March, 2006

TMurgent Technologies

Moving to Visual Studio 2005

TMurgent Developer Series

White Paper by Tim Mangan
Founder, TMurgent Technologies
March, 2006

Introduction

In late 2005 Microsoft released *Visual Studio 2005*. After porting several projects from *Visual Studio 2003,* we have some experience in that process. These projects were all C++ based (unmanaged code) and includes system and application programs. This paper discusses issues that ISVs face in deciding whether/when to port their products to the 2005 version.

Reasons for Porting

Often it is not if you port, but when. Software Developers typically choose to upgrade their tools at the beginning of a major development cycle. Even if keeping up-to-date with the latest tools from Microsoft does not help in the current project/release, it is generally a good idea to be using the latest tools so that you can quickly attack new opportunities in the future. With VS2005, we have seen a few reasons that ISVs might consider when deciding "when".

The first is to improve the stability of your software. With VS2005 comes a new version of the C runtime (msvcr80). In the new runtime, Microsoft has improved buffer protection of their code. Primarily this has to do with strings, but anytime you pass a buffer into a runtime (like wcscpy) you now will now also pass in a size indicator so that the runtime can actually know the size of the buffer and will not over-write it. This will cause you to make changes in your code as part of the port – but this turns out to be fairly easy to do (and is discussed later in this paper). Preventing buffer overruns not only will make your software more stable, but also improves security of the end-user system. Just like the OS components, your software is also vulnerable to buffer overrun attacks by hackers.

The second reason to port is developer productivity. While porting takes time, the new Integrated Developer Environment (IDE) will make developers more productive. Take a look at the information Microsoft provides on MSDN[1] for details, but here are our highlights related to improving the productivity of a developer working on an existing project that we love:

- IntelliSense works better now, resolving more expressions than before.
- New "Code Definition Window" which automatically locates the definition of an object you click on in a C++ or .h source file. It opens that file with that definition for viewing in the CDW window displaying the exact lines you wanted. You can scroll up/down that file as well to look for other definitions you might want. This works for both Microsoft defined expressions (e.g. "HRESULT") or you own.
- Better use of color. For years you have needed add-on products for this, but now the IDE natively uses color to give the developer more clues. For

[1] For example, "*What's New in Visual Studio 2005*" http://msdn2.microsoft.com/en-us/library/88fx1yx0(VS.80).aspx however this link may change.

example, if a #ifdef does not resolve in the current build (eg release versus debug) the unused code appears in grey.

- Improved help. We are using dynamic help more and more often now. But what is really great is how the help search function is more integrated with the world. Ask for a help search on a topic and you get results organized by 4 sources, Local MSDN, MSDN online, associated 'codezone' community groups (like Code Project and CodeGuru), and the Microsoft hosted user forums.
- Code snippets. This gives you a window in which code sections you frequently need can be at your fingertips. If you are into patterns this is a natural for you. I am not such a great fan of this feature, but some people just can't live without it once they have it.
- Settings export and import. You know all those secret things you have to do to set up a new Visual Studio install to work with your project? Now you can export those settings from an existing setup and import them into a new one.
- Deployment. The new "Click once" deployment will be of interest to some companies.

The third reason to port is to upgrade your source control system. If you are using CVS, you probably would be hesitant to move to Team Foundation Server (TFS) and I wouldn't try to talk you into that – so this might not be important to you. But if you are using Visual Source Safe (VSS) you will want to upgrade to TFS. TFS is more flexible, better integrated, and more stable than VSS ever way. Just to get off VSS may be worth the porting efforts! Microsoft has been looking at the competition and TFS is catching up.

The final reason is to access all the new stuff. While not important in a port, Microsoft is making so much progress in so many areas. If your product is big into graphical design, the edition for graphical artists (code name was Sparkle, but they are renaming it) will be a must.

What It Takes

The effort of porting is not too daunting, especially if performed at the beginning of a stable product and completed before the temptation to add in new functionality begins.

Visual Studio will convert existing project and solution files for you. We have seen this process to be accurate and (thus far) flawless. Conversion is one way and you can't go back. The conversion also analyzes the code and provides you with a nice output explaining what it did and areas that you need to manually address.

The new runtime will cause lots of warnings in your existing code. Microsoft has depreciated all of the unsafe interfaces, such as "wcscpy" and replaced them with new versions that have "_s" appended to the name. You can just build with the warnings (there is a compiler option to disable these) and things should work as before. You really should update to the new versions, however, so that you gain the benefits of improved stability and security. Here is a typical example of what it takes:

Old code:

```
wchar_t foo[256];
wcscpy(foo, L"Here is something!");
```

New code:

```
wchar_t          foo[256];
wcscpy_s(foo, sizeof(foo)/sizeof(wchar),  L"Here is something!");
```

With Unicode you need to be careful how each runtime method expresses the size. For example most Unicode string methods expect to see the number of wide characters, not number of bytes. Also one must be careful about not doing a sizeof(pointer). We have identified the 20 most common methods that developers will need to update. In many cases these new methods also return an errno_t to tell you if a buffer size issue occurred. We have found that the typical behavior is that these string methods will copy what they can, providing a null termination when they return an error.

While there are quite a few of these runtime methods you will have to fix up, fortunately the compiler warnings not only point you to the lines to be changed but also tell you the name of the replacement function. And obviously if you already exclusively use the ATL based string methods you may not have these issues.

The biggest issue is then your code does not know the size. This will force you to work back from this line of code to the source of the buffer itself. For example:

```
bool OddClass::method( PVOID pBuffer)
{
        wcscpy((wchar_t)pBuffer, L"Here is something risky!");
}
```

Needs to become:

```
Bool OddClass::method( PVOID pBuffer, size_t BufferInWchars)
{
        wcscpy_s((wchar_t)pBuffer, BufferInWchars,
                    L"Here is something less risky!");
}
```

And, of course, you will need to locate calls to method and fix them. I suppose you could do like Microsoft and make the new version "method_s" and even depreciate the old method yourself if you want.

The other big change is with time. The old runtime time structures will have a Y2K-like issue in less than 30 years. So Microsoft upgraded the time from 32 to 64 bits and now handles things through to the next century. The necessary changes to your code are also warned by the compiler in a similar way and need a similar amount of effort as the changes to string methods.

Prep for x64. As long as you are porting, even if you are not now porting to x64, you should look for upcoming issues there. By default, VS2003 did not warn about 64-bit portability issues but VS2005 will. Primarily you want to watch out for uses of a ulong for size information. Most vendor code is filled with size information passed around as a 32-bit unsigned int – which is the same size as a size_t type for the 32-bit compiler. But when you build for x64, size_t (and pointers, but not integers) will increase from 32 to 64 bits. The VS05 compiler will warn about the size mismatch and you could ignore it for now, but this is probably a good time to convert the types to remove the warnings and be more ready for x64 down the road. We will have another white paper on porting to x64 in the future.

The last consideration is really that of deployment. Moving to VS2005 and the new runtime means that target machines need the runtime components. Unlike 7.0, the 8.0 components are side-by-side assemblies so this takes more care. Also, being new it is far less likely that your customers already have these new assemblies. Microsoft provides good documentation on how to handle this. In your test lab it can be as simple as installing the new .NET 2.0 Framework on the test machines.

Conclusions

Hopefully, if you develop C++ applications this paper covers the primary issues you need to consider when deciding on whether/when to upgrade to VS2005. Our experiences may tend to be more kernel/system oriented than your projects and additional considerations may be needed that are not covered here.

About TMurgent Developer Series White Papers

At TMurgent, we help Software vendors with development issues, especially when it comes to system performance, management, and working with Terminal Services. Recent projects include helping companies make their products more Terminal Services aware, kernel interfacing, and porting to VS2005 and x64 platforms. We provide education to the developer community via the "*TMurgent Developer Series*" white papers to advertise our services. Please visit our website at www.tmurgent.com for more information.