C++/CLI

Jim Fawcett

CSE687 – Object Oriented Design

Spring 2009

References

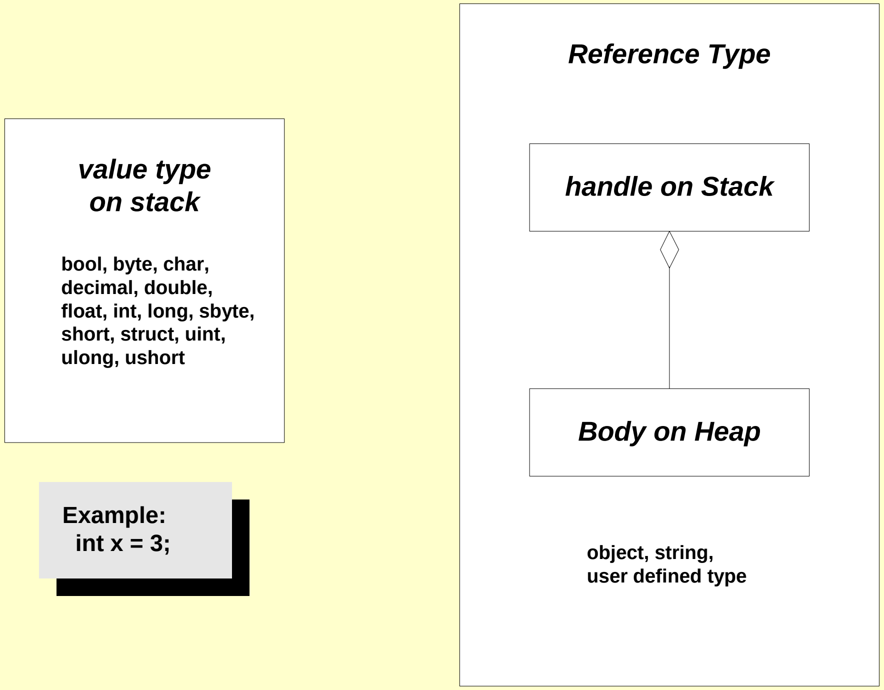
- C++/CLI
  - A Design Rationale for C++/CLI, Herb Sutter, http://www.gotw.ca/publications/C++CLIRationale.pdf
  - Moving C++ Applications to the Common Language Runtime, Kate Gregory, http://www.gregcons.com/KateBlog/CategoryView.aspx?category=C++#a7dfd6ea3-138a-404e-b3e9-55534ba84f22
- Manged Extensions
  - C++/CLI in Action, Nishant Sivakumar, Manning, 2007

Comparison of Object Models

- ***Standard C++ Object Model***
  - All objects share a rich memory model:
    - Static, stack, and heap
  - Rich object life-time model:
    - Static objects live for the duration of the program.
    - Objects on stack live within a scope defined by { and }.
    - Objects on heap live at the designer's discretion.
  - Semantics based on deep copy model.
    - That's the good news.
    - That's the bad news.
  - For compilation, a source file must include information about all the types it uses.
    - That's definitely bad news.
    - But it has a work-around, e.g., design to interface not implementation. Use object factories.

- ***.Net Managed Object Model***
  - More Spartan memory model:
    - Value types are stack-based only.
    - Reference types (all user defined types and library types) live on the managed heap.
  - Non-deterministic life-time model:
    - All reference types are garbage collected.
    - That's the good news.
    - That's the bad news.
  - Semantics based on a shallow reference model.
  - For compilation, a source file is type checked with metadata provided by the types it uses.
    - That is great news.
    - It is this property that makes .Net components so simple.

.Net Object Model



bool, byte, char,
decimal, double,
float, int, long, sbyte,
short, struct, uint,
ulong, ushort

object, string,
user defined type

Example:
```
int x = 3;
```

Example:
```
myClass mc = new myClass(args);
string myStr = "this is some text";
```

Language Comparison

- Standard C++
  - Is an ANSI and ISO standard.
  - Has a standard library.
  - Universally available:
    - Windows, UNIX, MAC
  - Well known:
    - Large developer base.
    - Lots of books and articles.
  - Programming models supported:
    - Objects
    - Procedural
    - Generic
  - Separation of Interface from Implementation:
    - Syntactically excellent
      - Implementation is separate from class declaration.
    - Semantically poor
      - See object model comparison.

- .Net C#, Managed C++, …
  - Is an ECMA standard, becoming an ISO standard.
  - Has defined an ECMA library.
  - Mono project porting to UNIX
  - New, but gaining a lot of popularity
    - Developer base growing quickly.
    - Lots of books and articles.
  - Programming models supported:
    - objects.
  - Separation of Interface from Implementation:
    - Syntactically poor
      - Implementation forced in class declaration.
    - Semantically excellent
      - See object model comparison.

Library Comparison

- Standard C++ Library
  - Portable across most platforms with good standards conformance
  - I/O support is stream-based
    - console, files, and, strings
  - Flexible container facility using Standard Template Library (STL)
    - But no hash-table containers
  - No support for paths and directories
  - Strings, no regular expressions
  - No support for threads
  - No support for inter-process and distributed processing
  - No support for XML
  - Platform agnostic

- .Net Framework Class Library
  - Windows only but porting efforts underway
  - I/O support is function-based
    - console and files
  - Fixed set of containers that are not very type safe.
    - Has hash-table containers
  - Strong support for paths and directories
  - Strings and regular expressions
  - Thread support
  - Rich set of inter-process and distributed processing constructs
  - Support for XML processing
  - Deep support for Windows but very dependent on windows services like COM

Comparison of Library Functionality

Functionality	.Net Framework Libraries	Standard C++ Library
Extendable I/O	Weak	Strong
strings	Strong	Strong
Composable Containers	Moderately good	Strong
Paths and Directories	Strong	No
Threads	Strong	No
Sockets	Moderately good	No
XML	Strong	No
Forms	Strong	No
Reflection	Strong	No

Comparison of Library Functionality

Functionality	Support Provided in Code from Website	Support Provided by you in Projects S’09
Extendable I/O	-	-
strings	-	-
Composable Containers	HashTable	No
Paths and Directories	FileInfo, FileSystem	No
Threads	Thread, Lock classes	No
Sockets	SocketCommunicator	No
XML	Reader, Writer, no DOM	XMLDOM class
Forms	-	-
Reflection	No	No

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; }
    }
    ```

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

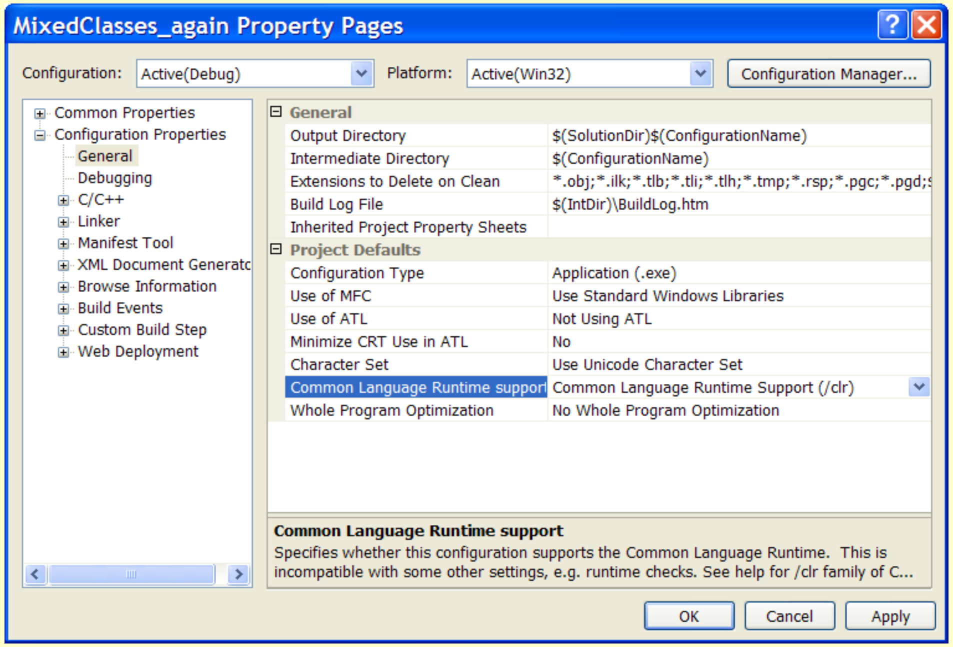
Managed Exceptions

- 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 C++ code in same file.
- Managed C++ can call C# code in a separate library and vice versa.



MixedClasses_again Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generatc
  - Browse Information
  - Build Events
  - Custom Build Step
  - Web Deployment

General	
Output Directory	$(SolutionDir)$(ConfigurationName)
Intermediate Directory	$(ConfigurationName)
Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;$
Build Log File	$(IntDir)\BuildLog.htm
Inherited Project Property Sheets	
Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Minimize CRT Use in ATL	No
Character Set	Use Unicode Character Set
Common Language Runtime support	Common Language Runtime Support (/clr)
Whole Program Optimization	No Whole Program Optimization

Common Language Runtime support
Specifies whether this configuration supports the Common Language Runtime. This is incompatible with some other settings, e.g. runtime checks. See help for /clr family of C...

OK Cancel Apply

Mixing Managed and Unmanaged Code

- You may freely mix unmanaged and managed C++ 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 with managed code.

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

Limitations of Managed Classes

- Generics and Templates are now supported.
- Only single inheritance of implementation is allowed.
- Managed classes can not inherit from unmanaged classes and vice versa. This may be a future addition.
- No copy constructors or assignment operators are allowed.
- Member functions may not have default arguments.
- Friend functions and friend classes are 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)
  - Or, you can call native C++ which then calls the Win32 API
- 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

End of Presentation