Extend Python Using C++ and ArcObjects

Mark Cederholm
UniSource Energy Services

Using ArcObjects in Python

Esri Developer Summit 2010

ArcMap and Python: Closing the VBA Gap

Tuesday 4:30 Mesquite C

Extend Python Using C++ and ArcObjects

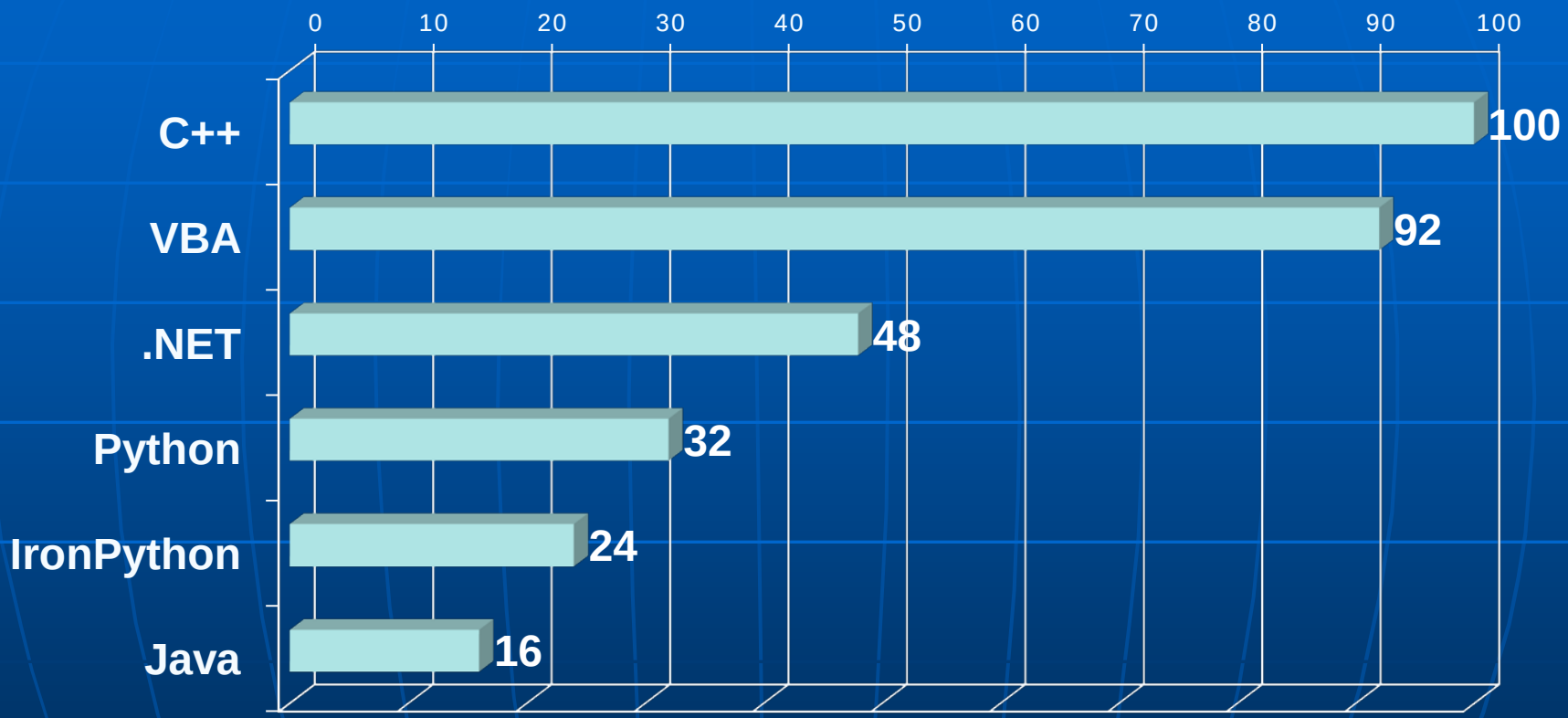
Wednesday 11:15 Mesquite B

Download presentations and code:

http://www.pierssen.com/arcgis10/python.htm

2:26:42 PM

COM Interop: relative speed

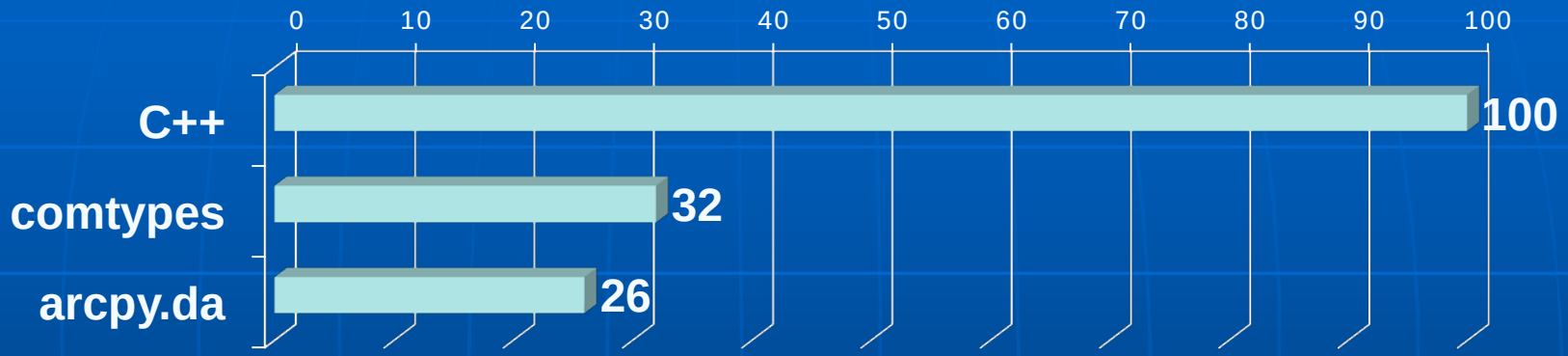


Benchmark = 500+K ShapeCopy operations

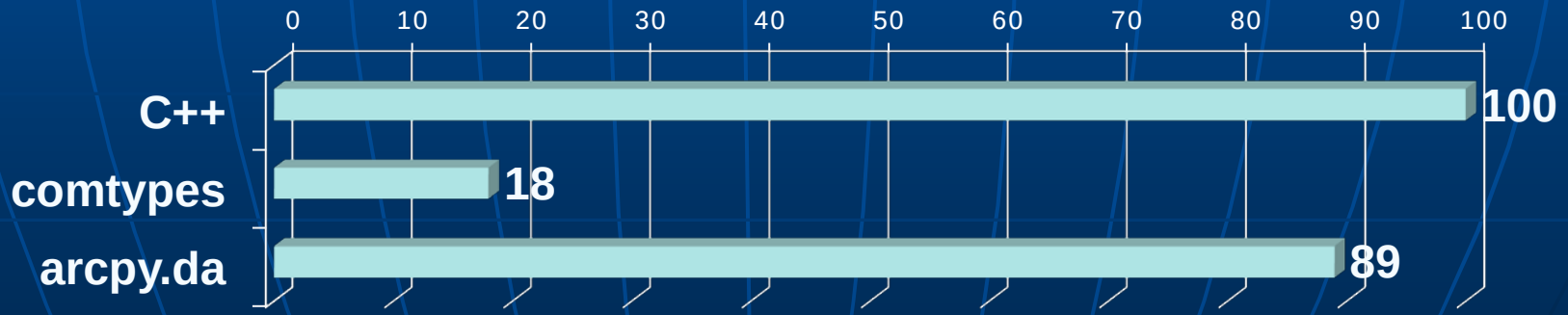
2:26:42 PM

Don’t forget arcpy.da (at 10.1)

Querying polygon geometry:



Querying SHAPE_AREA:



2:26:42 PM

Options to extend Python:

- COM/ATL + comtypes: No longer recommended
- Python/C API: Create C Python module (.pyd)

  SWIG, Cython: Automatically generate wrapper code to create a pyd
- ctypes: Create wrapper code on the Python side

Extending Python with C or C++:

http://docs.python.org/extending/extending.html

ArcObjects in Visual C++:

http://help.arcgis.com/en/sdk/10.0/arcobjects_net/conceptualhelp/index.html#/Appendix_ArcObjects_in_Visual_C/000100000n8q000000/

[All sample code and walkthroughs assume ArcGIS 10.1 pre-release with Python 2.7]

Free Visual C++ downloads:

- Visual C++ 2008 Express (required for Cython installation)

  http://www.microsoft.com/download/en/details.aspx?id=20682
- Visual C++ 2010 Express

  http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express
- Sample code and walkthroughs assume VC 2008E
- All walkthroughs tested successfully in VC 2010E (with minor modifications involving project variables)
- Express versions do not have ATL or MFC

Testing for memory leaks in VC Express:

```
#ifdef _DEBUG
        _CrtMemState s1, s2, s3;
        _CrtMemCheckpoint(&s1);
#endif
        char * sResult = MyFunction(sMyArg);
#ifdef _DEBUG
        _CrtMemCheckpoint(&s2);
        if (_CrtMemDifference(&s3, &s1, &s2))
        {
                OutputDebugString(L"Allocated blocks found:\n");
                _CrtMemDumpAllObjectsSince(&s1);
        }
#endif
```

In this example, only one block should be allocated

Walkthrough 1: Creating a Python module in Visual C++

[Sample Code: PythonC/standalonedemo]

2:26:42 PM

An example C++ class:

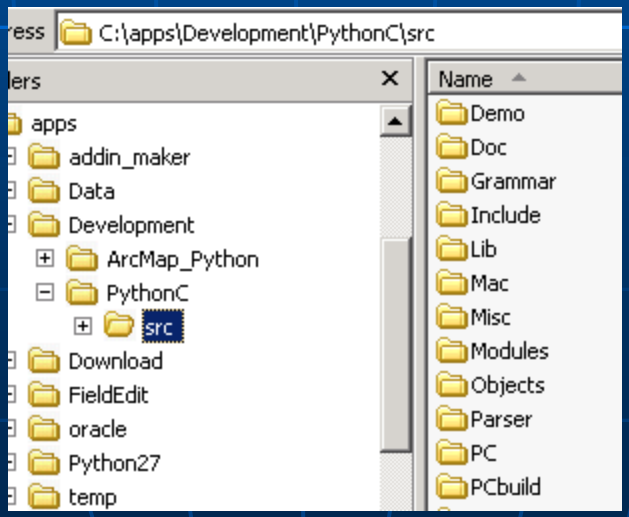
```
// demo.h
#include <string>
using namespace std;
class demo
{
private:
	bool _bInitialized;
	string _sResult;
public:
	demo();
	bool Init();
	char * Inventory(char * Workspace, char * FeatureClass);
private:
	void DoIt(char * ws, char * fc);
};
```

Step 1: Set up Python development package

- Download and unzip Python 2.7 source code package:

  http://www.python.org/ftp/python/2.7.2/Python-2.7.2.tgz

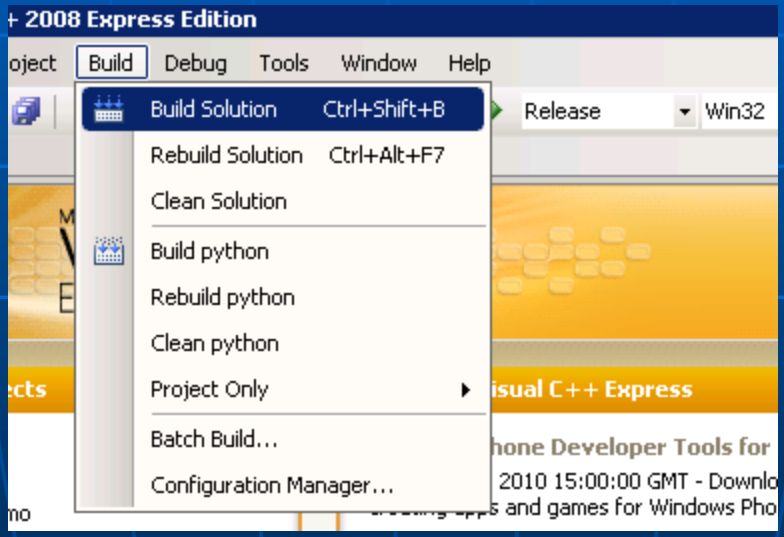
  [Download actually has a ".tar" extension which must be changed to ".tgz" for WinZip to handle it properly]
- Create a folder called "src" adjacent to where you intend to create Python extension projects
- Move contents of "Python-2.7.2" folder to src folder



2:26:42 PM

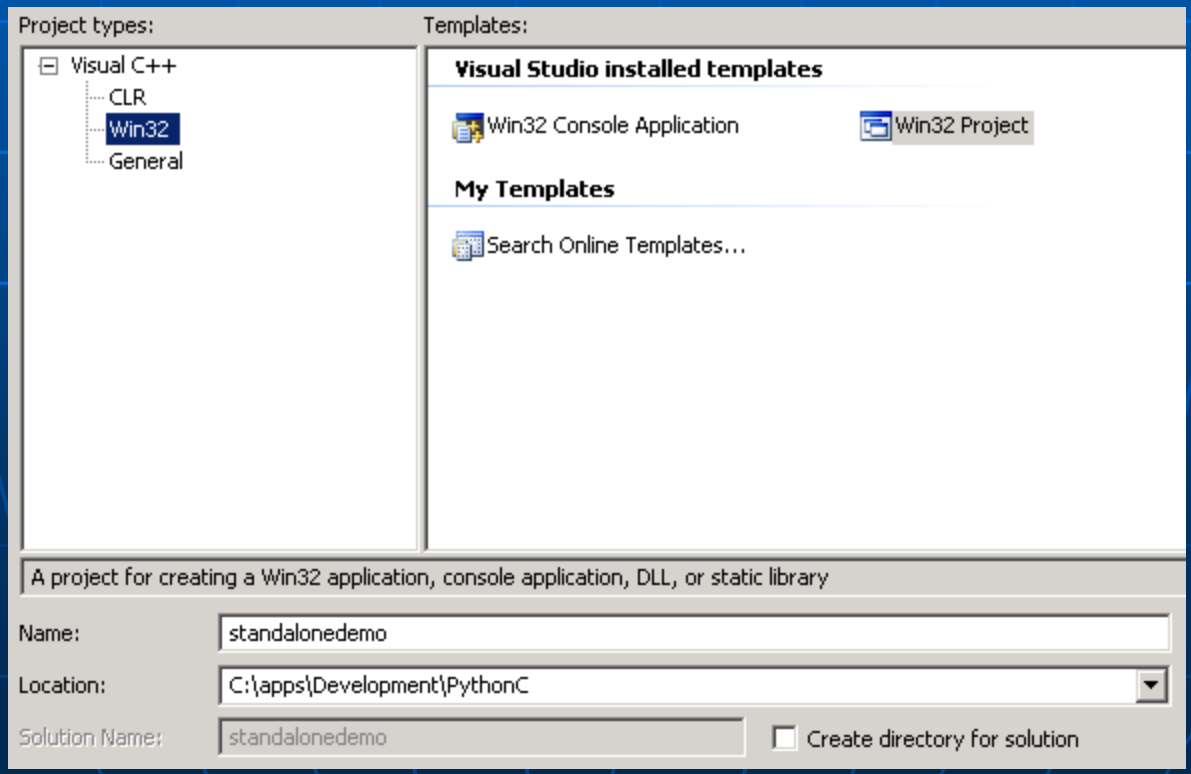
Step 1 (continued)

- Open src\PCbuild\pcbuild.sln in Visual C++
- Build solution in both “Debug” and “Release” configurations (ignore errors), creating python27_d.lib and python27.lib



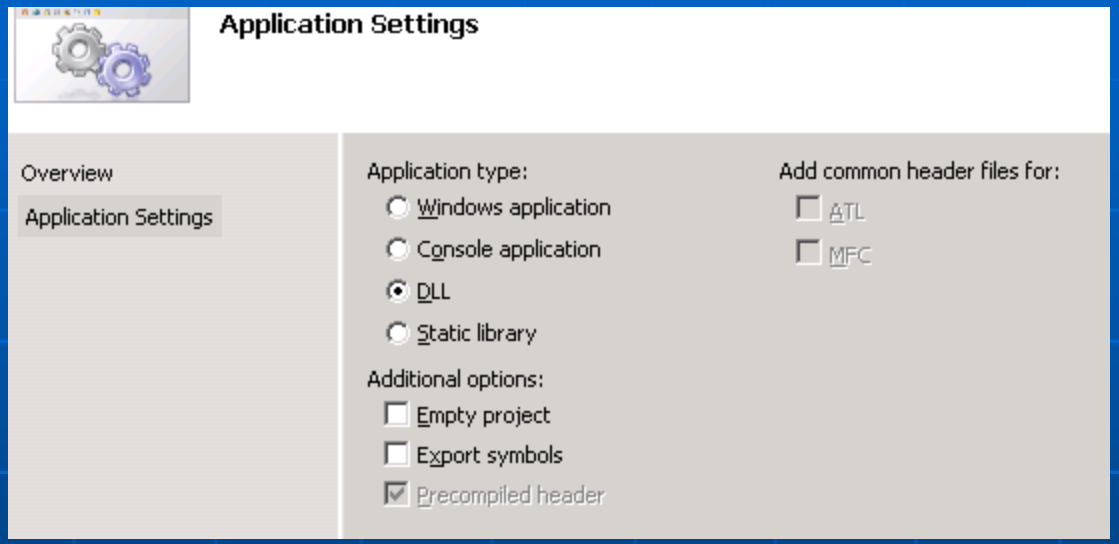
Step 2: Create a new Visual C++ project

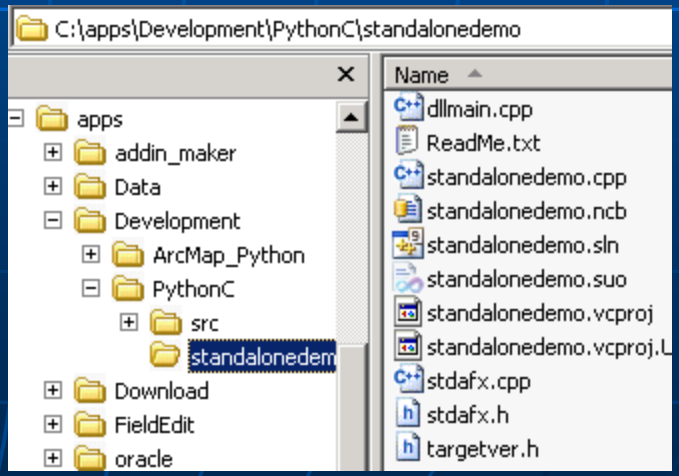
- Project type: Win32 Project
- Name: standalonedemo
- Uncheck “Create directory for solution”
- Location: desired project folder (same level as src)



Step 2 (continued)

- Application type: DLL

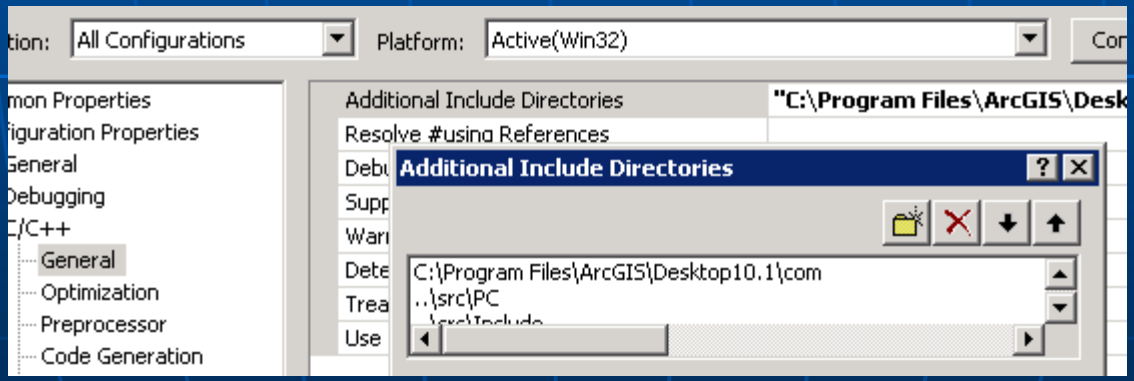




2:26:42 PM

Step 2 (continued)

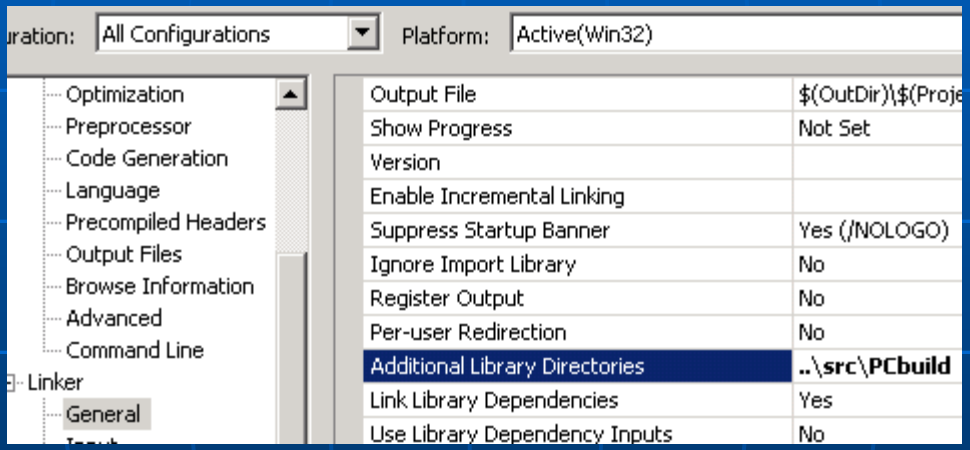
- Properties (All Configurations)
  C/C++|General|Additional Include Directories:
  ..\src\Include
  ..\src\PC
  [C:\Program Files*]\ArcGIS\Desktop10.1\com
  *wherever ArcGIS is installed



Step 2 (continued)

- Properties (All Configurations)

  Linker|General|Additional Library Directories: ..\src\PCbuild

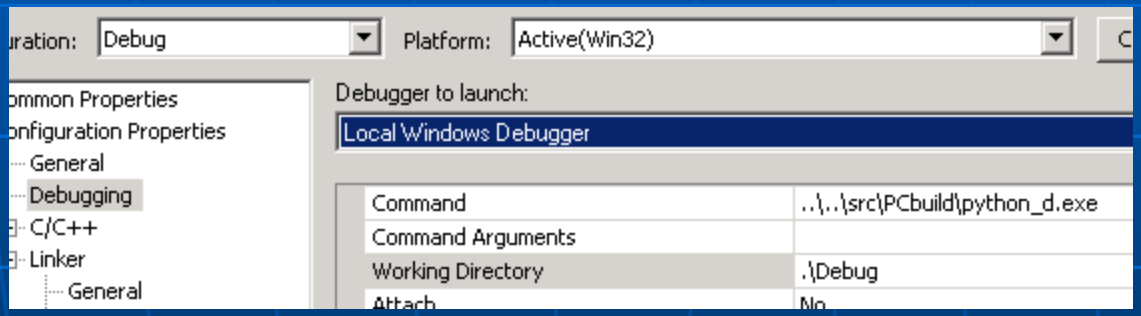


Step 2 (continued)

- Properties (Debug)

  Debugging|Command:
  ..\..\src\PCbuild\python_d.exe

  Debugging|Working Directory: .\Debug

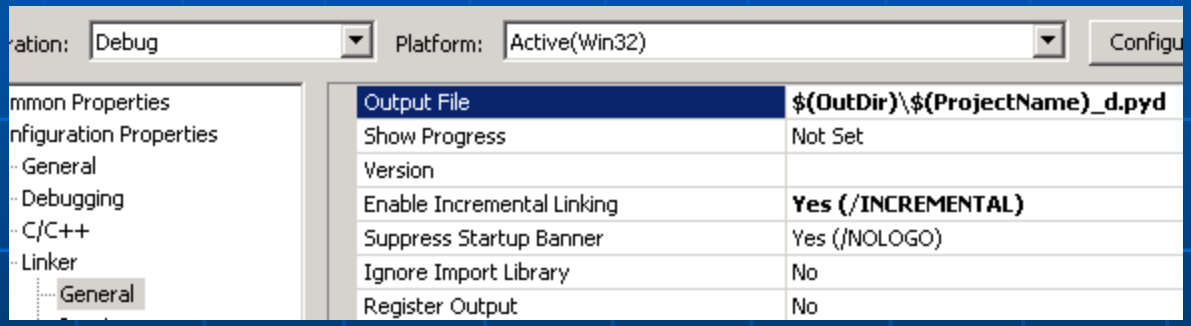


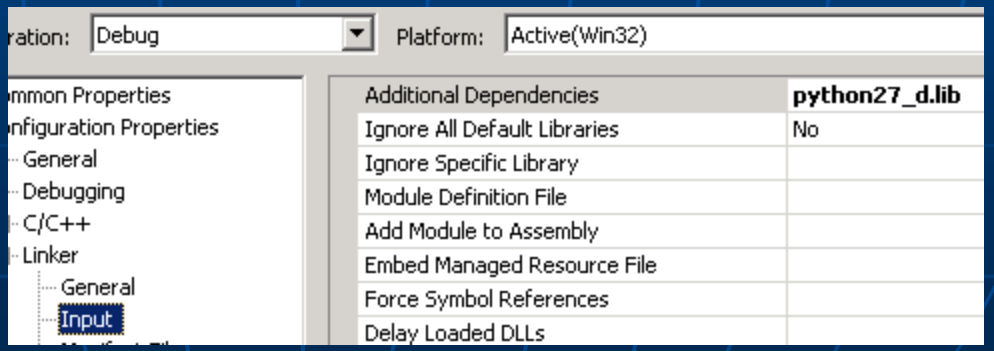
Step 2 (continued)

- Properties (Debug)

  Linker|General|Output File: $(OutDir)\$(ProjectName)_d.pyd

  Linker|Input|Additional Dependencies: python27_d.lib

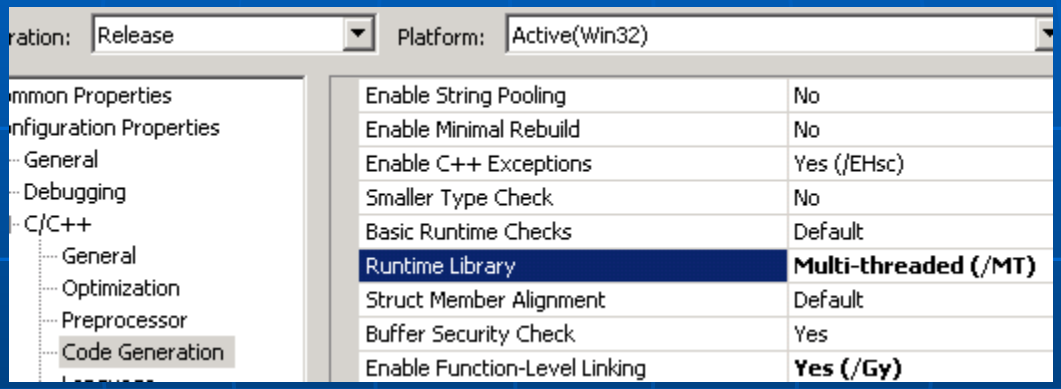




Step 2 (continued)

- Properties (Release)

  C/C++|Code Generation|Runtime Library: Multi-threaded (/MT)

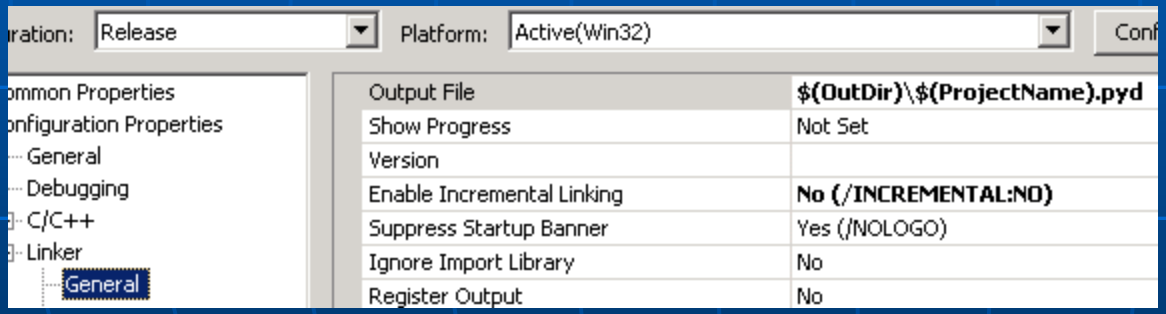


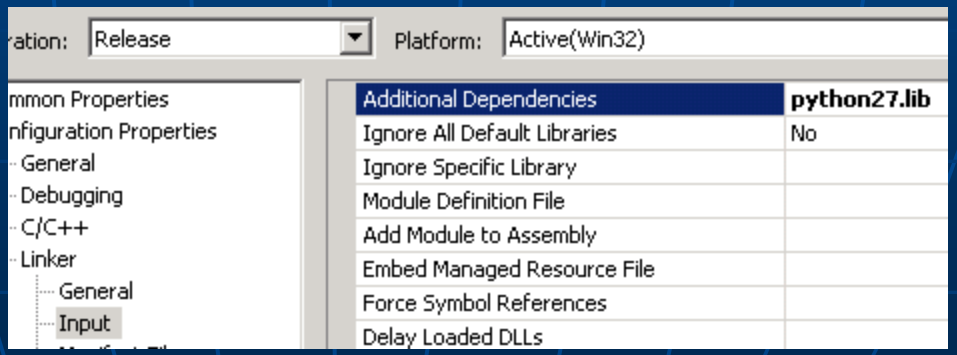
Step 2 (continued)

- Properties (Release)

  Linker|General|Output File: $(OutDir)\$(ProjectName).pyd

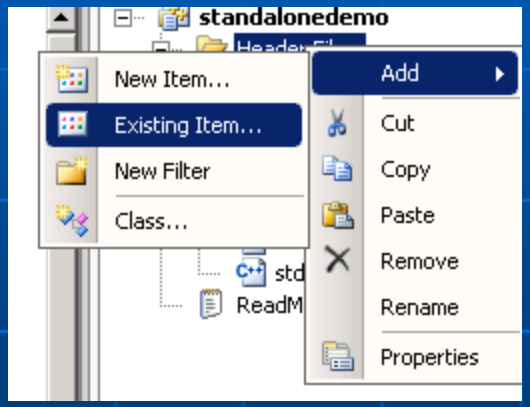
  Linker|Input|Additional Dependencies: python27.lib

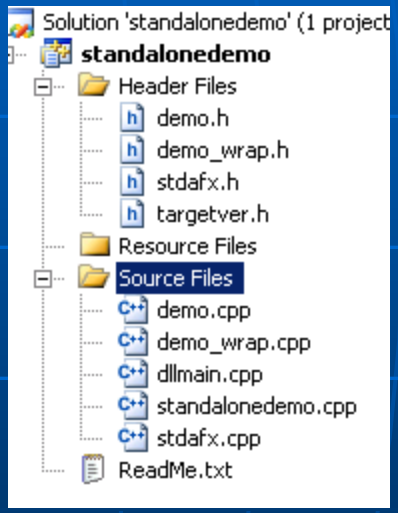




Step 3: Run Debug configuration

- Copy demo.h, demo.cpp, demo_wrap.h, demo_wrap.cpp into project and add as existing items





Step 3 (continued)

- Add sample code for stdafx.h

```
// TODO: reference additional headers your program requires here
#import "libid:6FCCEDE0-179D-4D12-B586-58C88D26CA78" no_namespace raw_
   rename("esriProductCode", "esriVersionProductCode") rename("VersionI
#import "esriFramework.olb" raw_interfaces_only raw_native_types no_nar
#import "esriSystem.olb" raw_interfaces_only raw_native_types no_namesp
#import "esriSystemUI.olb" raw_interfaces_only raw_native_types no_name
#import "esriEditor.olb" raw_interfaces_only raw_native_types no_namesp
#import "esriGeodatabaseUI.olb" raw_interfaces_only raw_native_types no
#import "esriArcMapUI.olb" raw_interfaces_only raw_native_types no_name
#import "esriGeometry.olb" raw_interfaces_only raw_native_types no_name
#import "esriDisplay.olb" raw_interfaces_only raw_native_types no_names
#import "esriGeoDatabase.olb" raw_interfaces_only raw_native_types no_r
#import "esriDataSourcesFile.olb" raw_interfaces_only raw_native_types
#import "esriDataSourcesRaster.olb" raw_interfaces_only raw_native_type
#import "esriDataSourcesGDB.olb" raw_interfaces_only raw_native_types r
#import "esriCarto.olb" raw_interfaces_only raw_native_types no_namespa
#import "esriEditorExt.olb" raw_interfaces_only raw_native_types no_nar
#import "esriNetworkAnalysis.olb" raw_interfaces_only raw_native_types
#import "esriOutput.olb" raw_interfaces_only raw_native_types no_namesp
#import "esriOutputUI.olb" raw_interfaces_only raw_native_types no_name
```

Step 3 (continued)

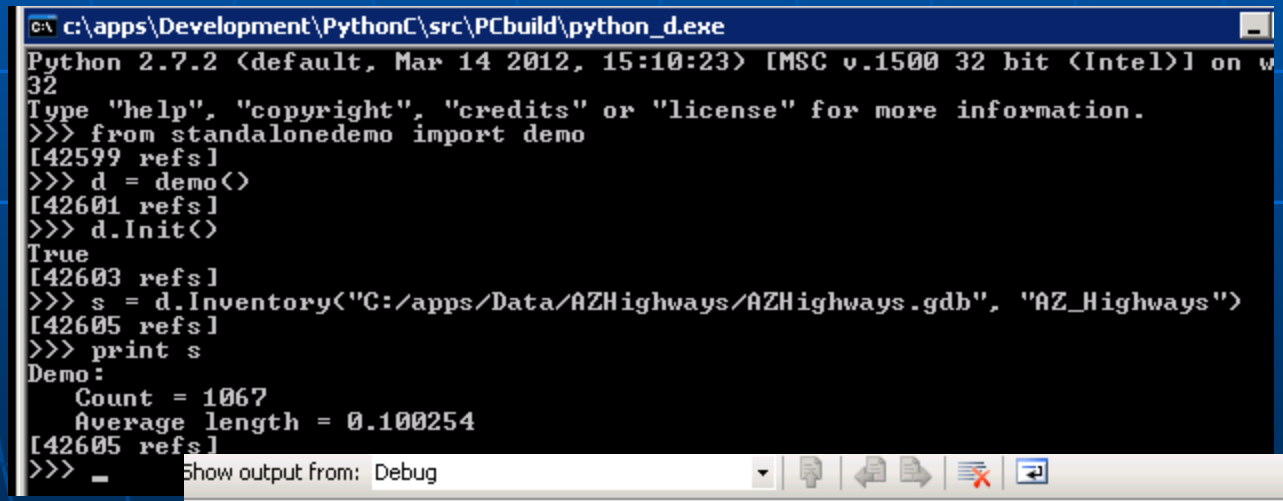
- Add sample code for standalonedemo.cpp

```
#include "stdafx.h"
#include <Python.h>
#include "demo_wrap.h"

static PyMethodDef module_methods[] = {{NULL}}; // No module-level methods

PyMODINIT_FUNC
initstandalonedemo(void)
{
    PyObject* m;
    if (PyType_Ready(&DemoType) < 0)
        return;
    m = Py_InitModule3("standalonedemo", module_methods,
                       "Example module that creates an extension type.");
    if (m == NULL)
      return;
    Py_INCREF(&DemoType);
    PyModule_AddObject(m, "demo", (PyObject *)&DemoType);
}
```

Step 3 (continued)

- Have a shapefile or file geodatabase handy
- Build and run:

```
>>> from standalonedemo import demo
>>> d = demo()
>>> d.Init()
>>> s =
d.Inventory("C:/apps/Data/AZHighways/AZHighways.gdb",
"AZ_Highways")
>>> print s
```



demo_wrap.h:

```
#include "demo.h“

typedef struct
{
        PyObject_HEAD;
        demo *demo;
} Demo;

extern PyTypeObject DemoType;

PyObject * Demo_new(PyTypeObject *type, PyObject *args,
        PyObject *kwds);
PyObject * Demo_Init(Demo *self);
PyObject * Demo_Inventory(Demo *self, PyObject *args);
void Demo_dealloc(Demo* self);
```

demo_wrap.cpp:

```
#include "stdafx.h"
#include <Python.h>
#include "demo_wrap.h"

PyMethodDef Demo_methods[] = {
      {"Init", (PyCFunction)Demo_Init, METH_NOARGS,
            "demo.Init(): Initialize COM and ArcObjects"},
      {"Inventory", (PyCFunction)Demo_Inventory,
METH_VARARGS,
            "demo.Inventory(string ws, string fc):
inventory feature class fc in workspace ws"},
      {NULL}
};
```

demo_wrap.cpp (continued):

```
PyTypeObject DemoType = {
    PyObject_HEAD_INIT(NULL)
    0,                         /*ob_size*/
    "standalonedemo.demo",     /*tp_name*/
    sizeof(Demo),              /*tp_basicsize*/
    0,                         /*tp_itemsize*/
    (destructor)Demo_dealloc,  /*tp_dealloc*/
…
    Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE, /*tp_flags*/
    "demo object",             /*tp_doc*/
…
    Demo_methods,              /*tp_methods*/ // Methods only
    0,                         /*tp_members*/ // No properties
…
    Demo_new,                  /*tp_new*/
};
```

demo_wrap.cpp (continued):

```
PyObject * Demo_new(PyTypeObject *type, PyObject
*args, PyObject *kwds)
{
        Demo *self;
        self = (Demo *)type->tp_alloc(type, 0);
        if (self != NULL)
        {
                self->demo = new demo();
                if (self->demo == NULL)
                {
                        Py_DECREF(self);
                        return NULL;
                }
        }
        return (PyObject *)self;
}
                void Demo_dealloc(Demo* self)
                {
                        delete self->demo;
                        self->ob_type->tp_free((PyObject*)self);
                }
```

demo_wrap.cpp (continued):

```
PyObject * Demo_Init(Demo *self)
{
        if (self->demo == NULL)
                return NULL;
        bool bResult;
        try
        {
                bResult = self->demo->Init();
        }
        catch(...)
        {
                bResult = false;
        }
        if (!bResult)
        {
                Py_INCREF(Py_False);
                return Py_False;
        }
        Py_INCREF(Py_True);
        return Py_True;
}
```

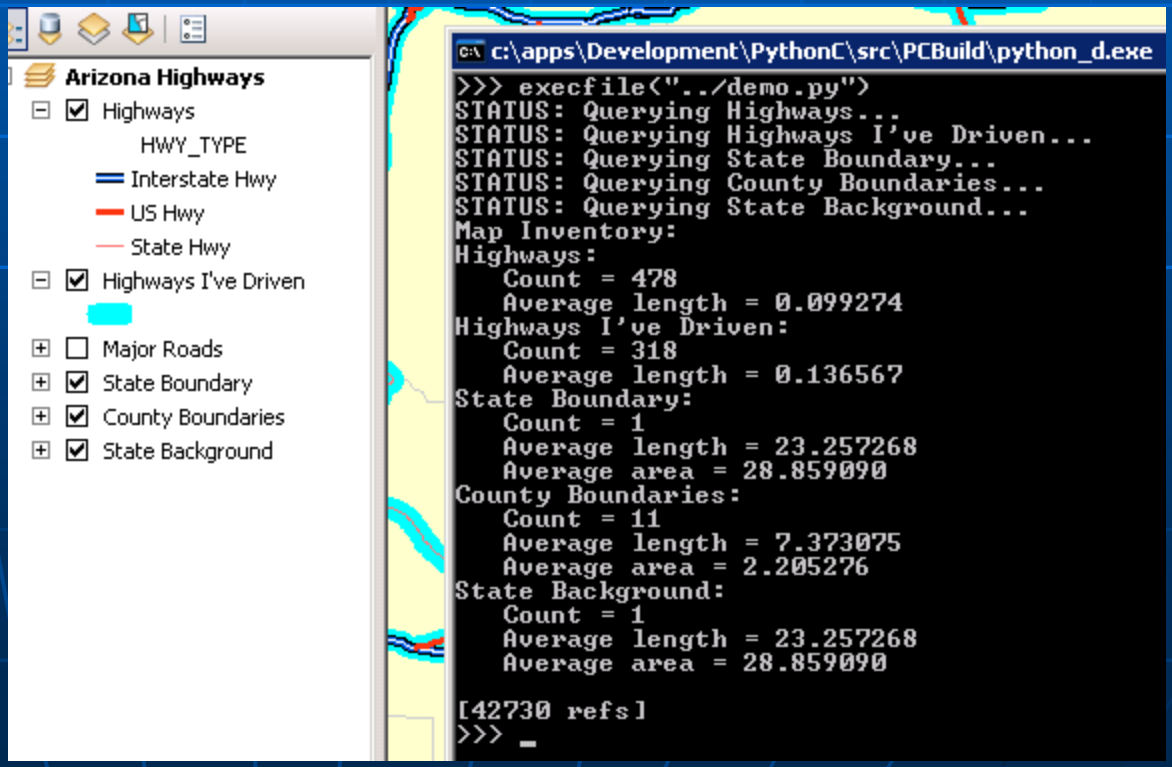
demo_wrap.cpp (continued):

```
PyObject * Demo_Inventory(Demo *self, PyObject *args)
{
	if (self->demo == NULL)
		return NULL;
	const char *ws, *fc;
	if (!PyArg_ParseTuple(args, "ss", &ws, &fc))
	{
		printf("Usage: demo.Inventory((string) workspace,
(string) featureclass)\n");
		return NULL;
	}
	char *s = self->demo->Inventory((char *)ws, (char *)fc);
	PyObject * pyResult = Py_BuildValue("s", s);
	return pyResult;
}
```

Debugging with python_d

- Cannot use site packages
- Cannot debug inside ArcMap
- However, you can use automation for debugging

[Sample Code: PythonC/arcmapdemo]



2:26:43 PM

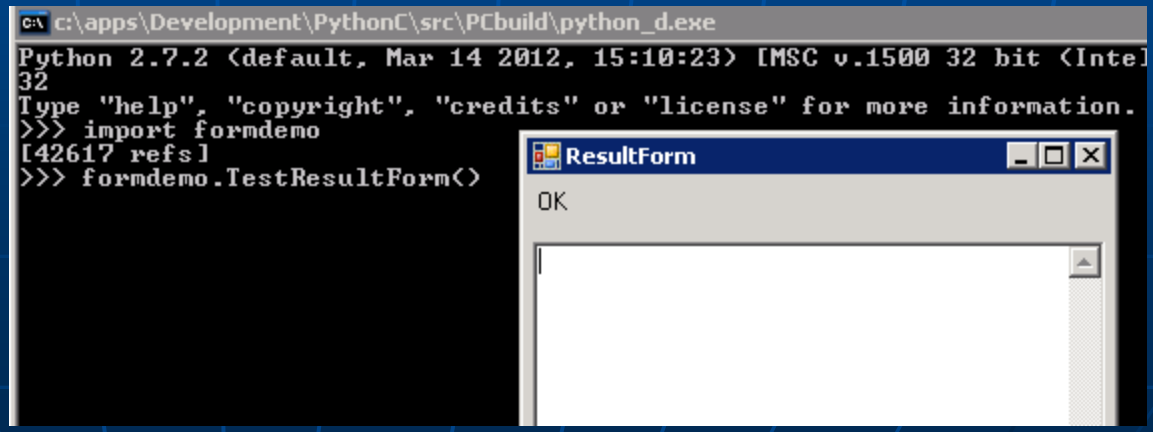
Can you incorporate .NET (CLR) forms?

- Yes, but there are several potential pitfalls
- Keep ArcObjects in unmanaged code!!!

```
#pragma managed(push, off) ... #pragma managed(pop)
```

- RECOMMENDED: Put forms in a separate pyd
- Better yet, why not use wxPython instead?

[Sample Code: PythonC/formdemo]



Why use SWIG or Cython?

Pros:

- Useful for wrapping existing code
- Save time creating wrapper code for new code

Cons:

- Create baggage
- Still require Python development package
- Still must debug in python_d
- Cannot directly consume and return ArcObjects

Why use SWIG?

Pros:

- Supports multiple languages: same C++ can be wrapped for Python and C#
- No compilation required - just unzip and use
- Fast, easy setup: for many cases requires just a few lines of configuration code

Cons:

- Produces a python wrapper along with the pyd
- Requires creating typemap wrapper code for Python objects
- Does not support callbacks to Python

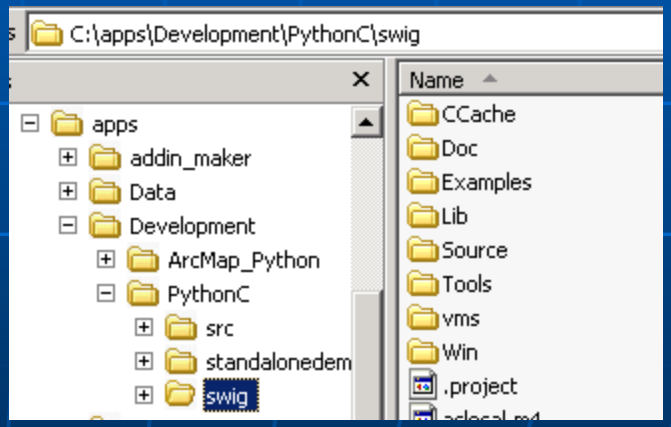
Walkthrough 2: Creating a Python module using SWIG

[Sample Code: PythonC/swigdemo]

2:26:43 PM

Step 1: Set up SWIG

- Set up Python development package as in walkthrough #1 step 1
- Download swigwin-2.0.4.zip
  http://www.swig.org/download.html
- Unzip swigwin-2.0.4.zip to temporary folder
- Create a folder called "swig" adjacent to where you intend to create Python extension projects
- Move contents of swigwin-2.0.4 folder to swig folder



Step 2: Create project

- Create a new project as in walkthrough #1 step 2, called swigdemo
- Properies (debug)

  Linker|General|Output File: $(OutDir)\_$(ProjectName)_d.pyd
- Properties (release)

  Linker|General|Output File: $(OutDir)\_$(ProjectName).pyd
- Add new items: swigdemo.i, and swigdemo_wrap.cpp
- Right-click swigdemo.i and bring up Properties (All Configurations)

  Custom Build Setup|General|Command Line:

  ..\swig\swig.exe -c++ -python -o $(ProjectDir)\$(InputName)_wrap.cpp $(InputPath)

  Custom Build Setup|General|Outputs: $(ProjectDir)\$(InputName)_wrap.cpp
- Right-click swigdemo_wrap.cpp and bring up Properties (All Configurations)

  C/C++|Precompiled Headers|Create/Use Precompiled Header: Not Using Precompiled Headers

Step 3: Build and run Debug configuration

- Copy demo.h and demo.cpp into project and add as existing items
  Add sample code for stdafx.h and swigdemo.i
- Build Debug configuration
- Copy swigdemo.py to Debug folder
- Run python_d:

```
>>> from swigdemo import demo
>>> d = demo()
>>> d.Init()
>>> s =
d.Inventory("C:/apps/Data/AZHighways/AZHighways.gdb",
"AZ_Highways")
>>> print s
```

Why use Cython?

Pros:

- Produces a standalone pyd
- Produces cleaner C wrappers

Cons:

- To install, Cython uses distutils to compile pyds
- Distutils expects VC 2008 unless you tweak it
- Requires writing some wrapper code (in Pyrex)

Walkthrough 3: Creating a Python module using Cython

[Sample Code: PythonC/cythondemo]

Step 1: Set up Cython

- Set up Python development package as in walkthrough #1 step 1
- Download Cython-0.15.1.zip

  http://cython.org/release/Cython-0.15.1.zip
- Unzip to temporary folder
- Verify that Visual C++ 2008 is installed
- Open a command prompt and cd to folder containing setup.py

```
python setup.py install
```

Step 2: Create project

- Create a new project as in walkthrough #1 step 2, called cythondemo
- Make sure python is in the PATH environment
- Add new items: cythondemo.pyx, and cythondemo_wrap.cpp
- Right-click cythondemo.pyx and bring up Properties (All Configurations)

  Custom Build Setup|General|Command Line:

  python -c "from Cython.Compiler.Main import compile; compile('cythondemo.pyx', output_file='$(InputName)_wrap.cpp', cplus=True)"

  Custom Build Setup|General|Outputs: $(ProjectDir)\$(InputName)_wrap.cpp
- Right-click cythondemo_wrap.cpp and bring up Properties (All Configurations)

  C/C++|Precompiled Headers|Create/Use Precompiled Header: Not Using Precompiled Headers

Step 3: Build and run Debug configuration

- Copy demo.h and demo.cpp into project and add as existing items
  Add sample code for stdafx.h and cythondemo.pyx
- Build Debug configuration
- Run python_d:

```
>>> from cythondemo import PyDemo
>>> d = PyDemo()
>>> d.Init()
>>> s =
d.Inventory("C:/apps/Data/AZHighways/AZHighways.gdb",
"AZ_Highways")
>>> print s
```

“And now for something completely different….”

2:26:44 PM

Why use ctypes?

Pros:

- No Python development package required
- Code can be debugged inside an ArcMap session
- Can easily consume and return comtypes-wrapped ArcObjects
- The DLL produced can be used in both Python and .NET

Cons:

- Only C exported functions can be used by ctypes: C++ objects cannot be directly exposed

Installing comtypes and wxPython and creating Python add-ins

Presentation:

ArcMap and Python: Closing the VBA Gap

ctypes – A foreign function library for Python

http://docs.python.org/library/ctypes.html

2:26:44 PM

Walkthrough 4: Creating a helper module for ArcMap using ctypes

[Sample Code: PythonC/ctdemo]

2:26:44 PM

The maphelper class:

```
// maphelper.h
#include <string>
using namespace std;
typedef void *StatusCallback(char * msg);
class maphelper
{
private:
        StatusCallback *_pStatusCall;
        string _sResult;
public:
        maphelper();
        void Init(StatusCallback *pStatusCall);
        IMap * GetFocusMap();
        char * Inventory(IMap * ipMap, IEnvelope * ipExtent);
private:
        void IntMapInventory(IMap * ipMap, IEnvelope * ipExtent);
        void Status(string sMsg);
};
```

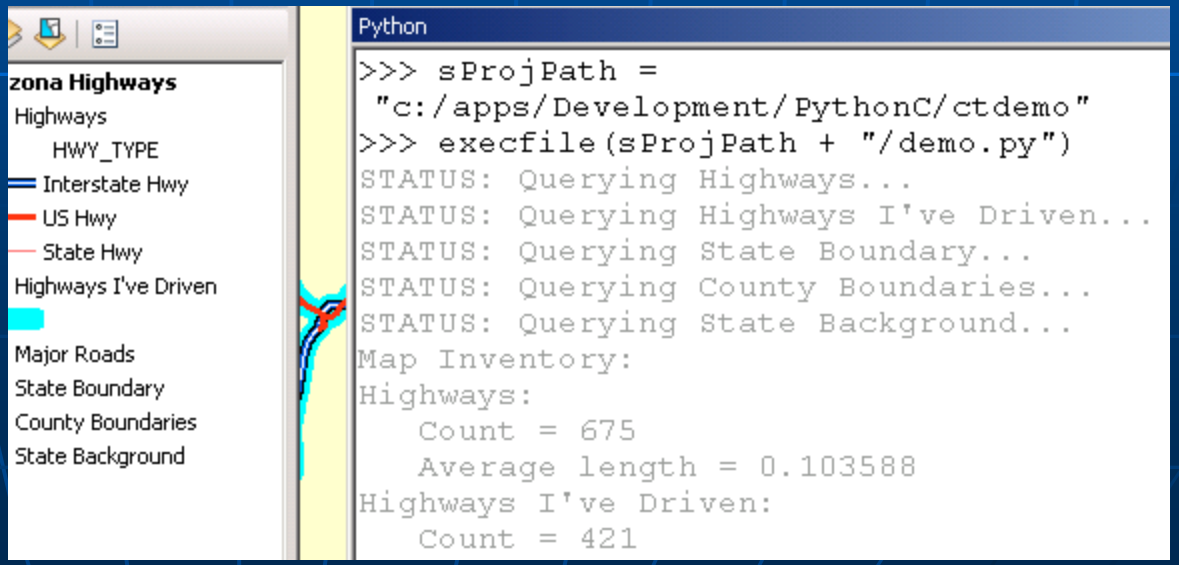
Step 1: Create project

- Project type: Win32 Project
  Name: ctdemo
  Uncheck "Create directory for solution"
  Application type: DLL
- Properties (all configurations)
  C/C++|General|Additional Include Directories:
  [C:\Program Files*]\ArcGIS\Desktop10.1\com
  *wherever ArcGIS is installed
- Properties (debug)
  Debugging|Command:
  [C:\Program Files*]\ArcGIS\Desktop10.1\bin\ArcMap.exe
  *wherever ArcGIS is installed
- Properties (release)
  C/C++|Code Generation|Runtime Library: Multi-threaded (/MT)

Step 2: Run Debug configuration

- Copy the following into project and add as existing items:
  maphelper.h, maphelper.cpp
- Add sample code for stdafx.h and ctdemo.cpp
- Copy ctdemo.py and demo.py to project folder
- Build and run:
  Open an mxd
  Open ArcMap's Python window

```
>>> sProjPath = "c:/apps/Development/PythonC/ctdemo"
>>> execfile(sProjPath + "/demo.py")
```



2:26:44 PM

```
// ctdemo.cpp

#include "stdafx.h"
#include "maphelper.h"

extern "C" __declspec( dllexport )
void * maphelper_New()
{
      maphelper *m = new maphelper;
      return (void *)m;
}

extern "C" __declspec( dllexport )
void maphelper_Init(void * self, StatusCallback *pStatusCall)
{
      maphelper *m = (maphelper*)self;
      m->Init(pStatusCall);
}
```

```
// ctdemo.cpp (continued)

extern "C" __declspec( dllexport )
char * maphelper_Inventory(void * self, IMap * ipMap,
IEnvelope * ipExtent)
{
      maphelper *m = (maphelper*)self;
      return m->Inventory(ipMap, ipExtent);
}

extern "C" __declspec( dllexport )
void maphelper_Delete(void * self)
{
      maphelper *m = (maphelper*)self;
      delete m;
}
```

```
# ctdemo.py
class maphelper_factory:
    _dll = None
    def __init__(self, path=None):
        from os import getcwd
        from ctypes import cdll # NOT windll!
        if not (self._dll is None):
            return
        if path is None:
            sPath = getcwd()
        else:
            sPath = path
        sPath += "\\ctdemo.dll"
        self._dll = cdll.LoadLibrary(sPath)
        from comtypes.client import GetModule
        sLibPath = GetLibPath()
        GetModule(sLibPath + "esriArcMapUI.olb")
    def new_maphelper(self):
        return maphelper(self._dll)
```

Beware: each call to LoadLibrary loads a new instance of the dll!

2:26:44 PM

```
# ctdemo.py (continued)
class maphelper:
    _dll = None
    _mh = None
    _cb = None
    def __init__(self, dll=None):
        from ctypes import c_void_p
        self._dll = dll
        self._mh = c_void_p(self._dll.maphelper_New())
    def __del__(self):
        self._dll.maphelper_Delete(self._mh)
    def Init(self, StatusCallback):
        from ctypes import CFUNCTYPE, c_char_p
        STATUSFUNC = CFUNCTYPE(None, c_char_p)
        try:
            self._cb = STATUSFUNC(StatusCallback)
            self._dll.maphelper_Init(self._mh, self._cb)
        except:
            pass
        return
```

```
# ctdemo.py (continued)
    def GetFocusMap(self):
        from ctypes import POINTER
        import comtypes.gen.esriCarto as esriCarto
        pMap = None
        try:
            pMap = POINTER(esriCarto.IMap)\
(self._dll.maphelper_GetFocusMap(self._mh))
        except:
            pass
        return pMap
    def Inventory(self, pMap, pEnv):
        from ctypes import c_char_p
        sResult = ""
        try:
            sResult = c_char_p(\
self._dll.maphelper_Inventory(self._mh, pMap, pEnv))
            sResult = sResult.value
        except:
            pass
        return sResult
```

```
# demo.py
sDebugPath = sProjPath + "/Debug"
import sys
sys.path.insert(0, sProjPath)
from ctdemo import *
import comtypes.gen.esriCarto as esriCarto
h = demohost()
mf = maphelper_factory(sDebugPath)
m = mf.new_maphelper()
m.Init(h.Status)
pMap = m.GetFocusMap()
pAV = pMap.QueryInterface(esriCarto.IActiveView)
pExtent = pAV.Extent
s = m.Inventory(pMap, pExtent)
print s
del m
```

ArcMap add-in examples (10.1):

Python add-in (ctypes/wxPython): PythonC/addins/pydemo
C# add-in (pinvoke): PythonC/addins/csdemo
Python add-in (pyd/CLR): PythonC/addins/clrdemo

Questions?

- Mark Cederholm
  mcederholm@uesaz.com
- This presentation and sample code may be downloaded at:

http://www.pierssen.com/arcgis10/python.htm