ATL

An introduction to

Active Template Library

Agenda

- ATL
  - Why
  - What
  - How
- Smart Pointers

Hvorfor ATL

- Implementeringen af DLL-indpakningen er stort set ens for alle COM-servere, og der er kun en lille forskel i DllGetClasObject.
- Alle COM-objekter skal implementere IUnknown (counted pointer idom + typecast).

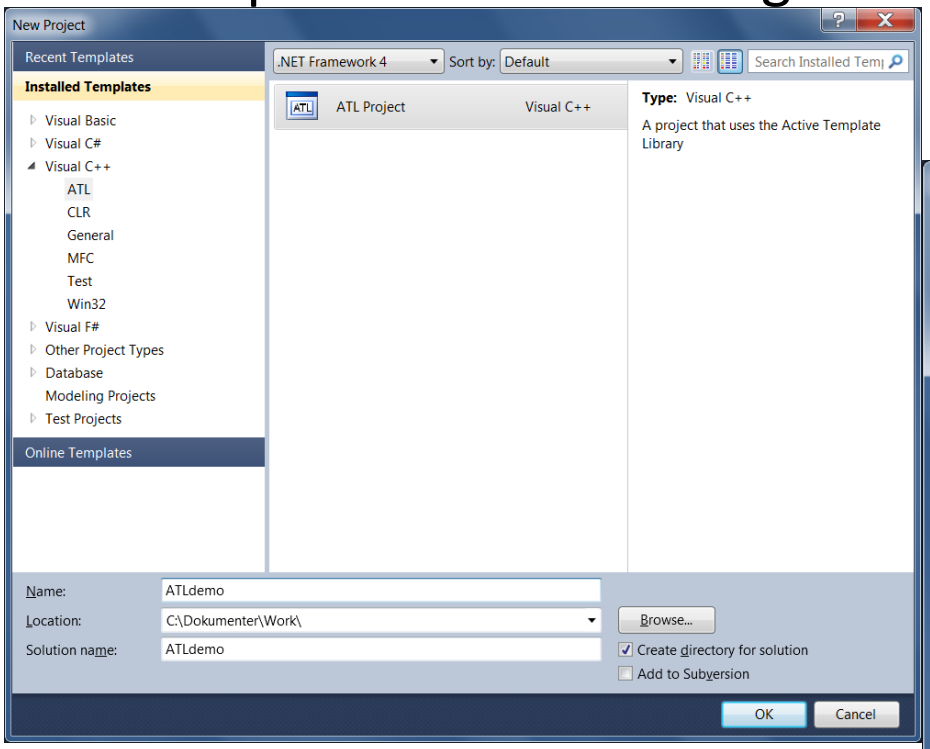
- → Få en wizard til at lave den trivielle slaveprogrammering!

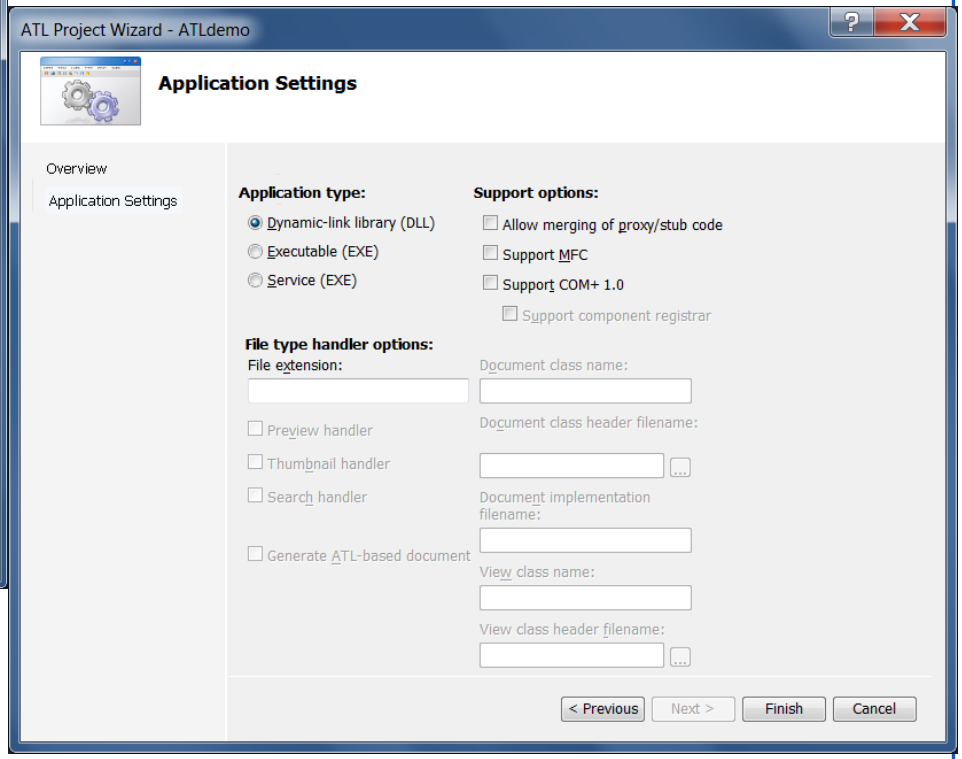
Hvad er ATL?

- ATL er et ”magisk” værktøj, der gør det let at udvikle COM-servere.
- ATL er ”blot” nogle header filer, et par CPP-filer og et par Wizards.
- **ATL er en samling template klasser** og template funktioner, nogle hjælpe klasser og makroer samt et par Wizards i Visual Studio.

How to create a COM-server with ATL

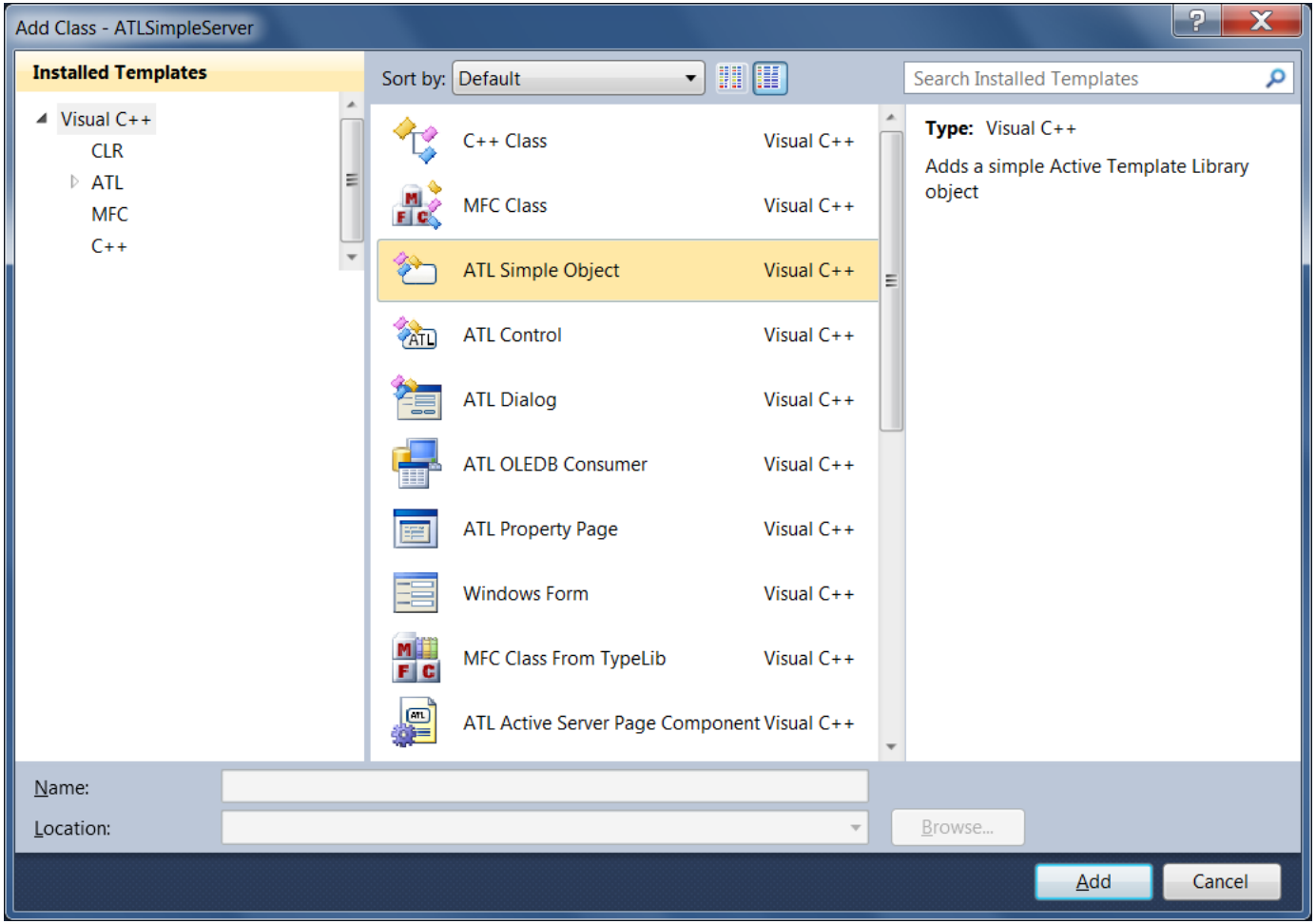
1. Create a new Visual C++ ATL project
2. Accept the default settings – click Finish.





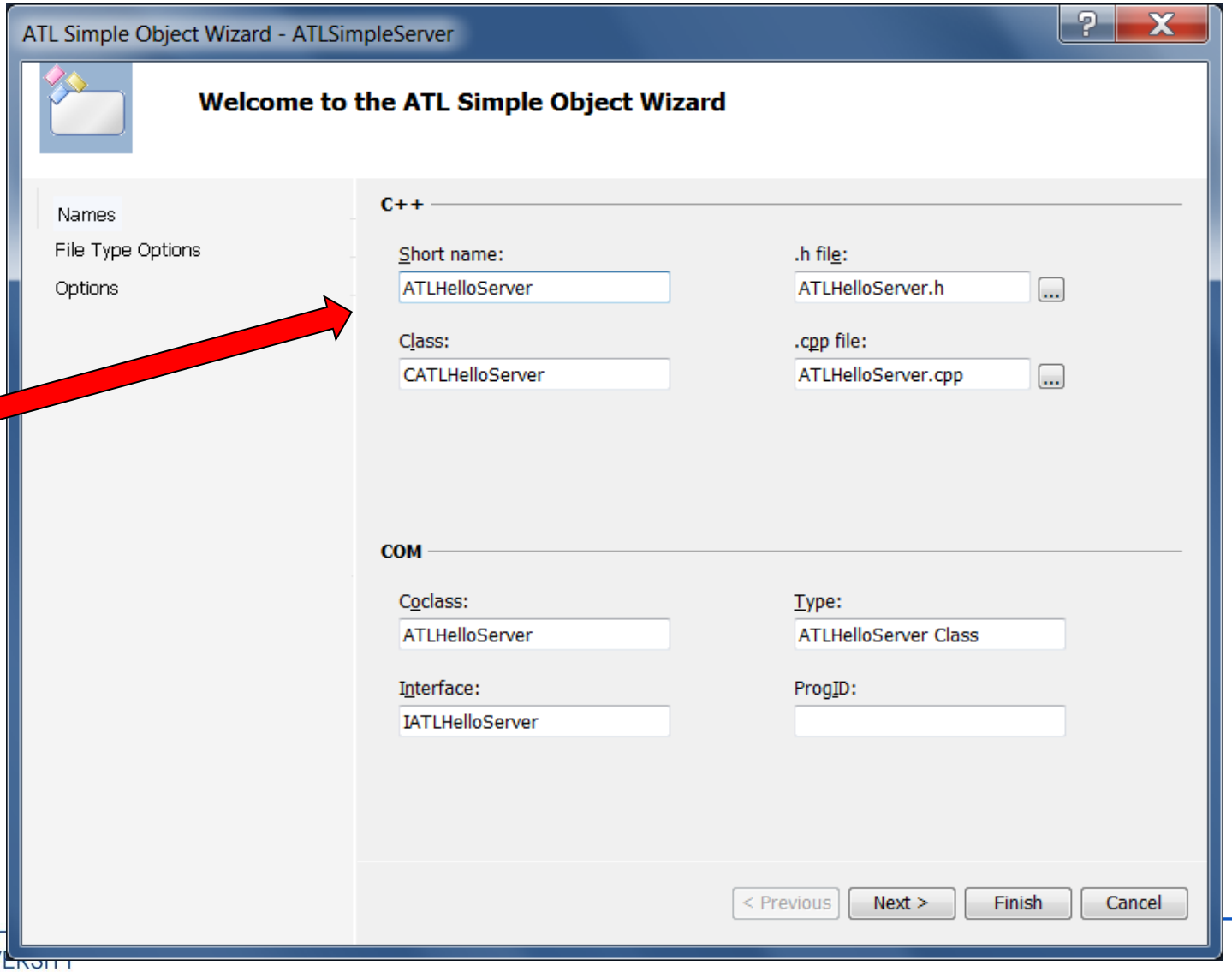
Adding a CoClass to the COM Server

- Right click on the project and select Add Class
  - Select ATL Simple Object



Enter the Name of the CoClass

- Enter short name and then click Next



File Type Handler Options

- The default options are suitable for many projects - Click Next

ATL Simple Object Wizard - ATLSimpleServer

File Type Handler Options

Names
File Type Options
Options

Add file type (preview/thumbnail/search handler) support
Preview Handler
Thumbnail Handler
Search Handler
File Extension:

ATL-based Document

Generic document class name:
.h of generic document:
Generate ATL-based document
.cpp of generic document:

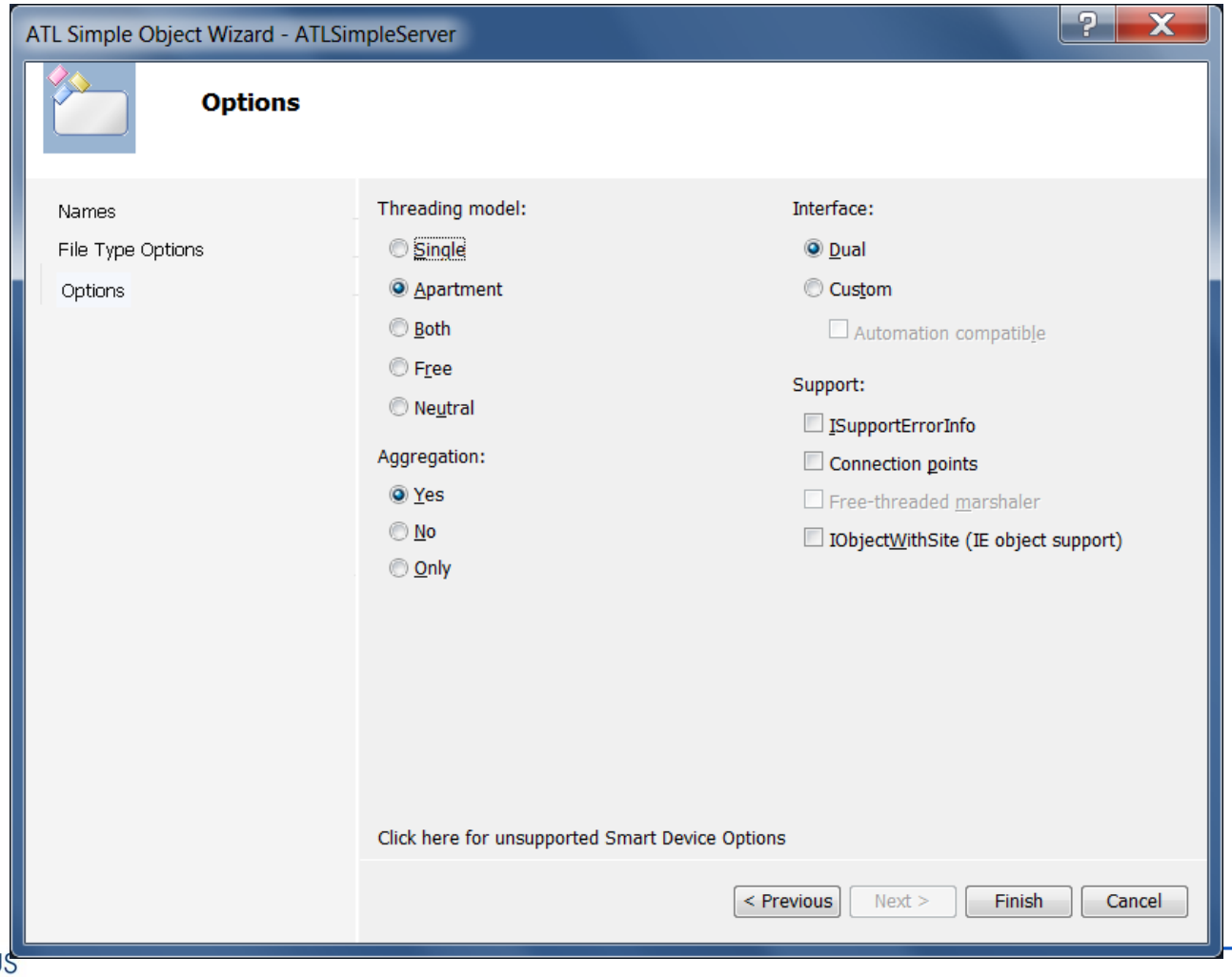
MFC-based Document (and View)

Document class name:
.h of MFC document:
View class name:
.h of view:

< Previous | Next > | Finish | Cancel

Select the Wanted Options

- The default options are suitable for many projects – Click Next



Add Methods and Properties

- When the CoClass has been added to the project you can switch to **Class View** and **right click on the interface** and select **Add Method** or **Add Property**…

ATLdemo - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Tools NDepend Architecture Test Analyze Window Help

Debug | Win32 | Loading device-room

ATLHelloSer.h*

(Global Scope)

```
class ATL_NO_VTABLE CATLHelloSer :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CATLHelloSer, &CLSID_ATLHelloSer>,
    public IDispatchImpl<IATLHelloSer, &IID_IATLHelloSer, &LIBID_ATLSimpleServerLib, /*wMajor =*/ 1, /*wMinor =*/ 0>
{
public:
    CATLHelloSer()
    {
    }

DECLARE_REGISTRY_RESOURCEID(IDR_ATLHELLOSER)


BEGIN_COM_MAP(CATLHelloSer)
    COM_INTERFACE_ENTRY(IATLHelloSer)
    COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()



    DECLARE_PROTECT_FINAL_CONSTRUCT()

    HRESULT FinalConstruct()
    {
        return S_OK;
    }

    void FinalRelease()
    {
    }

public:
```

Class View

<Search>

- Maps
- Global Functions and Vari
- Macros and Constants
- ATLSimpleServerLib
- CATLHelloSer
- CATLSimpleServerModul
- IATLHelloSer
- SimpleServerPS

View Class Diagram
Go To Definition F12
Go To Declaration Ctrl+F12
Browse Definition
Find All References Ctrl+K, R
Filter To Type
Add
Class Wizard... Ctrl+Shift+X
Subversion
Copy Ctrl+C
Sort Alphabetically
Sort By Object Type
Sort By Object Access
Group By Object Type
Properties Alt+Enter

Add Method...
Add Property...

Attributes

(Name) IATLHelloSer

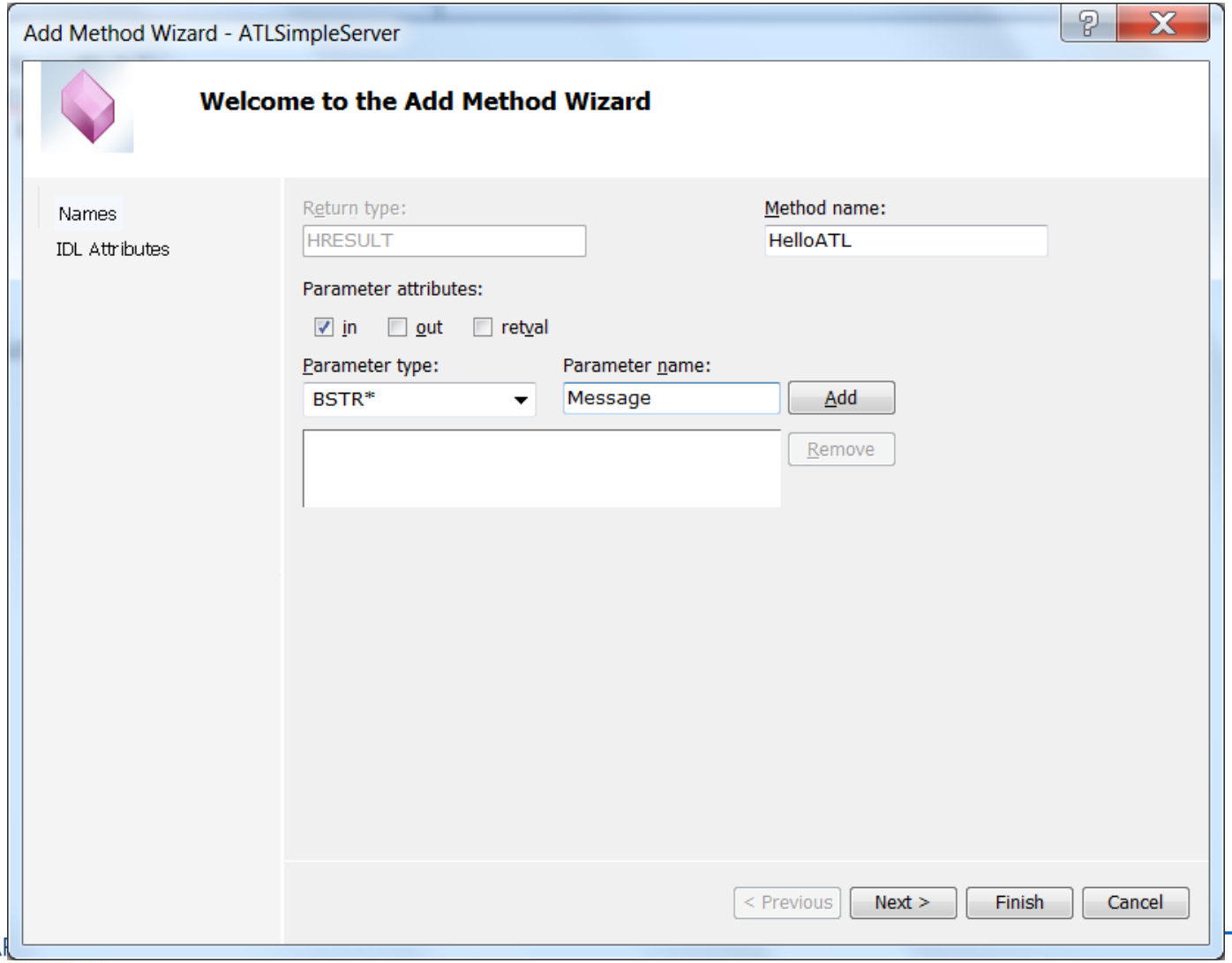
Output | Pending Changes

Ready

17:14 19-09-2010

Specify Signature

- Give the method a name and add parameters



Implement Functionality

- And finally you click on the method in the class and write the implementation of the method…

ATLdemo - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Tools NDepend Architecture Test Analyze Window Help

Debug | Win32 | Loading device-room

ATLHelloSer.cpp* | ATLHelloSer.h*

CATLHelloSer | HelloATL(void)

```
// ATLHelloSer.cpp : Implementation of CATLHelloSer

#include "stdafx.h"
#include "ATLHelloSer.h"


// CATLHelloSer



STDMETHODIMP CATLHelloSer::HelloATL(void)
{
    // TODO: Add your implementation code here

    return S_OK;
}
```

Class View

<Search>

- Maps
- Global Functions and Vari
- Macros and Constants
- ATLSimpleServerLib
- CATLHelloSer
- CATLSimpleServerModul
- IATLHelloSer
- ATLSimpleServerPS

- CATLHelloSer()
- FinalConstruct()
- FinalRelease()
- HelloATL(void)

Click here

Solut... Tea... Class...

Properties

CATLHelloSer VCCodeClass

(Name)	CATLHelloSer
File	c:\dokumenter\k
FullName	CATLHelloSer
IsAbstract	False
IsInjected	False
IsManaged	False
IsSealed	False

C++

100 %

Output | Pending Changes

Ready

ATL Class Overview

Classes in the Active Template Library (ATL) can be categorized as follows:

Class Factories	Data Types	MMC Snap-In	String and Text
Class Information	Debugging and Exception	Object Safety	Tear-Off Interfaces
Collection	Dual Interfaces	Persistence	Thread Pooling
COM Modules	Enumerators and Collections	Properties and Property Pages	Threading Models and Critical Sections
Composite Controls	Error Information	Registry Support	UI Support
Connection Points	File Handling	Running Objects	Windows Support
Control Containment	Interface Pointers	Security	Utility
Controls: General Support	IUnknown Implementation	Service Provider Support	
Data Transfer	Memory Management	Site Information	

Classes Shared Between MFC and ATL

Classes Shared Between MFC and ATL (*These utility classes can be used in any native C++ project*) :

Class	Description	Header file
CFileTime	Provides methods for managing the date and time values associated with a file.	atltime.h
CFileTimeSpan	Provides methods for managing relative date and time values associated with a file.	atltime.h
CFixedStringT	Represents a string object with a fixed character buffer.	cstringt.h
CImage	Provides enhanced bitmap support, including the ability to load and save images.	atlimage.h
COleDateTime	Encapsulates the DATE data type used in OLE automation.	atlcomtime.h
COleDateTimeSpan	Represents a relative time, a time span.	atlcomtime.h
CPoint	A class similar to the Windows POINT structure.	atltypes.h
CRect	A class similar to a Windows RECT.	atltypes.h
CSimpleStringT	Represents a CSimpleStringT object.	atlsimpstr.h
CSize	A class similar to the Windows SIZE structure.	atltypes.h
CStrBufT	Provides automatic resource cleanup for GetBuffer and ReleaseBuffer calls on CStringT.	atlsimpstr.h
CStringData	Represents the data of a string object.	atlsimpstr.h
CStringT	Represents a CStringT object.	atlstr.h
CTime	Represents an absolute time and date.	atltime.h
CTimeSpan	An amount of time, which is internally stored as the number of seconds in the time span.	atltime.h
IAtlStringMgr	Represents the interface to a CStringT memory manager.	atlsimpstr.h

Smart Pointers

Getting smart 1

Veteran COM programmers are used to a standard pattern.

1. You call a function or method that returns an interface pointer,
2. Use the interface pointer for some scope of time,
3. and then you release it.

Here's what the pattern looks like in source code:

```
void f(void) {
  IUnknown *pUnk = 0;
  // call
  HRESULT hr = GetSomeObject(&pUnk);
  if (SUCCEEDED(hr)) {
    // use
    UseSomeObject(pUnk);
    // release
    pUnk->Release();
    }
  }
```

Getting smart 2

If a function needs to acquire three interface pointers before doing any actual work, that means three call statements before the first use statement:

```
void f(void) {
  IUnknown *rgpUnk[3];
  HRESULT hr = GetObject(rgpUnk);
  if (SUCCEEDED(hr)) {
    hr = GetObject(rgpUnk + 1);
    if (SUCCEEDED(hr)) {
      hr = GetObject(rgpUnk + 2);
      if (SUCCEEDED(hr)) {
        UseObjects(rgpUnk[0], rgpUnk[1], rgpUnk[2]);
        rgpUnk[2]->Release();
        }
      rgpUnk[1]->Release();
      }
    rgpUnk[0]->Release();
    }
  }
```

Code like this often motivates programmers to set their tab stops to one or two spaces or petition management for 24-inch monitors.

Getting smart 3

- The Standard C++ Library contains a class, auto_ptr, that is hardcoded to call delete on a pointer in its destructor (which is guaranteed to execute even in the face of exceptions).
- Analogously, ATL contains a COM-smart pointer, CComPtr, that correctly calls Release in its destructor.
- The CComPtr class implements the client side of the basic COM reference counting model.
- CComPtr has one data member, which is the raw COM interface pointer.
- The type of this pointer is passed in as a template parameter:

```
CComPtr<IUnknown> unk;
CComPtr<IClassFactory> cf;
CComPtr<IFoo> spFoo;
```

Getting smart 4

- The default constructor initializes the raw pointer data member to null.
- The smart pointer also has constructors that take either raw pointers or smart pointers of the same type as arguments.
- In both cases, the smart pointer calls AddRef to manage the reference.
- CComPtr's assignment operator works with either raw pointers or smart pointers, and automatically releases the held pointer prior to calling AddRef on the newly assigned pointer.
- The destructor for CComPtr releases the held interface if it is non-null.

```
void f(IUnknown *pUnk1, IUnknown *pUnk2) {
  // ctor calls addref on pUnk1 if non-null
  CComPtr<IUnknown> unk1(pUnk1);

  // ctor calls addref on unk1.p if non-null
  CComPtr<IUnknown> unk2 = unk1;

  // operator = calls release on unk1.p if non-null
  // and calls AddRef on unk2.p if non-null

  unk1 = unk2;
  // destructor releases unk1 and unk2 if non-null
}
```

Getting smart 5

- C++ has an import compiler directive:
  #import "..*pathcomserver*.tlb" no_namespace named_guids**

  – The #import directive instructs the compiler to process the designated type library, converting the contents to C++ code that describes the COM interfaces contained within the type library.

  – The most interesting aspect of the contents of these header files is their use of Microsoft's smart pointer template classes.

  – For more detailed information on the #import directive see: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomg/html/dcomtyplib.asp

Smart Pointers

- A smart pointer is an object that looks, acts, and feels like a normal pointer but offers greater functionality.

```
// Create a COM-server and get a smart
// interface pointer to IATLHelloServer
IATLHelloServerPtr helloPtr(__uuidof(ATLHelloServer));
```

The COM interface we want to call by.

the COM class' CLSID

Demo

- For details regarding use of ATL to program a COM server and using it from C++ programs study the demo project **`ATLdemo08`**.

HINT

- If you get a Project Build Error PRJ0050 when you build a COM server then do either:
  1. Run Visual Studio in elevated mode
  2. Use Per-user Redirection Registration.
     (a Linker setting in Visual Studio).
     - Per-user redirection allows you to register without having to run in elevated mode.
     - Per-user redirection will force any writes to HKCR to be redirected to HKEY_CURRENT_USER (HKCU).