C++/CLI

Jim Fawcett

CSE681 – Software Modeling and Analysis

Fall 2006

References

- C++/CLI
  - A Design Rationale for C++/CLI, Herb Sutter, http://www.gotw.ca/publications/C++CLIRationale.pdf
  - Moving C++ Applications to the Common Language Runtime, Kate Gregory, http://www.gregcons.com/KateBlog/CategoryView.aspx?category=C++#a7dfd6ea3-138a-404e-b3e9-55534ba84f22
  - C++/CLI FAQ, http://www.winterdom.com/cppclifaq/
  - C++: Most Powerful Language for .NET Framework Programming, Kenny Kerr, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/VS05Cplus.asp?frame=true

Managed C++ Syntax

- Include system dlls from the GAC:
  - #include < System.Data.dll>
  - #include <mscorlib.dll> - not needed with C++/CLI
- 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 = gcnew String("Hello World");

Managed Classes

- **Syntax:**

class N { … };	native C++ class
ref class R { … };	CLR reference type
value class V { … };	CLR value type
interface class I { … };	CLR interface type
enum class E { … };	CLR enumeration type

  - N is a standard C++ class. None of the rules have changed.
  - R is a managed class of reference type. It lives on the managed heap and is referenced by a handle:
    - R^ rh = gcnew R;
    - delete rh; [optional: calls destructor which calls Dispose() to release unmanaged resources]
    - Reference types may also be declared as local variables. They still live on the managed heap, but their destructors are called when the thread of execution leaves the local scope.
  - V is a managed class of value type. It lives in its scope of declaration.
    - Value types must be bit-wise copyable. They have no constructors, destructors, or virtual functions.
    - Value types may be boxed to become objects on the managed heap.
  - I is a managed interface. You do not declare its methods virtual. You qualify an implementing class's methods with override (or new if you want to hide the interface's method).
  - E is a managed enumeration.
- N can hold "values", handles, and references to managed types.
- N can hold values, handles, and references to value types.
- N can call methods of managed types.
- R can call global functions and members of unmanaged classes without marshaling.
- R can hold a pointer to an unmanaged object, but is responsible for creating it on the C++ heap and eventually destroying it.

From Kate Gregory's Presentation see references

	Native	Managed
Pointer / Handle	`*`	`^`
Reference	`&`	`%`
Allocate	`new`	`gcnew`
Free	`delete`	`delete`[1]
Use Native Heap	✓	✓[2]
Use Managed Heap	✘	✓
Use Stack	✓	✓
Verifiability	`*` and `&` never	`^` and `%` always

[1] Optional

[2] Value types only

Mixing Pointers and Arrays

- Managed classes hold handles to reference types:
  - ref class R 2{ … private: String^ rStr; };
- Managed classes can also hold pointers to native types:
  - ref class R1 { … private: std::string* pStr; };
- Unmanaged classes can hold managed handles to managed types:
  - class N { … private: gcroot<String^> rStr; };
- Using these handles and references they can make calls on each other's methods.
- Managed arrays are declared like this:
  - Array<String^>^ ssarr = gcnew array<String^>(5);
  - ssarr[i] = String::Concat("Number", i.ToString()); 0<= i <= 4
- Managed arrays of value types are declared like this:
  - array<int>^ strarray = gcnew array<int>(5);
  - Siarr[i] = i; 0<=i<=4;

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 Length
    ```
    {
        int get() { return _len; }
        void set(int value) { _len = value; }
    }
    ```
  - property int Length; // short hand for the declaration above

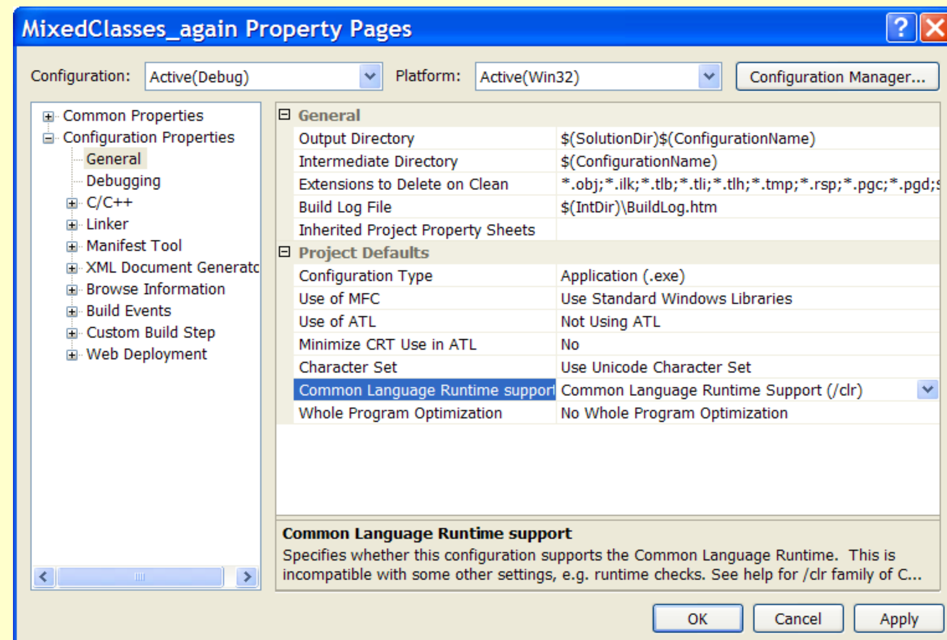
- A managed class may declare a delegate:
  - delegate void someFunc(int anArg);

Managed Exceptions

- A C++ exception that has a managed type is a managed exception.
- Application defined exceptions are expected to derive from System::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 handles 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.
- Note, all of the above means, that you can use .Net Framework Class Libraries with unmanaged code, and you can use the C++ Standard Library (not the STL yet) with managed code.

Features Supported (ECMA Std)

http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-372.pdf - Windows Internet Explorer

This clause specifies the features of a class that are new in C++/CLI. However, not all of these features are available to all classes. The class-related features that are supported by native classes (§20), ref classes (§21), value classes (§22), and interfaces (§25), are specified in the clauses that define those types. [*Note:* A summary of that support is shown in the following table:

Feature	Native class	Ref class	Value class	Interface
Assignment operator	X	X		
Class modifier	X	X	X	
Copy constructor	X	X		
Default constructor	X	X		
Delegate definitions	X	X	X	X
Destructor	X	X		X
Events		X	X	X
Finalizer		X		
Function modifiers	X	X	X	n/a
Initonly field		X	X	X
Literal field		X	X	X
Member of delegate type		X	X	
Override specifier	X	X	X	n/a
Parameter arrays	X	X	X	X
Properties		X	X	X
Reserved member names		X	X	X
Static constructor		X	X	X
Static operators	X	X	X	X

end note]

Limitations of Managed Classes

- Generics and Templates are now supported, but STL/CLI has not shipped yet.
- Only single inheritance of implementation is allowed.
- Managed classes can not inherit from unmanaged classes and vice versa. This is may be a future addition.
- No copy constructors or assignment operators are allowed for value types.
- Member functions may not have default arguments.
- Native types can grant friendship. Managed types cannot.
- 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 for each 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

Using Frameworks in MFC

from Kate Gregory’s Presentation

- Visual C++ 2005 allows you to use new Frameworks libraries in MFC Applications
- MFC includes many integration points
  - MFC views can host Windows Forms controls
  - Use your own Windows Forms dialog boxes
  - MFC lets you use Windows Forms as CView
  - Data exchange and eventing translation handled by MFC
  - MFC handles command routing
- MFC applications will be able to take advantage of current and future libraries directly with ease

End of Presentation