Windows® 10 IoT Core Native Remote Debugging

By Sean D. Liming and John R. Malin
Annabooks – www.annabooks.com

November 2016

Two application types are support in Windows 10 IoT Core: Universal Windows Platform (UWP) and native console applications C/C++. Remote debug support (VSGraphicsRemoteEngine.exe) for UWP is part of the core OS. Native Remote Debugging for C/C++ applications is not included since Visual Studio and Windows 10 will update at different intervals. In this paper, we will explore how to add native remote debugging, system setup for remote debugging, and a few items to help with native code development.

Deploy the Remote Debug Files

The remote debug tools come with the installation of Visual Studio 2015 and higher. Make sure you include Phone development during the installation. Windows 10 supports different processor architectures, ARM and Intel Architecture (x86 and x64). The binaries are found at this path:

C:\Program Files (x86)\Common Files\Microsoft Shared\Phone Tools\14.0\Debugger\target

With separate folders for the supported processor architectures:

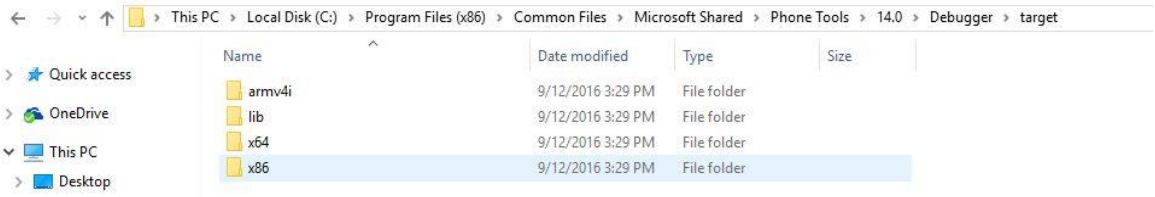


Figure 1 - Different Binaries for Processor Architecture

Only use the binaries that go with your target's processor architecture.

Note: if Unified Write Filter (UWF) is available in the image, make sure UWF is disabled. You can check if UWF is enabled, by running `UWFMGR get-config` from a remote PowerShell session. If UWF is enabled, disable UWF and reboot the target.

Here are the steps for putting the binaries on your target platform:

1. Make sure your target and host development system are on the same network.
2. Boot the target system to Windows 10 IoT Core.
3. On the host system, start the IoT Dashboard application.
4. In the My Devices section, right click on your target's IP address, and select Open Network Share from the context menu.

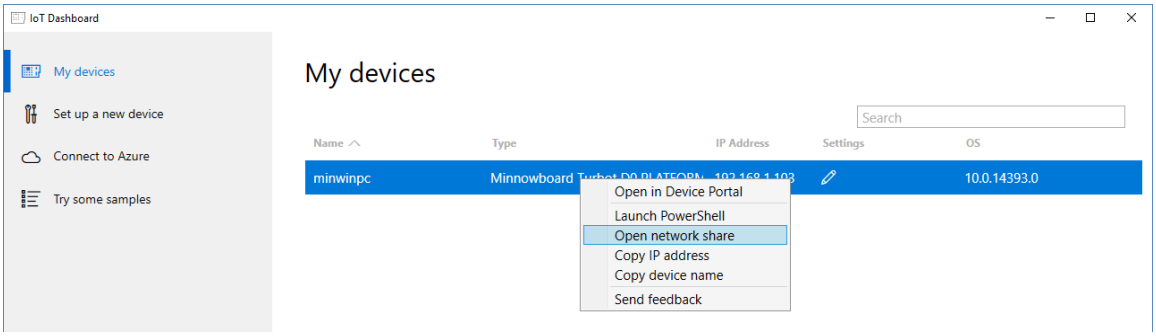


Figure 2 - Opening Network Share

5. Enter the credentials to access the remote target, and a file explorer window appears showing the remote target's C drive.
6. In the file explorer window, create a folder called RDBG.
7. Copy all the debug files for the target's processor architecture to the \RDBG folder on the target.

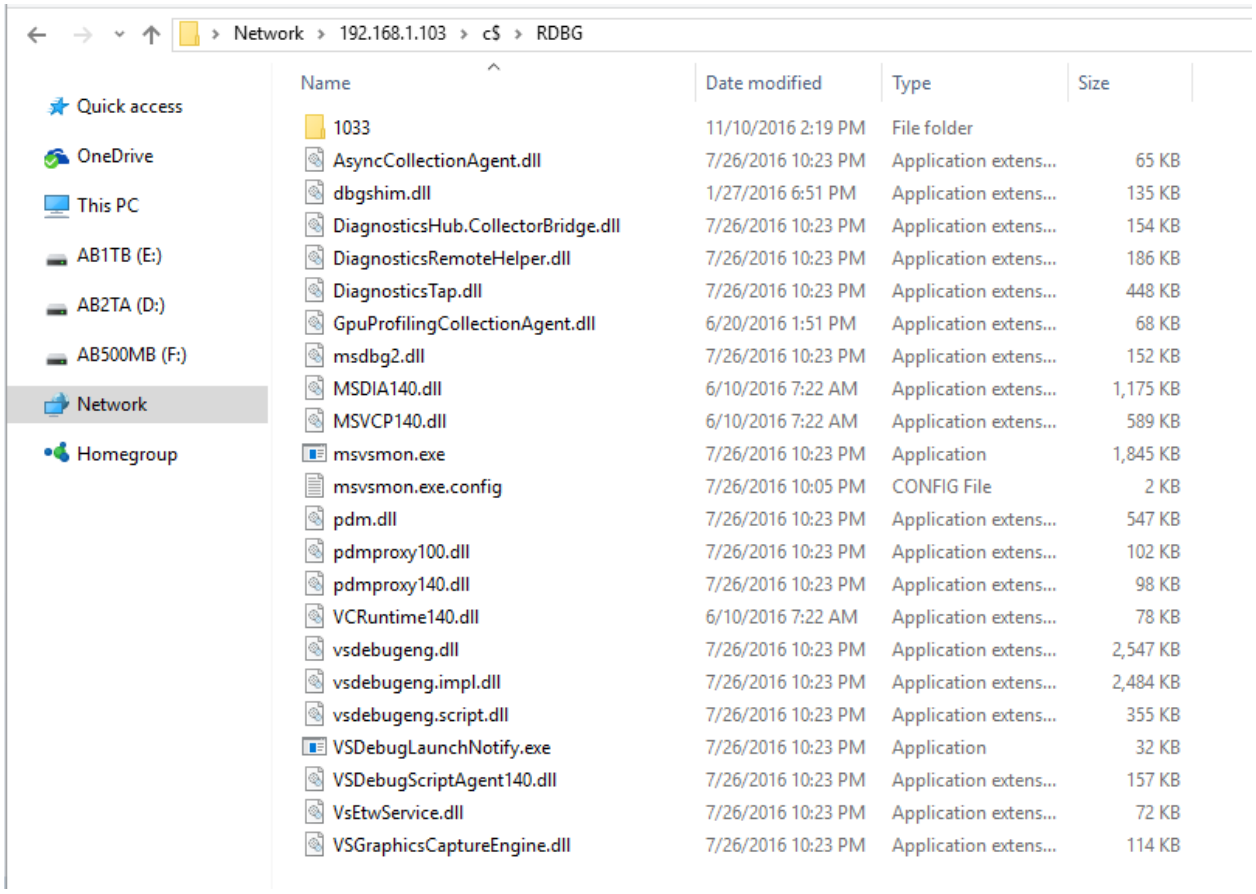


Figure 3 - New RDBG Folder and Remote Debug Files

Msvsmon.exe should be in the root of \RDBG folder.

Configuring the Firewall and Starting the Debugger

The firewall is enabled by default, thus the next step is to configure the firewall. Msvsmon.exe is the remote debug utility. You can configure the firewall via PowerShell remoting.

1. From the IoT Dashboard, right click on the target's IP address and select Launch Power Shell.
2. A PowerShell window opens, and you will have to enter the target's credentials again.
3. There are two options to configure the firewall.The first is to create a rule that allows msvsmon.exe to allow communications to come through the firewall:

```
netsh advfirewall firewall add rule name="Remote Debug" dir=in action=allow
program="C:\RDBG\msvsmon.exe" enable=yes
```

The second option is just to disable the firewall:

```
netsh advfirewall set allprofiles state off
```

4. Enter either of these in the PowerShell window and hit enter. You might want to create a .cmd file with either command if you ever have to run the remote debugger in a new image.

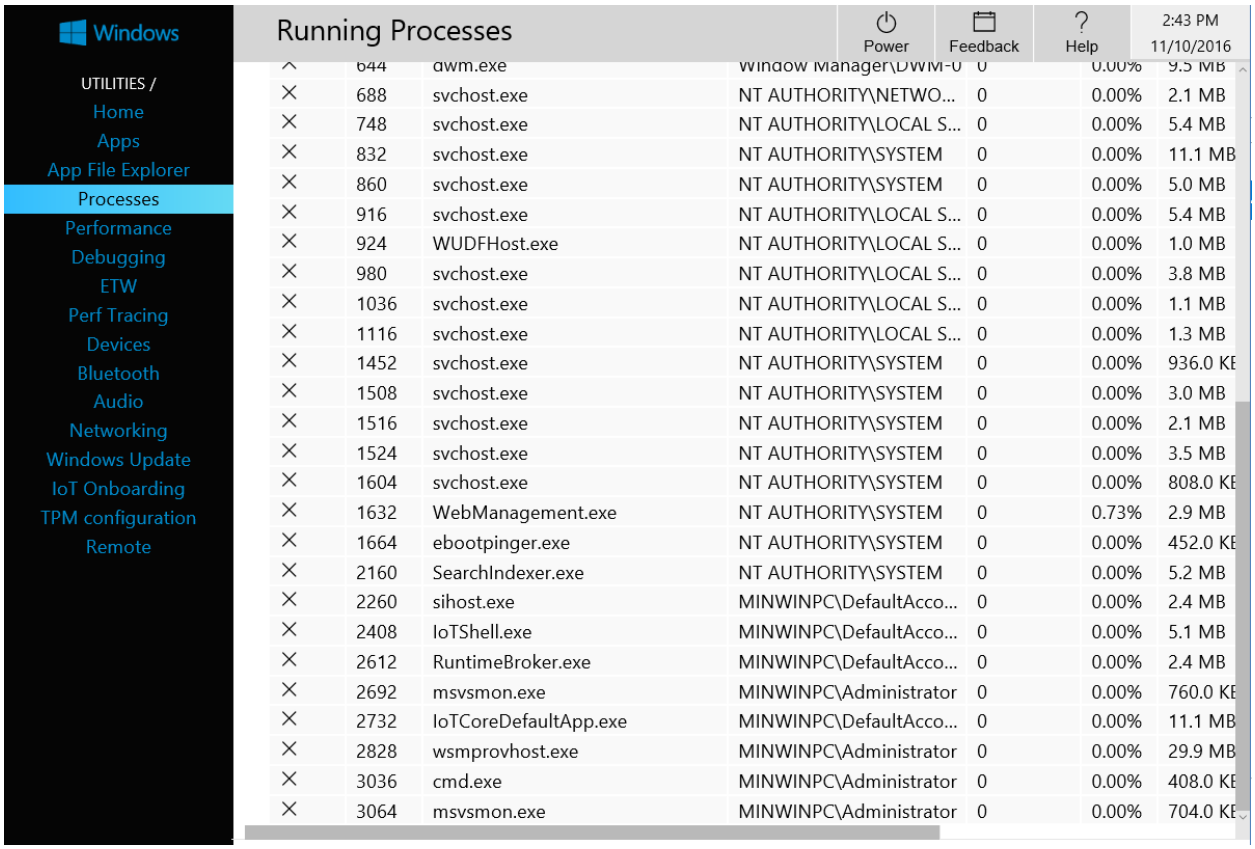
The next step is to start the debug session

5. In the PowerShell window, change directory to the \RDBG folder.
6. Enter the following to start the debugger:

```
msvsmon.exe /nowowwarn /noauth /anyuser /nosecuritywarn /timeout:36000
```

You might want to create a .cmd file with the command if you ever have to rerun the remote debug. The .cmd file would be best placed in the \RDBG folder.

7. You can see the debugger running in the target image, in IoT Dashboard, right click on your target's IP address and select Open in Device Portal.
8. An Internet Explorer or EDGE window will open. You will have to enter the credentials for the target.
9. In the Device Portal, click on Processes. You should see msvsmon.exe running.



644	dwm.exe	Window Manager\DWM-0	0	0.00%	9.5 MB
688	svchost.exe	NT AUTHORITY\NETWO...	0	0.00%	2.1 MB
748	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	5.4 MB
832	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	11.1 MB
860	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	5.0 MB
916	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	5.4 MB
924	WUDFHost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	1.0 MB
980	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	3.8 MB
1036	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	1.1 MB
1116	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	1.3 MB
1452	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	936.0 KE
1508	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	3.0 MB
1516	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	2.1 MB
1524	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	3.5 MB
1604	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	808.0 KE
1632	WebManagement.exe	NT AUTHORITY\SYSTEM	0	0.73%	2.9 MB
1664	ebootpinger.exe	NT AUTHORITY\SYSTEM	0	0.00%	452.0 KE
2160	SearchIndexer.exe	NT AUTHORITY\SYSTEM	0	0.00%	5.2 MB
2260	sihost.exe	MINWINPC\DefaultAcco...	0	0.00%	2.4 MB
2408	IoTShell.exe	MINWINPC\DefaultAcco...	0	0.00%	5.1 MB
2612	RuntimeBroker.exe	MINWINPC\DefaultAcco...	0	0.00%	2.4 MB
2692	msvsmon.exe	MINWINPC\Administrator	0	0.00%	760.0 KE
2732	IoTCoreDefaultApp.exe	MINWINPC\DefaultAcco...	0	0.00%	11.1 MB
2828	wsmprovhost.exe	MINWINPC\Administrator	0	0.00%	29.9 MB
3036	cmd.exe	MINWINPC\Administrator	0	0.00%	408.0 KE
3064	msvsmon.exe	MINWINPC\Administrator	0	0.00%	704.0 KE

Figure 4 - Running Processes

The system is ready for a native remote debug session.

Visual Studio Project Settings

Make sure that you have installed the IoT Core project Templates. The templates add the options to create Windows IoT Core applications. For C++ applications, there are three project types. The Blank Windows IoT Core Console Application creates a simple Hello World application that you can edit to create your own console application.

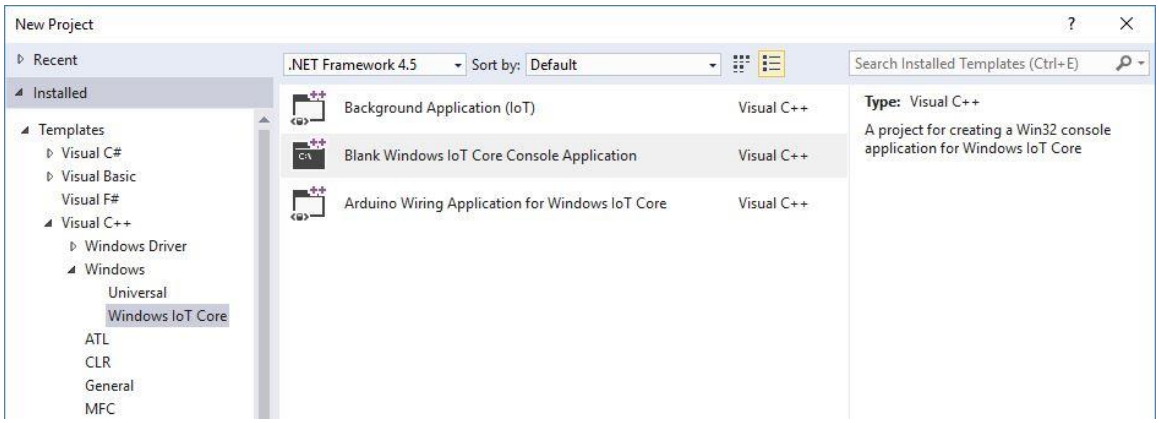


Figure 5 - Creating Console Application

The final step is to configure the C/C++ project debug settings in Visual Studio:

1. With your project open in Visual Studio, select Project-><Project name> Properties…
2. Select Debugging under Configuration Properties.
3. Set the Debugger to Launch: “Remote Windows Debugger”.
4. For both Active and Debug configurations, fill in the following information:

    - Remote Command: c:\<path>\Application name.
    - Working Directory: c:\<path>.
    - Remote Server Name: IP address of the target.
    - Connection: Remote with no authentication.
    - Debugger Type: Native only.
    - Deployment Directory: c:\<path>.
    - Deploy Visual C++ Debug Runtime Libraries: No.

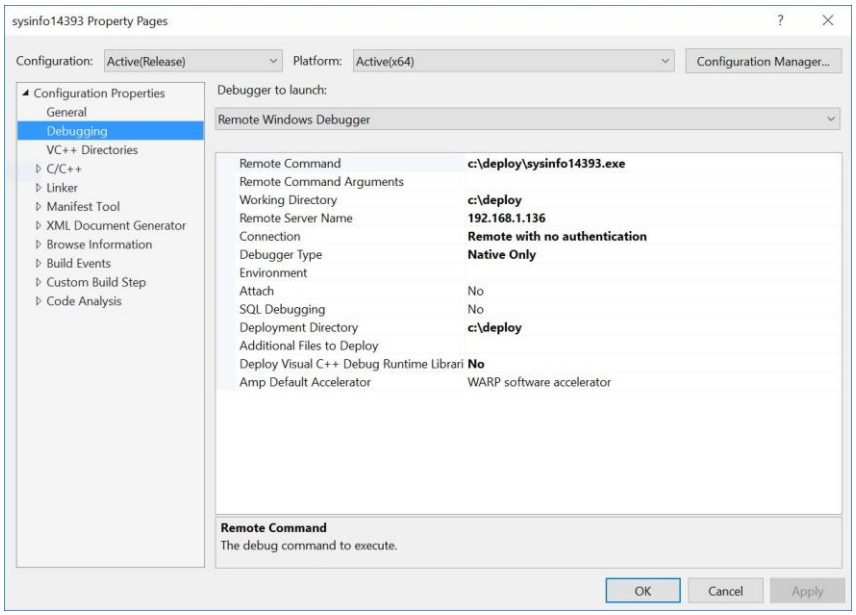


Figure 6 – Configuring the Project for Remote Debug

5. The final step is to make sure Deploy is enabled. From the menu, Build->Configuration Manager…
6. Make sure Deploy is checked for both Retail and Debug.

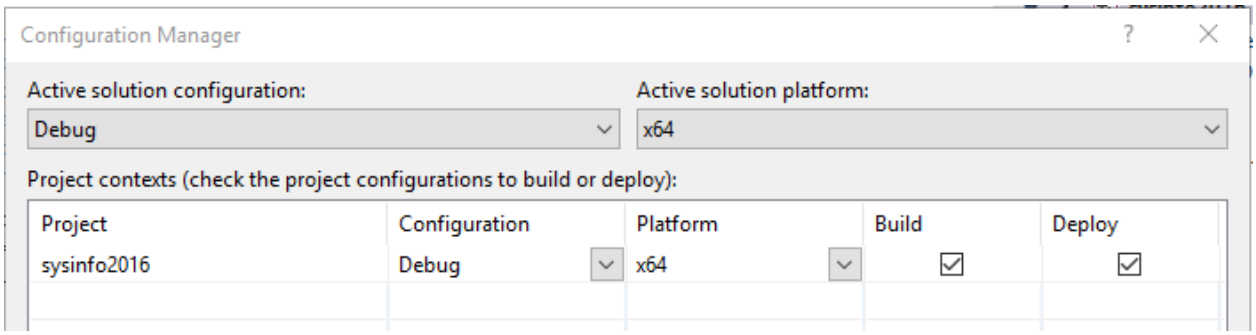


Figure 7 - Enabling Deployment

You can now deploy and remotely debug your native application. If your code sends any output to the console, the output will be displayed in PowerShell

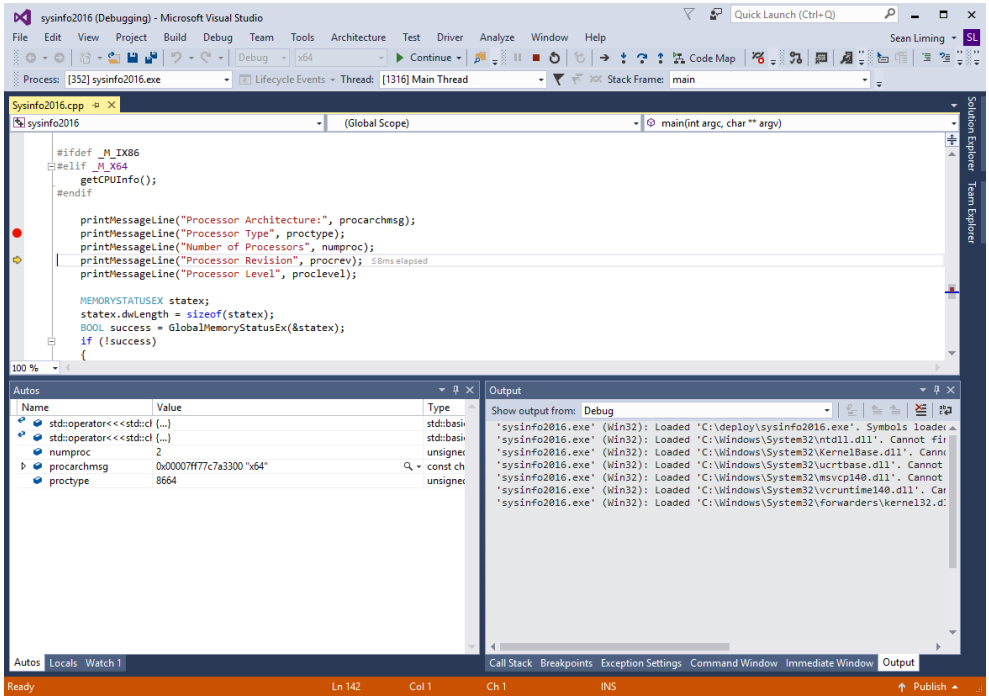


Figure 8 - Visual Studio Remote Debugging

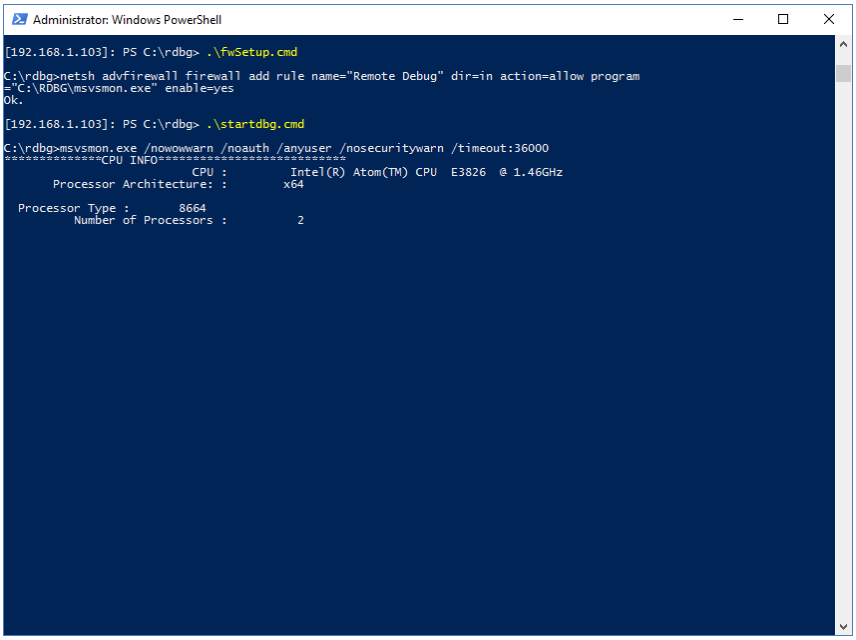


Figure 9 - Application Console Output

Processor Directives

UWP applications are design to run on different processor architectures by simply changing the target processor in Visual Studio. This allows you to create one program that target's different platforms. The same can be done for native applications, but you must address API calls that are unique to each processor architecture.

The processor directives (_M_IX86, _M_X64, and _M_ARM) can be put into your project to select code that is only for a specific processor architecture. For example, if there is code that will run on x86 and x64 but not on ARM, you would structure the processor directives as follows:

```
#ifdef _M_IX86
#elif _M_X64
      <<< your code here>>>
#endif
```

If there is code only for ARM, you would structure the processor ARM directive as follows:

```
#ifdef _M_ARM

      <<< your code here>>>
#endif
```

When you select the target processor architecture, the code with the other processor directives will be greyed out.

Summary: Going Native

There are many programming options available for IoT Core, and for the many hardcore C/C++ developers, C/C++ is not going away any time soon. The ability to remote debug native applications allows those familiar with programming in C/C++ the ability to create solutions using IoT Core.