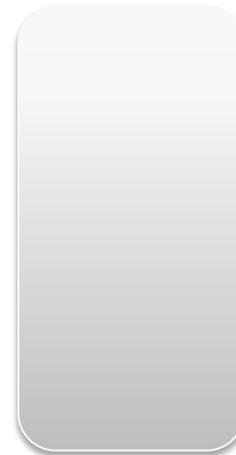


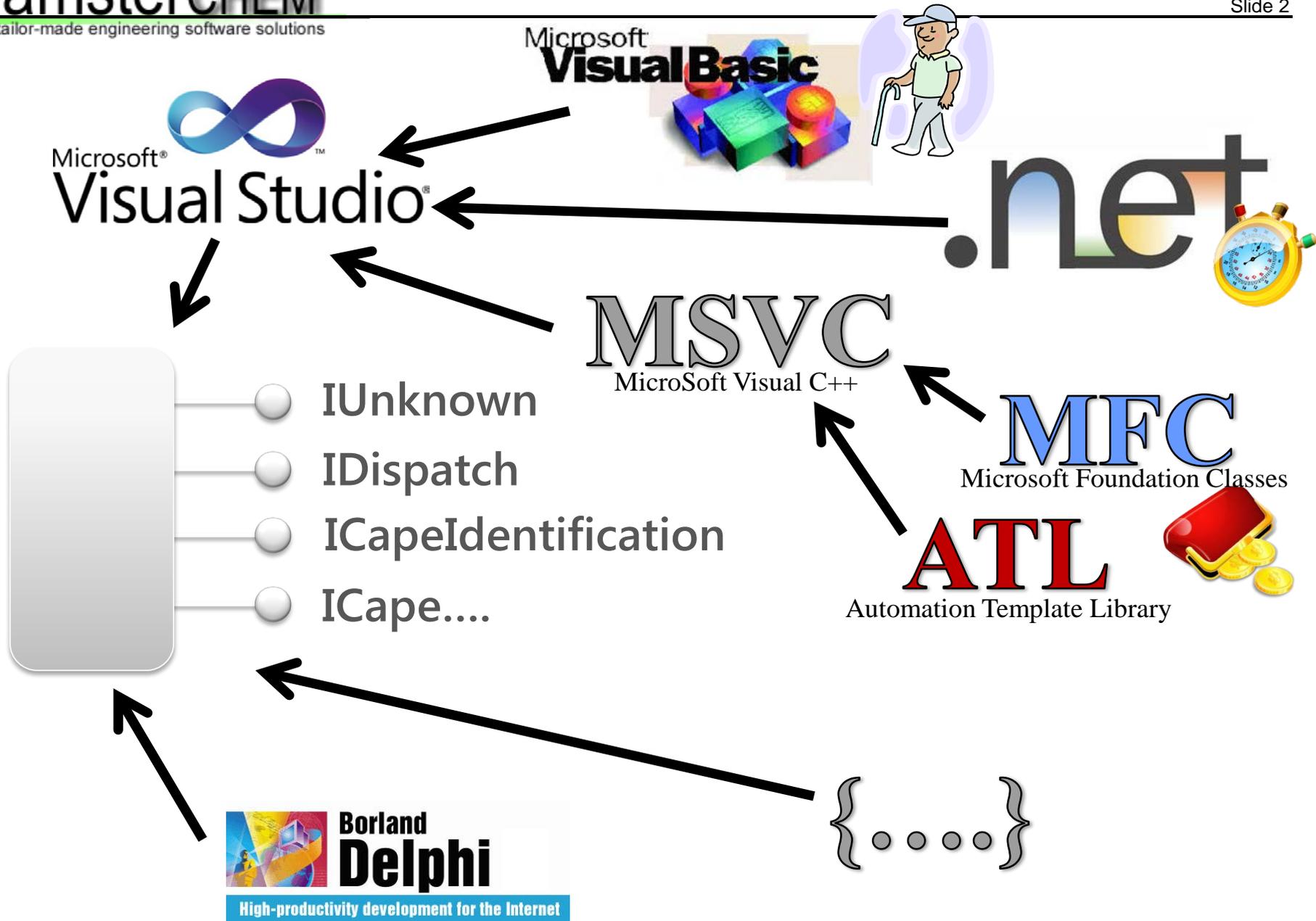
# COM AND CAPE-OPEN (WITHOUT FRINGES)

Jasper van Baten, AmsterCHEM



- IUnknown
- ~~● IDispatch~~
- ICapeIdentification
- ICape....

A.k.a. the boring stuff





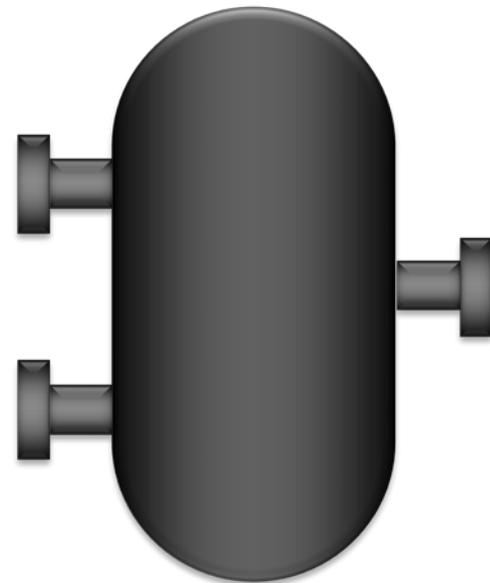
- IUnknown
- IDispatch
- ICapeIdentification
- ICape....

## C++

- MSVC
- Intel C++
- MinGW
- Cygwin
- (Borland C++)
- (OpenWatcom)
- ...



```
class UnitOperation {  
    ParameterCollection parameterCollection;  
};
```

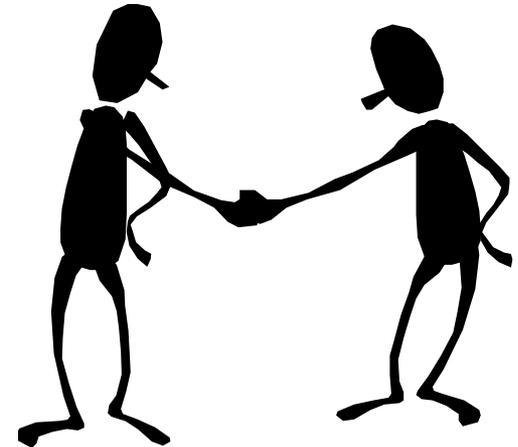


```
class ParameterCollection :  
public ICapeIdentification,  
public ICapeCollection {  
  
public:  
  
    //IUnknown  
  
    //IDispatch  
  
    //ICapeIdentification  
  
    //ICapeCollection  
  
};
```



```
//IUnknown
```

```
HRESULT _stdcall QueryInterface(const IID &iid, void **iface) {  
    HRESULT res=NO_ERROR;  
    if (iid==IID_IUnknown) {  
        *iface=(IUnknown*)(ICapeCollection*)this;  
    } else if (iid==IID_IDispatch) {  
        *iface=(IDispatch*)(ICapeCollection*)this;  
    } else if (iid==IID_ICapeIdentification) {  
        *iface=(ICapeIdentification*)this;  
    } else if (iid==IID_ICapeCollection) {  
        *iface=(ICapeCollection*)this;  
    } else {  
        *iface=NULL;  
        res=E_NOINTERFACE;  
    }  
    return res;  
}
```



```
//IUnknown
```

```
ULONG _stdcall AddRef() {  
    return 1;  
}
```

```
ULONG _stdcall Release() {  
    return 1;  
}
```



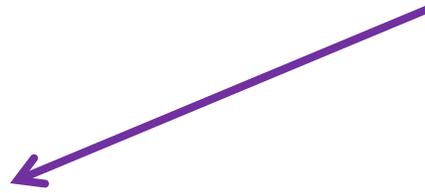
```
//IUnknown
```

```
int count=1;
```

```
ULONG _stdcall AddRef() {  
    return ++count;  
}
```

```
ULONG _stdcall Release() {  
    if ((--count)==0) delete this;  
    return count;  
}
```

Also at QueryInterface



## Early binding



- **Type is known (from TLB or IDL)**  
TLB = Type LiBrary  
IDL = Interface Definition Language
- **Type obtained via QueryInterface**
- **Call via VTABLE**
- **Parameters as in type specification**

## Late binding



- **No type import**
- **Calls made via IDispatch**
- **GetIDsOfNames: function name to ID (integer value)**
- **Invoke: use function ID**

**All parameters must be**

**VARIANTS**

- **All CAPE-OPEN interfaces inherit from IDispatch**
- **CAPE-OPEN requires multiple interfaces on each object**
- **Microsoft advises against multiple Dispatch implementations on a single object**
- **ATL does not support it:**

“Although it is possible to expose multiple dual interfaces on a single COM object, it is not recommended. If there are multiple dual interfaces, there must be only one **IDispatch** interface exposed. The techniques available to ensure that this is the case carry penalties such as loss of function or increased code complexity. The developer considering this approach should carefully weigh the advantages and disadvantages”.

```
interface ICapeIdentification : IDispatch {
    [id(0x00000001), propget, helpstring("property ComponentName")]
    HRESULT ComponentName([out, retval] BSTR* name);
    [id(0x00000001), propput, helpstring("property ComponentName")]
    HRESULT ComponentName([in] BSTR name);
    [id(0x00000002), propget, helpstring("property ComponentDescription")]
    HRESULT ComponentDescription([out, retval] BSTR* desc);
    [id(0x00000002), propput, helpstring("property ComponentDescription")]
    HRESULT ComponentDescription([in] BSTR desc);
};

[
    odl,
    uuid(678C099A-0093-11D2-A67D-00105A42887F),
    helpstring("ICapeCollection Interface"),
    dual,
    oleautomation
]
interface ICapeCollection : IDispatch {
    [id(0x00000001), helpstring("gets an item specified by index or name")]
    HRESULT Item(
        [in] VARIANT id,
        [out, retval] IDispatch** Item);
    [id(0x00000002), helpstring("Number of items in the collection")]
    HRESULT Count([out, retval] long* itemCount);
};
```

```
//IDispatch
```

```
HRESULT _stdcall GetTypeInfoCount(UINT *count) {  
    if (!count) return E_POINTER;  
    *count=0;  
    return NO_ERROR;  
}
```

```
HRESULT _stdcall GetTypeInfo(UINT,LCID,ITypeInfo **) {  
    return E_UNEXPECTED;  
}
```

```
HRESULT _stdcall GetIDsOfNames(const IID &,LPOLESTR  
*,UINT,LCID,DISPID *) {  
    return E_UNEXPECTED;  
}
```

```
HRESULT _stdcall Invoke(DISPID,const IID &,LCID,WORD,DISPPARAMS  
*,VARIANT *,EXCEPINFO *,UINT *) {  
    return E_UNEXPECTED;  
}
```

```
//ICapeIdentification
```

```
HRESULT _stdcall get_ComponentName(/*[out, retval]*/BSTR *name) {  
    if (!name) return E_POINTER;  
    *name=SysAllocString(L"ParameterCollection");  
    return NO_ERROR;  
}  
  
HRESULT _stdcall put_ComponentName(/*[in]*/ BSTR name) {  
    return E_NOTIMPL;  
}  
  
HRESULT _stdcall get_ComponentDescription(/*[out, retval]*/BSTR *desc) {  
    if (!desc) return E_POINTER;  
    *desc=SysAllocString(L"Parameter Collection Example");  
    return NO_ERROR;  
}  
  
HRESULT _stdcall put_ComponentDescription(/*[in]*/ BSTR desc) {  
    return E_NOTIMPL;  
}
```



```
//ICapeCollection
```

```
HRESULT _stdcall Item(/*[out, retval]*/VARIANT, IDispatch **item) {  
    .....  
}
```

```
HRESULT _stdcall Count(/*[out, retval]*/long *count) {  
    .....  
}
```

## Remaining requirements

- **In-proces COM server: DLL**
- **Class factory for PMC primary object:  
Implements IClassFactory:**

```
HRESULT CreateInstance(LPUNKNOWN, REFIID, void **);  
HRESULT LockServer(BOOL incr);
```

- **COM entry points:**

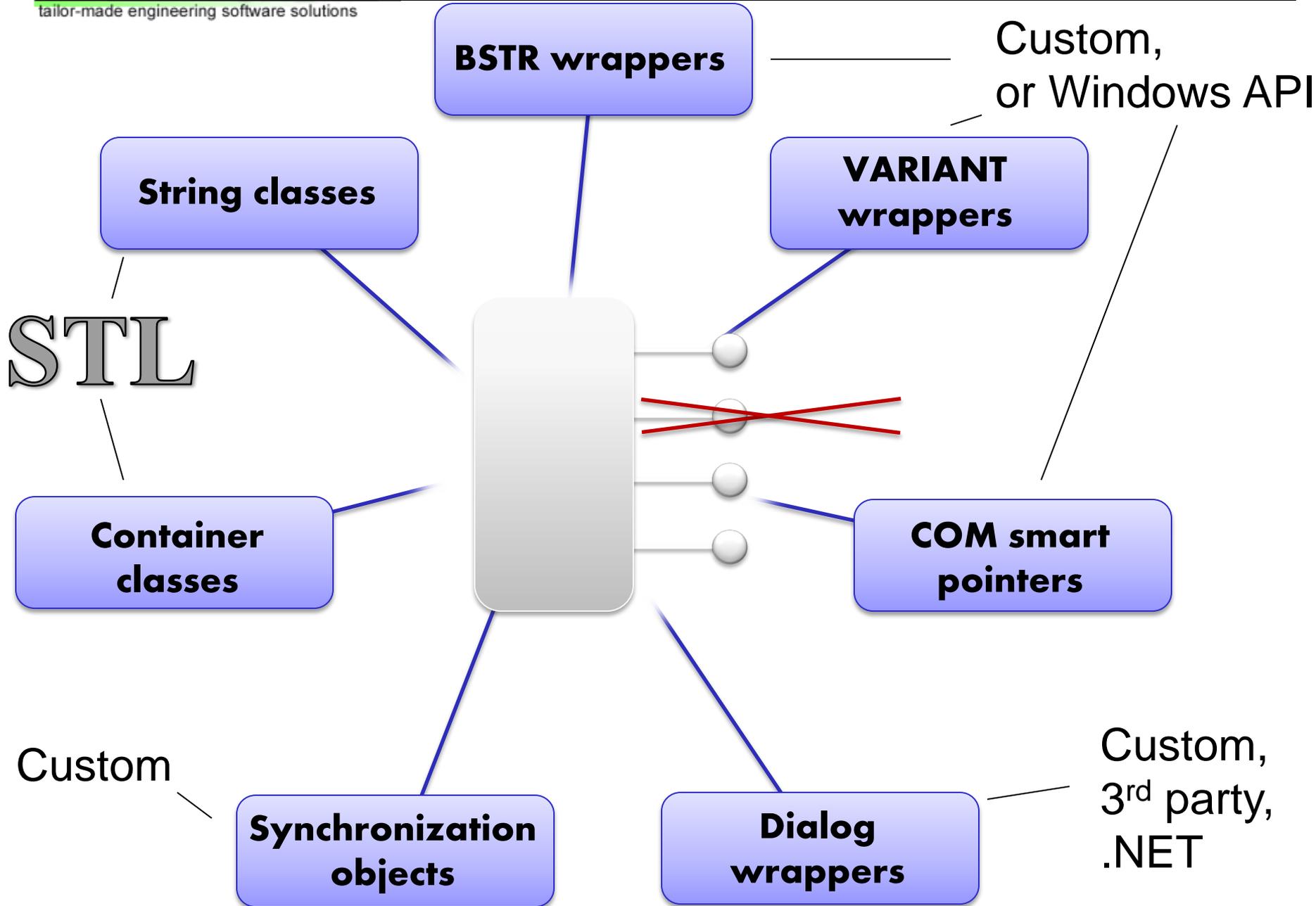
```
HRESULT DllGetClassObject(REFCLSID, REFIID, LPVOID* );  
HRESULT DllCanUnloadNow();  
HRESULT DllRegisterServer();  
HRESULT DllUnregisterServer();
```

**See water source code on [cocosimulator.org](http://cocosimulator.org)**

## Module lock count

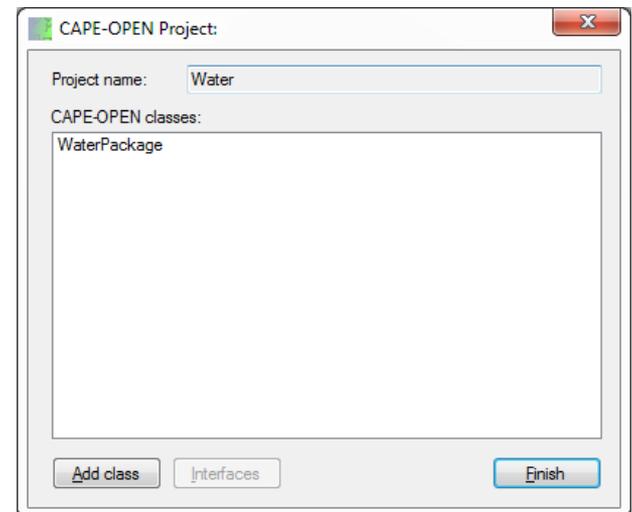
- **Increase while creating an object, decrease while destroying an object**
- **IClassFactory::LockServer()**
- **Return at DllCanUnloadNow()**
- **Thread safe:**
  - **InterlockIncrement()**
  - **InterlockDecrement()**





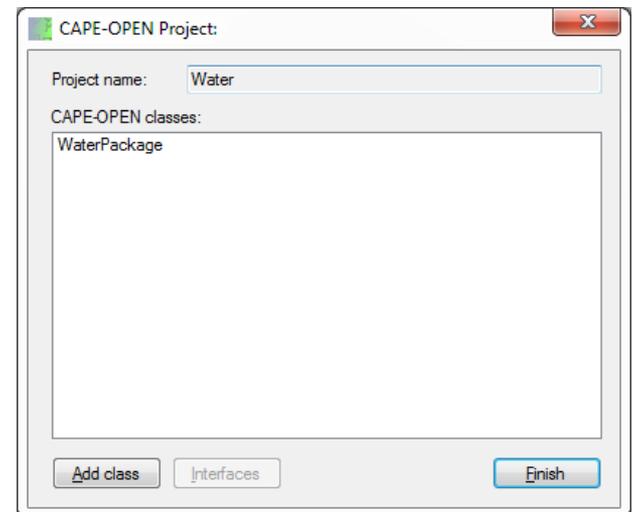
## Wizard is useful for:

- **All mundane work**
  - **Generate DLL framework**
  - **All base classes**
  - **Class factory**
  - **Basic COM object**
  - **Basic CAPE-OPEN object**
  - **CAPE-OPEN type lib**



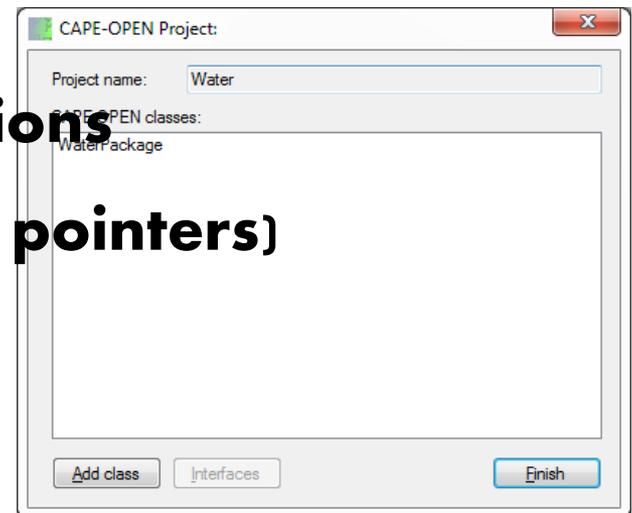
## Wizard is useful for:

- **Keeping track of registry entries**
  - **COM server**
  - **COM objects**
  - **CAPE-OPEN categories**
  - **CapeDescription keys**
  - **TLB registration usually not required**



## Wizard is useful for:

- **Generation of classes**
  - **Keep track of registry entries**
  - **Implement interfaces**
  - **Update QueryInterface**
  - **Generate all function headers**
    - **Including argument annotations**
- **Basic argument checking (NULL pointers)**
- **Exception handling**



## AmsterCHEM's new CAPE-OPEN wizard

- **Customer driven: no Visual Studio requirement**
- **Builds with a variety of C++ compilers**  
(including MinGW, Cygwin, MSCV Express  )
- **Zero external dependencies**
- **Lean code, without fringes, low footprint, fast**
- **Provides insights on middle ware requirements**

**Satisfaction all around...**

**Q&A**