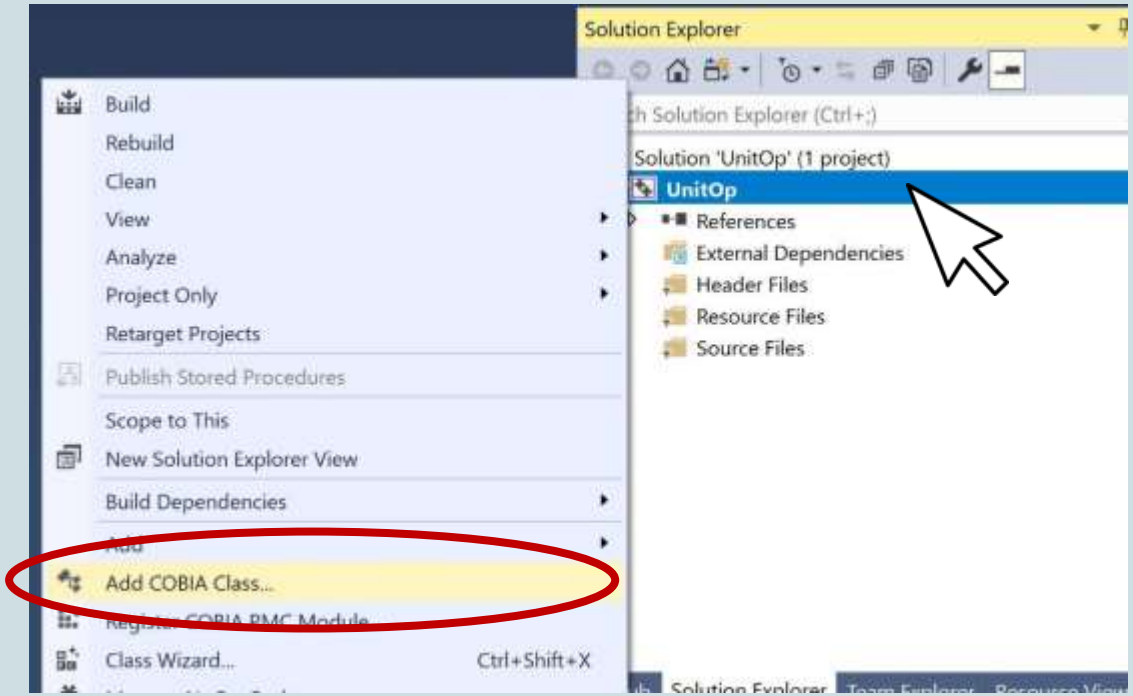
# Code generation by the COBIA wizard

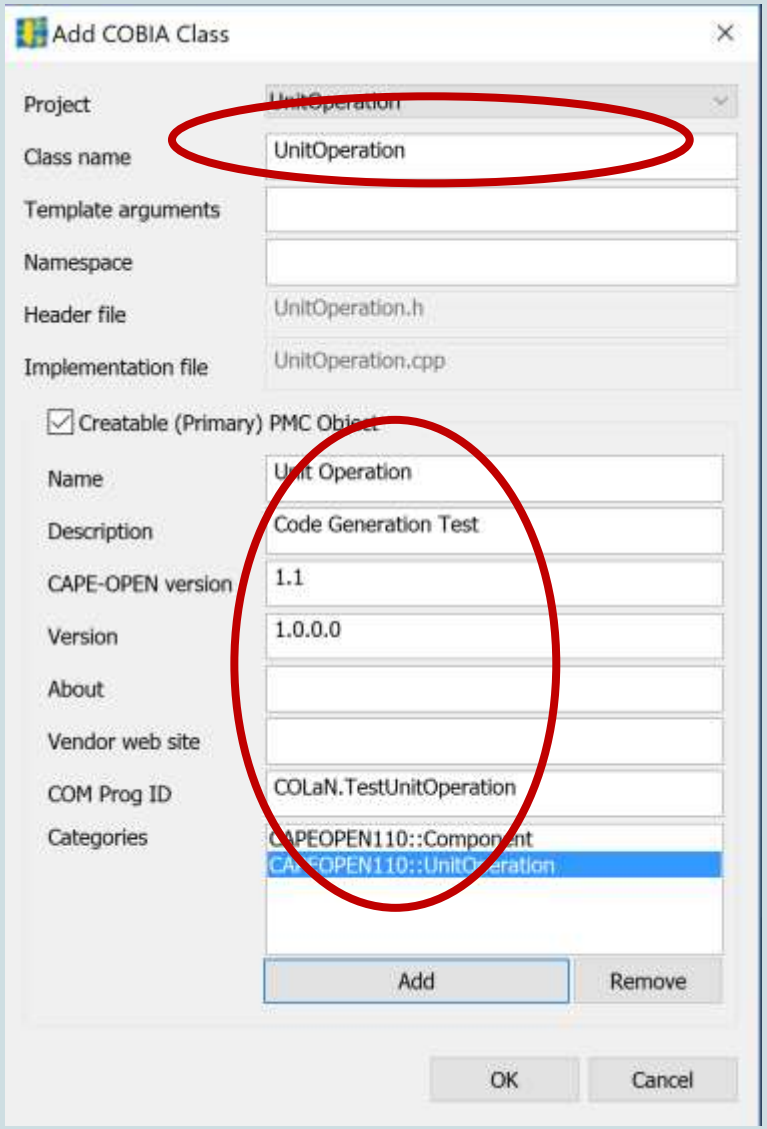
In the project properties set configuration type to DLL




Configuration DLL is required to give the PME access to the library

# CREATE A PMC Object



# CREATE A PMC Object

This will generate the following files:

COBIAEntryPoints.cpp

Holds the interface to the entry points and registration setting

UnitOperation.h

Header file for the UnitOperation

UnitOperation.cpp

Source file for the UnitOperation

# COBIAEntryPoints.cpp

```
#include <COBIA.h>
//this definition prior to COBIA_PMC.h ensures the entry points are
//created. Do this only in one compilation unit
#define COBIA_PMC_ENTRY_POINTS
//this definition prior to COBIA_PMC.h ensures the a default
//DllMain entry point is created (if COBIA_PMC_ENTRY_POINTS
//and _WIN32 are defined)
#define COBIA_PMC_DEFAULT_DLLMAIN
//PMC entry points are defined here:
#include <COBIA_PMC.h>
```

# COBIAEntryPoints.cpp

```
//! Define registration scope
/*!
PMC module must implement this function to indicate whether object
registration must be for all users or for the current user.

Alternatively, define either PMC_REGISTERFORALLUSERS or
PMC_REGISTERFORCURRENTUSER prior to including COBIA_PMC.h,
if COBIA_PMC_ENTRY_POINTS is defined

   return true if registration is for all users,
false if registration is for current user only
*/
bool isPMCRegistrationForAllUsers() {
  return false; //TODO: modify
}
```

# UnitOperation.h

**The wizard will insert the following code in UnitOperation.h**

```
#pragma once
#include <COBIA.h>
using namespace COBIA;
class UnitOperation :
public CapeOpenObject<UnitOperation> {

  CapeStringImpl name;
public:

  const CapeStringImpl getDescriptionForErrorSource() {
    return COBIATEXT("UnitOperation ") + name;
  }
```

# UnitOperation.h

```
//Registration info
static const CapeUUID getObjectUUID() {
  //Class UUID = DF2AD0C7-09EE-45C5-8AC3-CC1246D664D7
  return CapeUUID{{0xdf,0x2a,0xd0,0xc7,0x09,0xee,0x45,
       0xc5,0x8a,0xc3,0xcc,0x12,0x46,0xd6,0x64,0xd7}};
}
static void Register(CapePMCRegistrar registrar) {
  registrar.putName(COBIATEXT("Unit Operation"));
  registrar.putDescription(COBIATEXT("Code generation test"));
  registrar.putCapeVersion(COBIATEXT("1.1"));
  registrar.putComponentVersion(COBIATEXT("1.0.0.0"));
  registrar.putProgId(COBIATEXT("COLAN.TestUnitOperation"));
  registrar.addCatID(CAPEOPEN110::categoryId_Component);
  registrar.addCatID(CAPEOPEN110::categoryId_UnitOperation);
}
```

# UnitOperation.cpp

```
#include "UnitOperation.h"
#include <COBIA_PMC.h>

COBIA_PMC_REGISTER(UnitOperation);
```

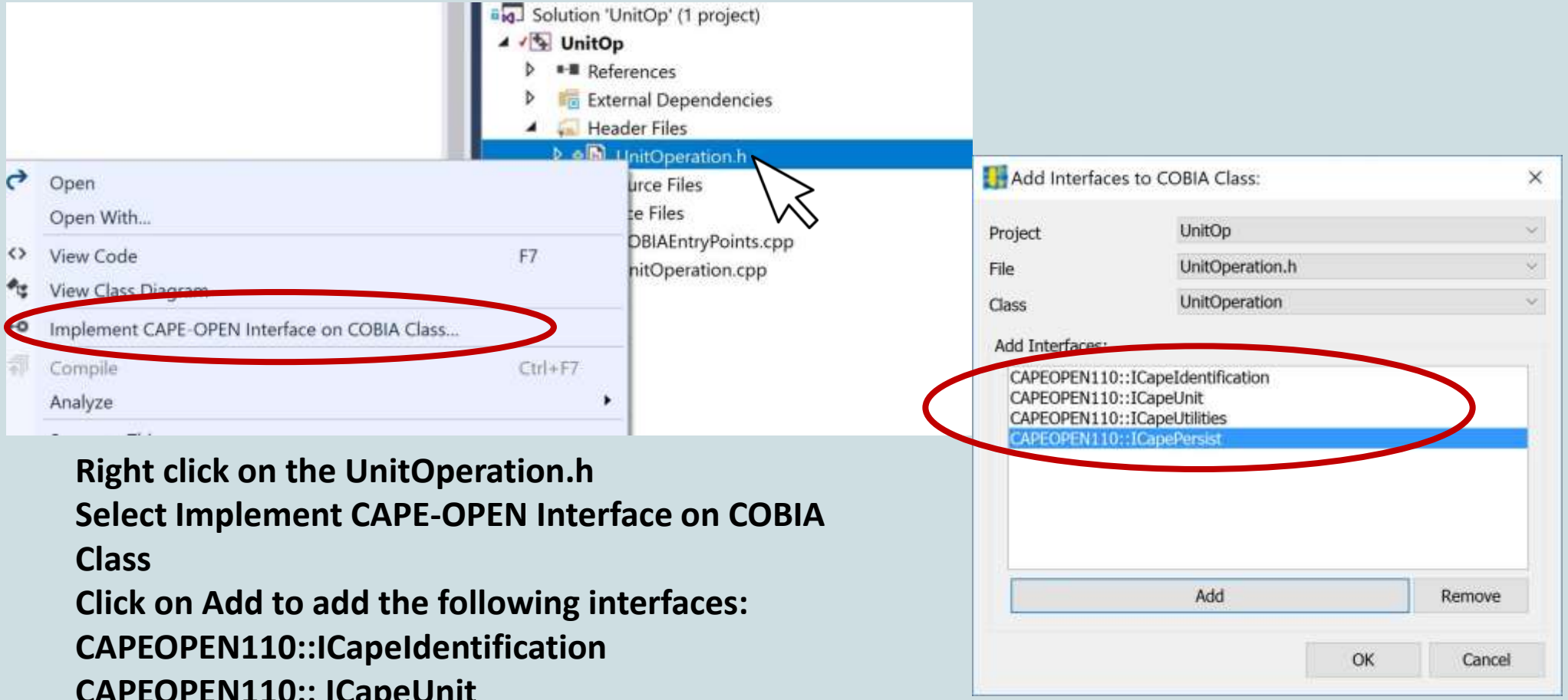
# UnitOperation Interfaces

**The wizard can be used to add interfaces to the UnitOperation:**

e.g.

- ICapeIdentification
- ICapeUnit
- ICapeUtilities
- ICapePersist

# Implement Interfaces




**Right click on the UnitOperation.h**
**Select Implement CAPE-OPEN Interface on COBIA Class**
**Click on Add to add the following interfaces:**
**CAPEOPEN110::ICapeIdentification**
**CAPEOPEN110:: ICapeUnit**
**CAPEOPEN110:: ICapeUtilities**
**CAPEOPEN110:: ICapePersist**

# UnitOperation Interfaces

**The wizard will insert the following code in UnitOperation.h**

```
//CAPEOPEN110::ICapeIdentification
void getComponentName(/*out*/ CapeString name) {
  #error getComponentName must be implemented
}
void putComponentName(/*in*/ CapeString name) {
  #error putComponentName must be implemented
}
void getComponentDescription(/*out*/ CapeString desc) {
  #error getComponentDescription must be implemented
}
void putComponentDescription(/*in*/ CapeString desc) {
  #error putComponentDescription must be implemented
}
```

# UnitOperation Interfaces

**The code in UnitOperation.h needs to be updated to:**

```
void getComponentName(/*out*/ CapeString name) {
  name = this->name;
}
void putComponentName(/*in*/ CapeString name) {
  this->name = name;
  dirty = true;
}
```

# UnitOperation

**To complete the Unit Operation classes for port collection and unit port needs to be added.**

**These classes have their own Interfaces and Methods.**

# My Experiences

Following the guideline of the COBIA Class wizard it was relative easy to create a unit operation with a port collection.
Running it was a bit more difficult:

- An issue with the base of the collection
- An issue with persistence

Both were quickly resolved by Jasper
So the Unit Operation did run.

Extending the Unit Operation was more difficult, e.g. adding parameters
Documentation is still a bit scarce, so what interfaces were required for the parameters?? Jasper gave some guidelines, but the parameters did not work as expected. The parameter mode was wrong and an issue with validate.
These were also solved by Jasper, so parameters are now supported as well.

# My Experiences cont'd

Next on my wish list was an energy port.
This one failed as well; an issue with ParameterSpec not implemented.
This is also fixed by Jasper, but on only very recent, so not yet tested.

# Remarks

The COBIA framework makes developing a PMC more efficient; the available adapter classes are easy to use.

The AmsterCHEM COBIA Class wizard is a powerful tool, but it only generates a skeleton and only for the Interfaces selected. It would be nice if there was some more documentation for COBIA.

A number of (small) errors were encountered during this exercise, so maybe it would be an idea if there was a test set within CO-LaN that should be run before releasing a new version of the software.