<http://www.ondotnet.com/pub/a/dotnet/2003/01/13/intromcpp.html>

**Introduction to Managed C++**

by [Sam Gentile](http://www.ondotnet.com/pub/au/1070)  
01/13/2003

Welcome to the first article in an initial series of three articles on Managed C++, or, to use its proper name, Managed Extensions for C++. (Note: I will use the terms "Managed Extensions," "MC++," and "Managed C++" interchangeably in this series.) Managed C++ often gets a bum rap in the world of .NET. Some of it is deserved, some is not, and some comes from just plain misunderstanding. Some people have even gone as far as terming it a "dead" language and of little use in a managed world filled with C# and VB. While it is certainly true that managed languages such as C# and VB have greatly increased developer productivity and simplified development in many areas (such as RAD UI development with Windows Forms), there are many areas where Managed C++ still is the best choice, or even maybe the only choice, in implementation language. These articles, and in particular this article, are designed to probe these questions and clear up some of the confusion that surrounds MC++, help you understand the advantages and disadvantages, and teach you some of the basics of the language so that you can put it use in your applications today.

**Target Audience**

A quick note: this article, and this series, assume that the reader is familiar with the basics of the .NET Framework (including the CLR) and has worked with "managed" languages such as C# and VB.NET. The reader may or may not be a C++ programmer. It helps, certainly, but adventuresome programmers are certainly welcome!

**But Isn't C++/MC++ a Dead Language?**

Due to the constant hype coming out of Microsoft and other places, there is a common *misconception* that Managed C++ and C++ itself are "dead" languages, and have no place, or part to play, in the world of the CLR. Nothing could be further from the truth. While it is absolutely true that C# is a "simpler" and highly productive language, it is not the be all and end all of languages. There are, quite simply, not only classes of applications that are more suitable for Managed C++, but also *classes of applications that can not even be created for .NET with any other language.*

Often, several languages are suitable for a given task. The choice of programming language can be influenced by the experience of the programmers, as well as aesthetic preferences. Indeed, this is one of the major advantages of the .NET platform, in that there is a lot of choice when deciding what programming language should be used for development. This is, of course, possible because all .NET language compilers are required to emit IL rather than native code. To a certain extent, this IL generated by a .NET compiler is more or less the same, which could lead one to the conclusion that the language used to create the managed type is irrelevant. Certainly, this is an impression that Microsoft drives home often -- people like me even claim "It's the Runtime Stupid!" But the fact of the matter is that this is not strictly true, as the C++ compiler performs some optimization on the IL it produces, resulting in code that performs better than code generated from the C# or VB.NET compilers. Some of these languages provide conveniences that result in extra IL. Are these two points important in your .NET application? It depends on the application.

Any good programmer, including any good .NET programmer, keeps a toolbox of languages and tools that most suit the problem at hand. Given the whole slew of languages available for .NET, why should one pick C++ to write .NET code?

**Advantages of Managed C++**

C++ is arguably the most powerful language ever invented. (And people do argue about things like this constantly!) It has always been a systems language, and it gives you the power and flexibility to produce truly flexible and innovative solutions. It is also one of the most widely used, and a first choice in implementing solutions from operating systems to office suites and most anything in between, resulting in the existence of billions of lines of C++ code, with millions more added every year. This same spirit has been carried over to MC++, where you have not only complete access to all of the features of the .NET Framework, but also the full power of the unmanaged language as well. To this point, MC++ is the only language where you can mix managed (.NET) code and unmanaged code in the same assembly, and even in the same source file.

Here are some specific advantages of MC++:

* The best performance of generated IL code because of both optimizations of the generated IL and less IL generated (as discussed in the previous section). This is specifically because MC++ is the **only** .NET compiler with a full optimizer back end, which is pretty much the same one that is used by the unmanaged compiler.
* MC++ is your language of choice if you want full control of the .NET environment:
  + Allows one to use all seven levels of CTS member access. C# allows only six.
  + Allows direct access to interior gc pointers, useful in a whole class of system applications such as system and .NET utilities.
  + Offers explicit control of expensive operations like boxing.
  + Supports multiple indexed properties on a type, unlike C#.
* MC++ is currently the only managed language that allows you to mix unmanaged and managed code, even in the same file. This leads to several other points:
  + Allows a developer to keep performance-critical portions of the code in native code.
  + Gives seamless access to all unmanaged libraries, such as DLLs, statically-linked libraries, COM objects, template libraries, and more.
  + Leverages existing investments in C++ programming skills and legacy C++ code.
  + Porting unmanaged code to .NET: MC++ allows you to take existing unmanaged code and compile it to managed code (with the /clr compiler switch and IJW).
  + Gives the ability to port code at one's own rate rather than re-write all at once.
  + Provides the easiest way to add .NET support to your existing native C++ Windows applications, by allowing you to bridge the gap between the two environments with as little work on your behalf as possible, and with the lowest performance penalty.
* MC++ is currently the only language that allows some form of multi-paradigm design and development with full support for generic programming and templates. This can lead to more options and better designs and implementations.

**Disadvantages of Managed C++**

* C++ is a more complex language than C# in both its syntax and areas where one could get into trouble. Since MC++ follows the C++ paradigm of "explicit is good", some MC++ constructs may seem really ugly. For simpler types of applications, and with certain types of developers, it may make more sense to use C#.
* Managed C++ code is non-verifiable, since C++ can perform unsafe operations. The implication of this is that MC++ code may not run in restricted environments that will not run code that is non-verifiable.
* Some minor features of the .NET platform are not supported yet, such as Jagged Arrays.
* IDEsupport is currently lacking, compared to other managed languages, since there's little or no designer support (but Everett will change this).

**What Exactly are Managed Extensions for C++?**

Managed Extensions for C++ are *extensions* to the Visual C++ compiler and language to allow them to create .NET code and enable access to the functionality of the .NET Framework. They include a set of *Keywords* and *Attributes* to extend the C++ language to work with, and generate, managed code. There are also some additional *Pragmas*, *Pre-processor Directives*, and *Options* for the *Compiler*, as well as some *Linker Options*. The first interesting thing to notice is that the Managed Extensions use C++ keywords and syntax, but they follow .NET rules for types and facilities. This creates, in effect, a language within a language.

The following sections show all of these for reference.

**Keywords**

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| [\_\_abstract](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__abstract.htm) | Declares a class that cannot be instantiated directly. |
| [\_\_box](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__box.htm) | Creates a copy of a \_\_value class on the common language runtime heap. |
| [\_\_delegate](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__delegate.htm) | Declares a reference to a unique method (a function pointer) of a managed class. |
| \_\_event | Declares an event method of a managed class. |
| \_\_finally | Declares a finally block associated with a try block. |
| [\_\_gc](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__gc.htm) | Declares a gc type. |
| [\_\_identifier](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__identifier.htm) | Enables the use of a C++ keyword as an identifier. |
| \_\_interface | Declares an interface. |
| [\_\_nogc](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__nogc.htm) | Declares a native C++ class that is not garbage-collected. |
| [\_\_pin](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__pin.htm) | Prevents an object or embedded object of a managed class from being moved by the common language runtime during garbage collection. |
| [\_\_property](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__property.htm) | Declares a property member for a managed class. |
| public, **protected**, and **private** | Determines the accessibility of specified types and methods outside of an assembly. |
| [\_\_sealed](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vchowfinal.htm) | Prevents a \_\_gc class from being a base class, or a method from being overridden by one in a derived class. |
| [\_\_try\_cast](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__try_cast.htm) | Performs the specified cast or throws an exception if the cast fails. |
| [\_\_typeof](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__typeof.htm) | Returns the System::Type of a given type. |
| [\_\_value](ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/vcmex/html/vclrf__value.htm) | Declares a value type. |

**Attributes**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
|  | Creates a user-defined attribute. |

**Pragmas**

|  |  |
| --- | --- |
| **Pragma** | **Description** |
| managed, unmanaged | Determines if code is compiled to MSIL or unmanaged code. |

**Preprocessor Directives**

|  |  |
| --- | --- |
| **Directive** | **Description** |
| #using | Imports metadata into a managed application. For more information, see "21.4 Importing Metadata with #using" or "21.5 Metadata as Binary Headers" in the Managed Extensions specification. |

**Compiler Options**

|  |  |
| --- | --- |
| **Option** | **Description** |
| /AI | Specifies a directory to search to resolve file references passed to the #using directive. |
| /clr | Compiles C++ and Managed Extensions for C++ source code to MSIL. |
| /FU | Forces the use of a file name, as if it had been passed to the #using directive. |

**Linker Options**

|  |  |
| --- | --- |
| **Option** | **Description** |
| /ASSEMBLYMODULE | Adds a MSIL module to the assembly of a project. |
| /ASSEMBLYRESOURCE | Adds a link to a managed resource from a project. |
| /NOASSEMBLY | Creates a MSIL module that is not an assembly by itself, but can be part of an assembly. |

**Your First Managed C++ Program**

Okay, ready to dive in? Let's look at the simplest possible program.

#using mscorlib.dll; // required for MC++

void main()

{

System::Console::WriteLine(S"Managed C++ Rocks!!");

}

This can either be built as a Visual Studio .NET Managed C++ Console Application or from the command line using the /CLR compiler option. To do this, you can open a command window using the VS.NET command window, as follows:

C:\Code\OReilly\HelloWorld>cl /CLR HelloWorld.cpp

Microsoft (R) C/C++ Optimizing Compiler Version 13.10.2292 for

.NET Framework

Copyright (C) Microsoft Corporation 1984-2002. All rights reserved.

HelloWorld.cpp

Microsoft (R) Incremental Linker Version 7.10.2292

Copyright (C) Microsoft Corporation. All rights reserved.

/out:HelloWorld.exe

HelloWorld.obj

C:\Code\OReilly\HelloWorld>HelloWorld

Managed C++ Rocks!!

The first statement of the program is a preprocessor directive, the #using statement. This directive tells the compiler to find and load the assembly *mscorlib.dll*, which contains the .NET BCL (Base Class Library, i.e. System). Console is the BCL class for basic text I/O and the WriteLine method prints out a string. The prefix S tells the compiler that this is a managed string literal.

**What Scenarios are They Good For?**

Although one could write all of their managed code in MC++, there are certain scenarios that lend either an advantage to using MC++, or require it altogether to accomplish the task. Interop Directly with Existing Unmanaged Code Native, or unmanaged code, is, of course, code that existed before .NET and cannot take advantage of any of the runtime's features, as it runs outside the CLR. One thing is certain: there are millions of lines of existing native C++ code and there are likely to be for some years to come. One of the areas that MC++ excels in, and is in fact unique in amongst the .NET languages, is the ability to take an existing unmanaged (C++) application, recompile it with the /clr switch, have it generate MSIL and then run under the CLR, unmanaged. This extraordinary feat is aptly termed "It Just Works (IJW)!" There are some limitations, but for the most part, the application will just run. The C++ code can consist of old-fashioned printf statements, MFC, ATL, or even templates!

IJW is one form of Interop. There are also COM Interop and Platform Invoke (P/Invoke). The important thing to note is that MC++ is unique amongst CLR languages in this ability to directly interoperate with native code. Managed Wrappers around Existing Native Code Building on IJW; not only will MC++ let you interoperate directly with native code, but it is unique in the ability of creating "managed wrappers." This combination of unmanaged and managed code allows developers to expose unmanaged C++ functionality to other managed languages like C#, Eifel.NET, and VB.NET. MC++ allows you to mix native and managed code even in the same file; this is extremely valuable. This allows developers to "port" a native application to a managed one, a portion at a time. Moreover, it allows the developer to choose which portions of the native functionality they wish to "expose" to managed clients through the managed wrapper. We have already mentioned that MC++ generates IL that performs better due to the ability of the MC++ compiler to optimize the generated IL. However, there are currently applications that, for performance reasons, will have to maintain portions in native code. Visual C++ .NET 2003 performs several new optimizations on native code, which further increases performance.

**What's Next?**

In this article, we have looked at some of the reasons why one may want to use Managed C++ (or not). Then we looked at what the Managed Extensions are. From there, we wrote our first MC++. I hope that I have at least interested you. In the next article of the series, we will look at the powerful facility of IJW (It Just Works) and how we can take existing unmanaged C++ code and compile and run it within the managed environment, without modification! The last article will focus on mixing managed and unmanaged code together.

[*Sam Gentile*](http://www.ondotnet.com/pub/au/1070)*is a well-known .NET consultant and is currently working with a large firm, using Visual C++ .NET 2003 to develop both unmanaged and managed C++ applications.*