How to translate C/C++ to C++/CLI

Questions? Contact Anton Rotter-Sieren at [Anton.D.Rotter-Sieren@usace.army.mil](mailto:Anton.D.Rotter-Sieren@usace.army.mil)

Table of Contents

[Managed References: 1](#_Toc493244616)

[Unmanaged References: 2](#_Toc493244617)

[Converting from Unmanaged to Managed: 4](#_Toc493244618)

[To convert a primitive single value pointer to a primitive type: 4](#_Toc493244619)

[To convert a char \* to a String ^: 5](#_Toc493244620)

[To convert a primitive unmanaged array into a primitive managed array: 5](#_Toc493244621)

[Here’s how to convert a primitive unmanaged array pointer into a primitive managed array. Note that the size of the array pointer has to be known before doing the conversion: 5](#_Toc493244622)

[Here’s how to convert an unmanaged string array to managed string array. Note that the size of the array pointer has to be known before doing the conversion: 6](#_Toc493244623)

[Convert a C Struct into a managed class: 6](#_Toc493244624)

[Converting from Managed to Unmanaged: 8](#_Toc493244625)

[Convert primitive managed ref into primitive unmanaged pointer: 8](#_Toc493244626)

[Convert a primitive managed array into a primitive unmanaged array: 8](#_Toc493244627)

[Convert managed string into char pointer: 9](#_Toc493244628)

[Convert a managed String ^ array into a char double star array: 9](#_Toc493244629)

[Other Useful Information 10](#_Toc493244630)

# Managed References:

Any objects that rely on the garbage collector to clean them up are considered managed references. Managed references are .NET objects, so they can be used in C# and VB.NET. Arrays and strings are the most common managed references, any managed class you make is also a managed reference.

All managed references require the karat (^) after their type.

Examples:

String ^ str;

array<int> ^ arr;

array<String ^> ^ strArr;

To initialize managed references, a call to gcnew is required. No karat is required for outermost after the call to gcnew

Examples:

String ^ str = gcnew String(“Hello World”);

array<int> ^ arr = gcnew array<int>(100); //makes an int array of size 100

array<String ^> ^ strArr = gcnew array<String ^>(100); //makes a String array of size 100

Managed classes require the ref keyword after visibility

Example:

public ref class MyClass

{

};

To make a class static, use the keywords abstract sealed

Example:

public ref class DSS abstract sealed

The ref keyword used in C# and VB.NET is replaced by the percent(%) sign. The % sign will go after the karat in managed references

In C#:

public void Foo(ref string str, int num, out double dub, ref float flo)

in C++/CLI:

public void Foo(String ^% str, int num, [Out] double % dub, float % flo)

# Unmanaged References:

Unmanaged references are the scary part about C/C++, which is pointers. Pointers themselves are just memory addresses. It’s important to note that unmanaged references require a call to delete after every new. An unmanaged reference uses the star (\*) to signify itself as an unmanaged reference. C# and VB.NET cannot use unmanaged references

Examples:

int \* intPtr; //a pointer to an integer

int \* intArrPtr; //a pointer to a integer array. Note how a pointer to an integer and an integer array is indistinguishable.

int intArr[10]; //an integer array

char \* str; //a pointer to a char array, which is actually a string.

char \*\* strArr; //an array of char pointers. In other words, an array of strings.

To initialize unmanaged references, a call to new is required. No star is required for outermost after the call to new. Also for every call to new, you should have a call to delete, to prevent memory leaks.

Examples:

int \* intPtr = new int;

delete intPtr;

int \* intArr = new int[100];

delete intArr;

char \* str = new char[12] //Hello World is 11 characters long, however c strings terminated by a null terminator, \0,

delete str;

char \*\* strArr = new char\*[100]; //an array of strings of size 100.

for (int i = 0; i < 100; i++)

strArr[i] = new char[10];

for (int i = 0; I < 100; i++)

delete strArr[i];

delete strArr;

As stated before, pointers are memory addresses, so to get the object they point to, you need to dereference the pointer. The dereference operator is also the star (\*). The star goes behind the pointer you are dereferencing. The square bracket operator for arrays also dereferences the pointer at a specific index.

Examples:

int a = 5;

int \* intPtr = new int;

\*intPtr = a; //the pointer has not changed addresses, but the value it’s pointing at has changed. It’s now 5

delete intPtr;

//now showing [] derefence below

int \* arr = new int[2];

arr[0] = 5;

arr[1] = 22;

You can also get the memory address of any variable using the address operator, which is ampersand(&). Using the & will return a pointer to the variable.

Example:

int a = 5;

int \* intPtr = &a;

\*intPtr = 6; //a will also be 6 now.

# Converting from Unmanaged to Managed:

Note: unmanaged and managed primitive types are interchangeable, no conversion needed.

## To convert a primitive single value pointer to a primitive type:

//setup

int \* intPtr = new int;

\*intPtr = 5;

//start conversion

int primitiveInt = \*intPtr;

delete intPtr;

## To convert a char \* to a String ^:

//setup

char \* strPtr = new char[12];

strcpy(strPtr,”Hello World”); //strPtr is now “Hello World”

//start conversion

String ^ str = gcnew String(strPtr); //str is now “Hello World”

delete strPtr;

## To convert a primitive unmanaged array into a primitive managed array:

//we will be converting an integer array

//setup

int intArr[10] = {0,1,2,3,4,5,6,7,8,9};

//start conversion

array<int> ^ managedIntArr = gcnew array<int>(10);

for (int i = 0; i < 10; i++)

managedIntArr[i] = intArr[i];

## How to convert a primitive unmanaged array pointer into a primitive managed array:

Note that the size of the array pointer has to be known before doing the conversion

//we will be converting an integer array

//setup

int arraySize = 10;

int \* intArrPtr = new int[arraySize];

intArrPtr[0] = 0;

intArrPtr[1] = 1;

etc…

//start conversion

array<int> ^ managedIntArr = gcnew array<int>(arraySize);

for (int i=0; i<arraySize;i++)

managedIntArr[i] = intArrPtr[i];

delete intArrPtr;

## How to convert an unmanaged string array to managed string array:

Note that the size of the array pointer has to be known before doing the conversion.

//setup

int arrLen = 2;

char \*\* strArrPtr = new char\*[arrLen];

strArrPtr[0] = new char[7];

strcpy(strArrPtr,”Hello ”);

strArrPtr[1] = new char[6]

strcpy(strArrPtr,”World”);

//start conversion

array<String ^> ^ managedStrArr = gcnew array<String^>(arrLen);

for (int i=0; i<arrLen;i++)

managedStrArr[i] = gcnew String(strArrPtr[i]);

//managedStrArr[0] is now “Hello “

//managedStrArr[1] is now “World”

for (int i=0; i<arrLen;i++)

delete strArrPtr[i];

delete strArrPtr;

## Convert a C Struct into a managed class:

Create a new managed class.

Have a pointer to the C Struct as a member of your class.

Then for each member of the C Struct, make a property

See example below:

Say we have the following C Struct:

typedef struct {

int num;

char \*name; //For all intents and purposes, let’s just say this member is read only

} myStruct;

We follow the steps above to make the following class:

public ref class MyStructWrapper

{

public:

myStruct \* theStruct;

~MyStructWrapper() //A C++/CLI destructor. Same as implementing Dispose()

{

delete theStruct; //To avoid memory leak

}

property int Num {

int get() { return theStruct->num; } //When accessing a pointer, you use an ->instead of .

void set(int val) { theStruct->num = val;}

}

property String ^ Name {

String ^ get() { return gcnew String(theStruct->name); }

}

};

# Converting from Managed to Unmanaged:

Some new information about converting from managed to unmanaged before we begin. Converting a managed reference is not a trivial matter, due to how the garbage collector can potentially move objects around in memory. Unmanaged C++ has no way of knowing if the garbage collector has moved the objects in x memory address to y memory address, so the way we deal with that is by marking the managed reference. The marker we use is called pin\_ptr.

## Convert primitive managed ref into primitive unmanaged pointer:

void Foo(int % num) //in C# this would be void Foo(ref int num)

{

//setup

pin\_ptr<int> numPinned = &num; //syntax of pin\_ptr is pin\_ptr<type> varName = address

//start conversion

int \* intPtr = numPinned;

}

## Convert a primitive managed array into a primitive unmanaged array:

//setup

array<float> ^ floatArr = gcnew float(2);

floatArr[0] = 0;

floatArr[1] = 1;

pin\_ptr<float> floatArrPinned = &floatArr[0]; //NOTE: the [0] is important here for arrays

//start conversion

float \* floatArrPtr = floatArrPinned;

## Convert managed string into char pointer:

Converting a managed string into a char pointer is also not a trivial process. The first thing that needs to be done is a Marshal of the managed string. To Marshal something means to convert it from managed to unmanaged. We call the function

IntPtr Marshal::StringToHGlobalAnsi(String ^ str)

Also, every time you marshal something, you must Marshal free it. You call the function

void Marshal::FreeHGlobal(char \* ptr);

So when you marshal a String ^, you get a IntPtr type, which must be cast into a char \*. Here’s how to do exactly that:

String ^ str = gcnew String(“Hello World”);

IntPtr marshalledStr = Marshal::StringToHGlobalAnsi(str); //marshall the String ^

char \* strPtr = static\_cast<char\*>(marshalledStr.ToPointer()); //convert to char \*

//do stuff with the char \*…

Marshal::FreeHGlobal(strPtr); //call this when you are finished with the char \* to prevent memory leaks

## Convert a managed String ^ array into a char double pointer array:

//setup

array<String ^> ^ managedStrArr = gcnew array<String ^>(2);

managedStrArr[0] = “Hello “;

managedStrArr[1] = “World”;

//start conversion

char \*\* strArrPtr = new char\*[2];

array<IntPtr> ^ marshalArr = gcnew array<IntPtr>(2); //Necessary to have this

for (int i=0; i<2;i++)

{

IntPtr marshalStr = Marshal::StringToHGlobalAnsi(managedStrArr[i]);

char \* strPtr = static\_cast<char \*>(marshalStr.ToPointer());

marshalArr[i] = marshalStr;

strArrPtr[i] = strPtr;

}

//do stuff with the char \*\*…

//time to free/delete stuff we don’t need anymore

for (int i=0; i<2;i++)

Marshal::FreeHGlobal(marshalArr[i]);

delete strArrPtr; //there’s no need to delete the marshal array as it is a managed array.

//Since the IntPtr in the marshal array is actually a unmanaged pointer, there’s no need to delete each char \* in the char \*\*, since the FreeHGlobal would take care of that for you. Do not delete both, as that would cause runtime problems.

# Other Useful Information

Whenever you see a const char \* as a parameter when converting C++ to C++/CLI, that means that the function guarantees the string will not be modified. It is okay to use a String ^ in your wrapper function.

However, if it is just a char \*, or any other pointer, it is best to make the parameter a reference in your wrapper function. Use String ^%

Example:

Converting c function

void foo(const char \* fileName, char \* newFileName, float \* floats, int sizeOfFloats)

In C++/CLI, you’d convert that to

void Foo(String ^ FileName, String ^% NewFileName, float % floats, int sizeOfFloats)

Also, whenever converting a char \* (not a const char \*) into a String ^, you must reassign the String ^ to the new value

Example:

Converting c function

void foo(const char \* fileName, char \* newFileName, float \* floats, int sizeOfFloats)

C++/CLI conversion below

void Foo(String ^ FileName, String ^% NewFileName, float % floats, int sizeOfFloats)

{

pin\_ptr<float> floatsPinned = &floats[0];

IntPtr marshalToCharStar1 = Marshal::StringToHGlobalAnsi(FileName);

IntPtr marshalToCharStar2 = Marshal::StringToHGlobalAnsi(NewFileName);

char \* ptrToFileName = static\_cast<char \*>(marshalToCharStar1.ToPointer());

char \* ptrToNewFileName = static\_cast<char \*>(marshalToCharStar2.ToPointer());

foo(ptrToFileName, ptrToNewFileName, floatsPinned, sizeOfFloats);

NewFileName = gcnew String(ptrToNewFileName) //Pay attention here what we do

Marshal::FreeHGlobal(marshalToCharStar1);

Marshal::FreeHGlobal(marshalToCharStar2);

}