Modern COM Programming in D

Lionello Lunesu

Information Services - XML

“Hello?”

```
import std.stdio;

/// The entry point of the application.
/// Params:
///  args = the command line arguments
void main(string[] args)
{
    writeln("世界,你好!");
}
```

Overview

1. Modern COM Programming?
2. The D Programming Language
3. Projecting COM interfaces
4. Under the hood
5. Hands on!

COM in C++

```
HRESULT hr = ::CoInitialize(NULL);
if (FAILED(hr))
     return 1;

// Create a new empty document
IXMLDOMDocument* pDocument = NULL;
hr = ::CoCreateInstance(CLSID_DOMDocument60, NULL, CLSCTX_INPROC,
     IID_IXMLDOMDocument, (void**)&pDocument);
if (FAILED(hr))
     goto CleanUp;

// Load an XML document from an inline string
BSTR bstrXml = ::SysAllocString(L"<root>\47026\72514,\47540\54575!<a>hello</a><a>world</a></root>");
VARIANT_BOOL success = VARIANT_FALSE;
hr = pDocument->loadXML(bstrXml, &success);
if (FAILED(hr) || !success)
     goto CleanUp;

VARIANT vNodeValue;
::VariantInit(&vNodeValue);

// Get the root of the parsed XML document
IXMLDOMElement* pRoot = NULL;
hr = pDocument->get_documentElement(&pRoot);
if (SUCCEEDED(hr))
{
     // Get the first node's value
     IXMLDOMNode* pText = NULL;
     hr = pRoot->get_firstChild(&pText);
     if (SUCCEEDED(hr))
     {
```

COM in C++

- Very verbose
  - Smart pointers help a lot
  - Still have to remember ownership, QueryInterface
  - Method calls instead of properties
- Manual error checking
  - Easy to forget a check
  - Which sucks less: triangulation or goto?
  - Where did the error occur?
- *What is this code trying to do?*
  - Useful code << boilerplate code

COM in C#

```
class Program
{
    static void Main(string[] args)
    {
        // Create a new empty document
        var doc = new MSXML2.FreeThreadedDOMDocument60();

        // Load an XML document from an inline string
        var success = doc.loadXML("<root>世界，你好！<a>Hello</a><a>World</a></root>");
        if (!success)
            return;

        // Get the first node's value
        var text = doc.documentElement.firstChild.nodeValue;
        // Write the node's value to the console
        System.Console.WriteLine(text);

        // Perform a simple XPath query
        var nodelist = doc.selectNodes("/root/a/text()");

        // Do iteration through the child nodes
        foreach (MSXML2.IXMLDOMNode node in nodelist)
            System.Console.WriteLine(node.nodeValue);

        // Access the child nodes by index
        for (int t = 0; t < nodelist.length; ++t)
            System.Console.WriteLine(nodelist[t].nodeValue);
    }
}
```

COM in C#

- It just works
  - No CoInit/Uninit, AddRef/Release, QueryInterface
- …but uses marshaling
  - Bad performance
- Non-deterministic behavior
  - When are the interfaces released?
- dispinterface becomes ‘dynamic’ type
  - No intellisense, no auto-complete

Modern COM in D

```d
import std.stdio, winrtxml;

void main()
{
	// Create a new empty document
	auto doc = XmlDocument();

	// Load an XML document from an inline string
	doc.LoadXml("<root>世界你好<a>hello</a><a>world</a></root>");

	// Get the first node's value
	auto text = doc.DocumentElement.FirstChild;
	// Write the node's value to the console
	writeln(text.NodeValue);

	// Perform a simple XPath query
	auto nodes = doc.SelectNodes("/root/a/text()");

	// Access the child nodes by index
	foreach (node; nodes.First)
		writeln(node.NodeValue);

	// Do iteration through the child nodes
	for (uint t = 0; t < nodes.length; ++t)
		writeln(nodes[t].NodeValue);
}
```

Modern COM in D

- Best of both worlds:
- The **productivity** of C#
  - No AddRef/Release/QueryInterface
  - Garbage collection
- The **performance** of C++
  - Access all types natively = no marshaling
  - Deterministic behavior (RAII)
- …plus other unique features!

The D Programming Language

- Systems programming language inspired by C++
- Created by Walter Bright, *first native C++ compiler*
- Supported by Andrei Alexandrescu, *Modern C++ Design*
- Community driven, open-source (Not GPL)
- Multi-platform (Windows, Linux, FreeBSD, OSX)
- Multi-paradigm (IP, OOP, FP, DBC, TDD)

High performance

- Systems language
  - Direct access to OS, C, C++ APIs
- Compiles to native code
- Support for inline assembly
- Compile-time function evaluation
- Built-in profiler and code coverage tool

Improved productivity

- No pre-processor, no header files
- All types are platform independent
  - `int` is *always* 32 bit; `long` is *always* 64 bit
- Built-in arrays and associative arrays
  - Arrays know their length and can be resized
  - Strings are arrays of Unicode characters
- Support for unit-tests, contracts, invariants
- Meta-programming (code generation)

Native COM support

- The D compiler recognizes `IUnknown`
  - COM virtual function table layout
  - STDMETHODCALL calling convention
- *“Look Mom, no marshaling!”*

D's modern feature set

- Type deduction

```
auto ptr = CoTaskMemAlloc(1000);
```

- Scope guards (RAII)

```
scope(exit) CoTaskMemFree(ptr);
```

- Closures

```
auto square = (int i) { return i*i; };
```

- Safe variadic arguments (uses RTTI)

```
writefln("%s %s %s", "abc", 2, 5.6);
```

- Universal iteration

```
foreach (key, value; myHashTable) …
```

Modern COM programming 101

- Encapsulate the COM interface
  - Create a new type that mimics the interface
  - Smart pointer semantics
  - Same number of methods
- Use this new type from your app
  - Never access the raw interface
- Inside each method:
  1. Convert input parameter(s)
  2. Call raw COM method
  3. Check HRESULT, if FAILED throw
  4. Convert output parameter(s)

Projecting COM interfaces

IDL (input)

```
[
  odl,
  uuid(000208D5-0000-0000-C000-000000000046),
  helpcontext(0x00020000),
  dual,
  oleautomation
]
interface _Application : IDispatch {
    [id(0x00000094), propget, helpcontext(0x00020001)]
    HRESULT Application([out, retval] Application** RHS);
    [id(0x00000095), propget, helpcontext(0x00020002)]
    HRESULT Creator([out, retval] XlCreator* RHS);
    [id(0x00000096), propget, helpcontext(0x000207d1)]
    HRESULT Parent([out, retval] Application** RHS);
    [id(0x00000131), propget, helpcontext(0x000203e9)]
    HRESULT ActiveCell([out, retval] Range** RHS);
    [id(0x000000b7), propget, helpcontext(0x000203ea)]
    HRESULT ActiveChart([out, retval] Chart** RHS);
    [id(0x0000032f), propget, hidden, helpcontext(0x000203eb)]
    HRESULT ActiveDialog([out, retval] DialogSheet** RHS);
    [id(0x000002f6), propget, hidden, helpcontext(0x000203ec)]
    HRESULT ActiveMenuBar([out, retval] MenuBar** RHS);
    [id(0x00000132), propget, helpcontext(0x000203ed)]
    ...
```

D (output)

```
struct _Application
{
    private raw._Application _ptr;

    public this(this) { if (_ptr) ptr.AddRef(); }
    public ~this() { if (_ptr) ptr.Release(); }

    @property
    public Application Application() { ... }

    @property
    public XlCreator Creator() { ... }

    @property
    public Application Parent() { ... }

    @property
    public Range ActiveCell() { ... }

    @property
    public Chart ActiveChart() { ... }

    @property
    public DialogSheet ActiveDialog() { ... }

    @property
    public MenuBar ActiveMenuBar() { ... }
    ...
```

Working Prototype

- Template that projects any COM interface

```
alias Project!(raw.IXmlDocument) IXmlDocument;
```

- At compile time:
  - Generate a wrapper struct for each COM interface
  - Enumerate the methods of the COM interface
  - Generate a stub for each COM method
    - Determine stub name
    - Enumerate all the parameters of the COM method
    - Insert conversion/check code for each parameter

How it works

The method that generates the trampoline:

```
string ProjectMethod(T, string METHOD, string PTR)() pure
{
        string ProjectedName = …;
        string retval = "void", input, prologue, parameters, epilogue;
        foreach (Index, ParameterType; ParameterTypeTuple!(T,METHOD))
        {
                // Set retval, input, prologue, parameters, epilogue.
        }
        return retval ~ " " ~
                ProjectedName ~ "(" ~ input ~ ") { " ~
                prologue ~ " VERIFY = " ~
                PTR ~ "." ~ S ~ "(" ~ parameters ~ "); " ~
                epilogue ~ " } ";
}
```

How it’s used

```
// include the projection templates
import projected;
// include the "IDL"
import winrtxml;

// project the interface
alias Project!(raw.XmlDocument) XmlDocument;
```

All other interfaces are automatically projected *at compile time* as needed.

Output:

```
M:\>rdmd main.d
projecting interface IXmlDocument
projecting interface IXmlDocumentType
projecting interface IXmlNamedNodeMap
projecting interface IXmlNode
projecting interface IXmlNodeList
projecting interface IIterator
projecting interface IXmlDocument
projecting interface IXmlDomImplementation
projecting interface IXmlElement
projecting interface IXmlAttr
projecting interface IXmlDocumentFragment
projecting interface IXmlText
projecting interface IXmlComment
projecting interface IXmlProcessingInstruction
projecting interface IXmlEntityReference
projecting interface IXmlCDataSection

世界你好
hello
world
hello
world

M:\> _
```

“IDL”

```
interface IXmlDocument : IDispatch
{
    static immutable uuidof =
        uuid("F7F3A506-1E87-42D6-BCFB-B8C809FA5494");

    HRESULT get_Doctype(out IXmlDocumentType value);
    HRESULT get_Implementation(out IXmlDomImplementation value);
    HRESULT get_DocumentElement(out IXmlElement value);
    HRESULT CreateElement(in HSTRING tag, out IXmlElement element);
    ...
```

D’s magic features

- template
- mixin
- static if
- Compile-time reflection
- Tuples
- Compile-time function evaluation
- Native COM support

template

- Best described as “reusable pieces of code”
- Templates can have parameters
  - Like C++: typename, int
  - Unlike C++: alias, string, enum, tuple, *anything*
- Template can contain
  - Like C++: class, struct, method declarations
  - Unlike C++: declarations, statements, …

template

- Declaration:

```
template name(A,B) { … }
class classname(T) { … }
struct structname(T) { … }
T methodname(T)(…) { … }
```

- Instantiation:

```
name!(A,B).somefunc();
classname!(T) a;
int b = methodname!int(2);
mixin name!(A,B);
```

mixin

- Template mixin
  - Copy-paste, without the duplication:

```
mixin sometemplate!(A,B);
```

- String mixin
  - Most powerful feature… ever!
  - Compiler treats contents of string as code

```
mixin("int method(int x) "~
      "{ return x*x; }");
```

static if

- Control what gets compiled when
- Like #if and #ifdef
  - but can inspect types!
- Argument to static if must be constant
- Very useful when used with __traits

```
static if (__traits(compiles,
      (new T).Length)) { … }
```

Compile-time reflection

- Type inspection using __traits expression
- Get all members of an interface

```
__traits(allMembers, ISomeInterface)
```

- Get a type's name

```
auto m = somemethod.stringof;
```

- Get a type's mangled name

```
auto m = somemethod.mangleof;
```

(the mangled string contains all the parameters)

Tuples

- Compile-time sequence of elements

```
Tuple!(somealias, "somestring")
```

- Expression tuples

```
Tuple!("asdf", 3, 5.5)
```

- Type tuples (also known as *typelist*)

```
Tuple!(int, SomeClass, float)
```

- Can be iterated using *static* foreach

```
foreach (member;
  __traits(allMembers, IXmlNode))
```

Compile-time function evaluation

- C++ constant folding

  int x = 2*8; // will be evaluated at compile time
  int y = factorial(5); // …maybe

- Extreme constant folding

```
string ProjectMethod(T, string S, string P)() { … }
auto code =
   ProjectMethod(IXmlNode,”get_NodeName”,”ptr”)();
mixin(code);
```

The Project template

```
public struct Project(T, string CLSID = "")
{
     T ptr;                                  // the interface pointer (init'd to null)

     mixin ProjectInterface!(T, "ptr");      // project all the methods of the interface

     static if (is(T.requires))              // require any other interfaces? Project those also
          mixin ProjectRequiredInterface!(ptr,
                    NoDuplicates!(RequiresInterfaces!(T.requires)));

     static if (CLSID != "")                 // Activatable? Then we add a static operator ()
         public static typeof(this) opCall()
         {
             String strActivatableClassId;
             VERIFY = strActivatableClassId.Initialize(CLSID);
             IInspectable pInspectable;
             VERIFY = ActivateInstance(*strActivatableClassId, pInspectable);
             scope(exit) pInspectable.Release();
             typeof(return) _this;
             VERIFY = pInspectable.QueryInterface(&(T.uuidof), cast(void**)&(_this.ptr));
             return _this;
         }

     this(this) { if (ptr) ptr.AddRef(); }  // post-blit
     ~this() { if (ptr) ptr.Release(); }    // destructor
}
```

Done:

- Interface methods
- Required interfaces
- Activation
- Checks CLSID after activation
- Automatic `QueryInterface` (cached)
- Shows entry/exit from trampoline
- Transparent type conversions COM ↔ D

Type conversions

COM

- Failure HRESULT
- Success HRESULT != S_OK
- Last [out] parameter
- propget / propput
- BSTR
- T : IUnknown
- VARIANT
- SAFEARRAY
- IDispatch-based callback

D

- Exception
- Assertion failure
- return value
- Properties
- Native D wchar[]
- Project!T
- std.variant.Variant
- D array
- D delegate

To be done

- Use verbatim IDL or typelib metadata
  - idl2d is part of the visuald project
- Pre-compiled projection for intellisense
- More conversions (Enumerator, etc…)
- Enhanced COM support (coclass, dispinterface)

Prerequisites

- Create D-compatible “IDL”
- Reference D 2.0 compiler (DMD v2.048)
  - Free, not GPL
  - http://digitalmars.com/d/2.0/
- coffimplib tool to convert api-*.lib to OMF
  - Free
  - ftp://digitalmars.com/coffimplib.zip
- My files
  - http://dpxml-lio/d/
- Optional: VisualD Visual Studio plug-in
  - Free, not GPL
  - http://dsource.org/projects/visuald/