Managed and Unmanaged C++

Jim Fawcett

CSE681 – Software Modeling and Analysis

Fall 2005

References

- Essential Guide to Managed Extensions of C++, Challa and Laksberg, Apress, 2002

- Developing Applications with Visual Studio.Net, Richard Grimes, Addison-Wesley, 2002

Managed C++ Syntax

- Include system dlls from the GAC:
  - #include <mscorlib.dll>
- Include standard library modules in the usual way:
  - #include <iostream>
- Use scope resolution operator to define namespaces
  - using namespace System::Text;
- Declare .Net value types on stack
- Declare .Net reference types as pointers to managed heap
  - String* str = new String(S"Hello World");
- Managed arrays are declared in .Net style:
  - int anArray __gc[] = new int __gc[5];
  - Int32 anotherArray[] = new Int32[5]; // __gc not required for
    // managed types

Managed Classes

- **Syntax:**

  __gc class myClass { … };

  myClass* mc1 = new myClass();

  myClass& mc2 = *new myClass();

- Can hold pointers and references to reference types.
- Can hold values, pointers, and references to value types.
- Deletion of pointers to managed objects fails to compile.
- Can call global functions and members of unmanaged classes without marshaling.
- Can hold a pointer to an unmanaged object, but is responsible for creating it on the C++ heap and eventually destroying it.

Defaults

- A pointer to a managed type is managed. For unmanaged types you can specify a managed pointer:
  - int __gc* pInt;
- A reference to a managed type is managed. For unmanaged types you can specify a managed reference:
  - int __gc& rInt = &i;
- An array of managed types is managed. You specify a managed array of unmanaged types as:
  - int array _gc[] = new int _gc[10];
- A class by default is unmanaged. You specify managed as:
  - __gc class myClass { … };

Type Conversions

C++ Type	CTS Signed Type	CTS Unsigned Type
char	Sbyte	Byte
short int	Int16	UInt16
int, __int32	Int32	UInt32
long int	Int32	UInt32
__int64	Int64	UInt64
float	Single	N/A
double	Double	N/A
long double	Double	N/A
bool	Boolean	N/A

Extensions to Standard C++

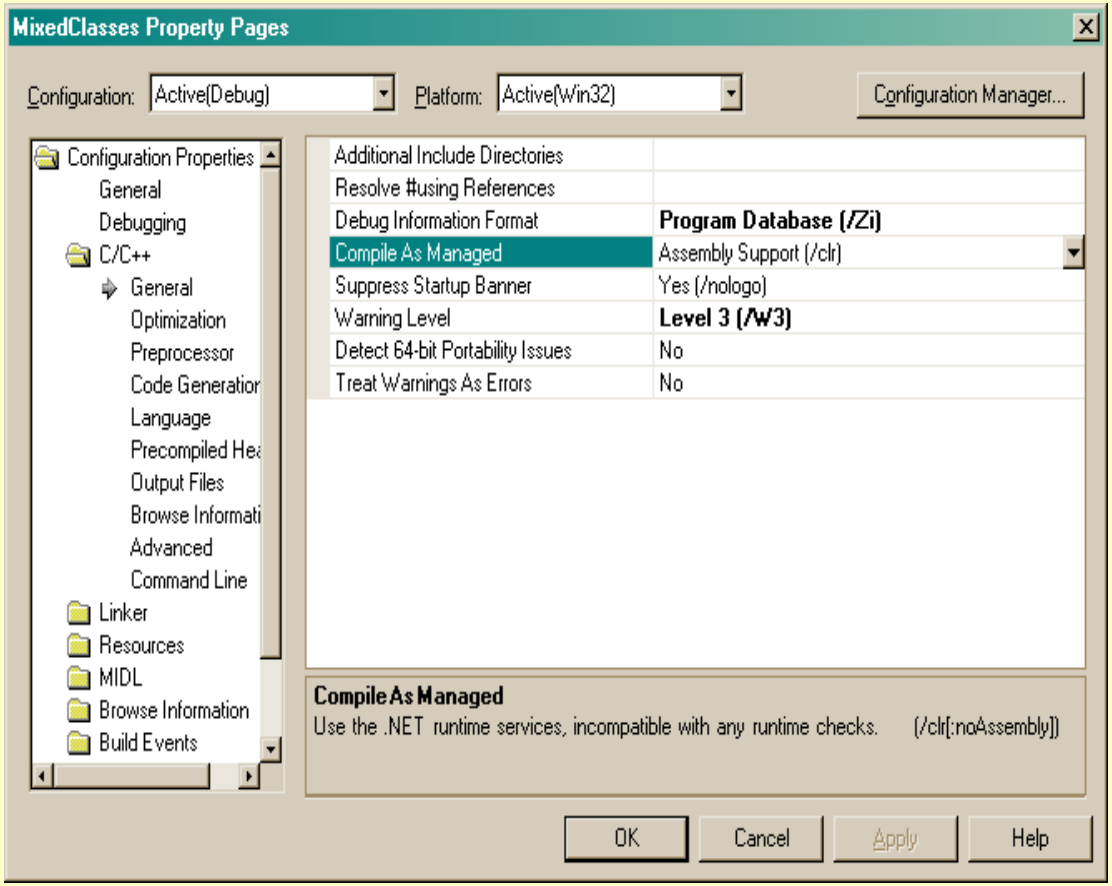
- Managed classes may have the qualifiers:
  - abstract
  - sealed
- A managed class may have a constructor qualified as static, used to initialize static data members.
- Managed classes may have properties:
  - __property int get_Length() { return _len; }
  - __property void set_Length(int len) { _len = len; }
- A managed class may declare a delegate:
  - __delegate void someFunc(int anArg);

Managed Exceptions

- A C++ exception that has a managed type is a managed exception.
- Application defined exceptions are expected to derive from System::ApplicationException (this appears to be depricated) so just derive from System::Exception.
- Managed exceptions may use a finally clause:
  - try { … } catch(myExcept &me) { … } __finally { … }
- The finally clause always executes, whether the catch handler was invoked or not.
- Only reference types, including boxed value types, can be thrown.

Code Targets

- An unmanaged C++ program can be compiled to generate managed code using the /clr option.
- You can mix managed and unmanaged code using #pragma managed and #pragma unmanged. Metadata will be generated for both.



Mixing Managed and Unmanaged Code

- You may freely mix unmanaged and managed classes in the same compilation unit.
  - Managed classes may hold pointers to unmanaged objects.
  - Unmanaged classes may hold pointers to managed objects wrapped in gcroot:
    - #include <vcclr.h>
    - Declare: gcroot<System::String*> pStr;
  - That helps the garbage collector track the pStr pointer.
  - Calls between the managed and unmanaged domains are more expensive than within either domain.

Limitations of Managed Classes

- No templates – sigh!
- Only single inheritance of implementation is allowed.
- Managed classes can not inherit from unmanaged classes and vice versa.
- No copy constructors or assignment operators are allowed.
- Member functions may not have default arguments.
- Friend functions and friend classes are not allowed.

- Typedefs in managed classes are currently not allowed.
- Const and volatile qualifiers on member functions are currently not allowed.

Platform Invocation - PInvoke

- Call Win32 API functions like this:
  - [DllImport("kernel32.dll")]
    extern "C" bool Beep(Int32,Int32);
  - Where documented signature is:
    BOOL Beep(DWORD,DWORD)
- Can call member functions of an exported class
  - See Marshaling.cpp, MarshalingLib.h

Additions to Managed C++ in VS 2005

- Generics
  - Syntactically like templates but bind at run time
  - No specializations
  - Uses constraints to support calling functions on parameter type
- Iterators
  - Support foreach construct
- Anonymous Methods
  - Essentially an inline delegate
- Partial Types, new to C#, were always a part of C++
  - Class declarations can be separate from implementation
  - Now, can parse declaration into parts, packaged in separate files

End of Presentation