Universal Drivers Lab

Deep Dive on Universal Driver Development – Version 2

***Abstract*:**

Let us get coding! Take all the concepts you have learned about Universal Drivers and put them in action. Join us for a guided hands-on session where we deep dive into building Universal Drivers. You will learn how to write a Universal Driver, create a componentized and declarative driver package and build a UWP app that communicates with the driver.

In addition to the content from last time, this version covers the new RS3 feature that enables the delivery of a store app along with a driver.

Intended Audience: IHVs, OEMs, Driver Developers

***Prerequisites*:** The lab assumes a working knowledge of Microsoft Windows drivers and C++ development.

***Install and configure*:** You will need:

* Step 1: Install Windows 10 Build 16299 or greater.
  + To obtain this build, you must sign up for the Windows Insider Program at: <http://insider.windows.com/> and select “Getting Started”.
  + Download and Install the latest Windows 10 Insider Preview build at: <https://www.microsoft.com/en-us/software-download/windowsinsiderpreviewiso>
  + **NOTE:** The lab and sample are written assuming an x64 device and OS are being used.
* Step 2: Install VS2017 (Community Edition is sufficient) – <https://www.visualstudio.com/downloads/>
  + During Setup, be sure to include the following Workloads:
    - “Universal Windows Platform development”
      * Make sure the option is selected in the Optional list in the summary – “**C++ Universal Windows Platform tools**”
    - “Desktop development with C++”

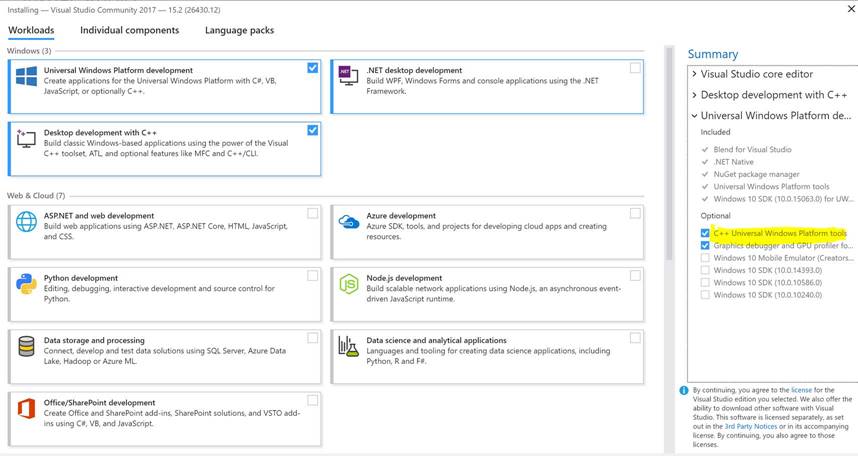


Figure 1: Visual Studio options

* Step 3: Install the Windows 10 Fall Creator’s Update SDK (Build 16299)
  + <https://developer.microsoft.com/en-US/windows/downloads/windows-10-sdk>
* Step 4: Install the Windows 10 Fall Creator’s Update WDK (Build 16299)
  + <https://developer.microsoft.com/en-us/windows/hardware/windows-driver-kit>
  + **NOTE:** SDK and WDK versions MUST match
  + **NOTE:** OS build MUST be greater than or equal to WDK version.
  + Make sure to select the option to integrate with Visual Studio at the end.
* Step 5: Enable test signing on the device: – <https://docs.microsoft.com/en-us/windows-hardware/drivers/install/the-testsigning-boot-configuration-option>
  + For a machine that has Secure Boot and BitLocker:

1. In Windows 10, suspend BitLocker -> reboot
2. Enter BIOS to disable Secure Boot -> save and exit to enter OS (BitLocker will be automatically resumed)
3. In Windows 10, suspend BitLocker -> edit BCD to enable test-signing -> reboot
4. Enter Windows 10, test-signing is now enabled.
   * For a machine that has Secure Boot and no BitLocker:
5. Enter BIOS to disable Secure Boot -> save and exit to enter OS
6. In Windows 10, edit BCD to enable test-signing -> reboot
7. Enter Windows 10, test-signing is now enabled.

Disable / Enable secure boot - <https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/disabling-secure-boot>

Suspend / Resume BitLocker - <https://technet.microsoft.com/en-us/library/ee424315(v=ws.10).aspx>

* + **NOTE:** If it says “Test Mode” in the bottom right of the desktop, testsigning is properly enabled.

Copyright (c) 2017 Microsoft. All rights reserved.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Contents

[Lab objective 5](#_Toc499049172)

[Exercise 1: Understanding Hardware Identity 5](#_Toc499049173)

[Task 1: View the Hardware ID for your graphics device 5](#_Toc499049174)

[Task 2: Install a demo dev node for this lab 7](#_Toc499049175)

[Exercise 2: Creating a Basic Driver Package 9](#_Toc499049176)

[Task 1: Open and build the supplied sample solution 9](#_Toc499049177)

[Task 2: Add the driver so it is installed by the INF 9](#_Toc499049178)

[Task 3: Add some registry values 10](#_Toc499049179)

[Task 4: Install your new driver 10](#_Toc499049180)

[Exercise 3: Add a service 12](#_Toc499049181)

[Task 1: Create a Service 12](#_Toc499049182)

[Task 2: Add service installation logic to the INF 12](#_Toc499049183)

[Task 3: Build and Install 14](#_Toc499049184)

[Exercise 4: Add a user interface with a Hardware Support App (HSA) 15](#_Toc499049185)

[Task 1: Build an App 15](#_Toc499049186)

[Task 2: Add a custom capability to the App 16](#_Toc499049187)

[Task 3: Grant access to the driver’s RPC server with the capability 17](#_Toc499049188)

[Task 4: Build and Install the new Toaster Driver 17](#_Toc499049189)

[Task 5: Build and Deploy the Toaster Support App 18](#_Toc499049190)

[Exercise 5: Add Componentized Customizations including HSA reference 18](#_Toc499049191)

[Task 1: Create Extension INF Project 18](#_Toc499049192)

[Task 2: Add a new INX file to the project 19](#_Toc499049193)

[Task 3: Add Version Information 19](#_Toc499049194)

[Task 4: Add a unique ExtensionId 19](#_Toc499049195)

[Task 5: Add some installation information 19](#_Toc499049196)

[Task 6: Add some customized registry keys 20](#_Toc499049197)

[Task 7: Add a SoftwareComponent identity for our HSA 20](#_Toc499049198)

[Task 8: Build and Install 20](#_Toc499049199)

[Exercise 6: Create a Component INF to deliver the HSA 22](#_Toc499049200)

[Task 1: View your new component device 22](#_Toc499049201)

[Task 2: Create Component INF Project and add a new INX file to the project 22](#_Toc499049202)

[Task 3: Add Version Information and installation information 23](#_Toc499049203)

[Task 4: Use the AddSoftware directive 23](#_Toc499049204)

[Task 5: Add the App PFN 24](#_Toc499049205)

[Task 6: Build and Install 24](#_Toc499049206)

[Exercise 7: Validate it is Universal 25](#_Toc499049207)

[Task 1: Run INFVerif /u 26](#_Toc499049208)

[Task 2: Run API Validator 26](#_Toc499049209)

[Conclusion 26](#_Toc499049210)

# Lab objective

Participants will learn about the capabilities of Microsoft Windows to develop, package, and install Universal Drivers for Windows 10 and newer operating systems. The tutorials for this lab will be presented as hands-on tasks.

# Exercise 1: Understanding Hardware Identity

All devices on a system are described to the Windows Operating System using a Hardware ID (HWID). HWIDs are created and enumerated by a bus driver. For example, the PCI bus enumerates HWIDs for all its child devices.

## Task 1: View the Hardware ID for your graphics device

* Open Device Manager (open a run dialog, run “devmgmt.msc”)
* Expand “Display adapters”
* Right click on the child node and select “Properties”
* Click on “Details”
* Select “Hardware Ids” property

You will see a list of HWIDs that represent the graphics adapter on your system. These are ordered from most specific to least specific.

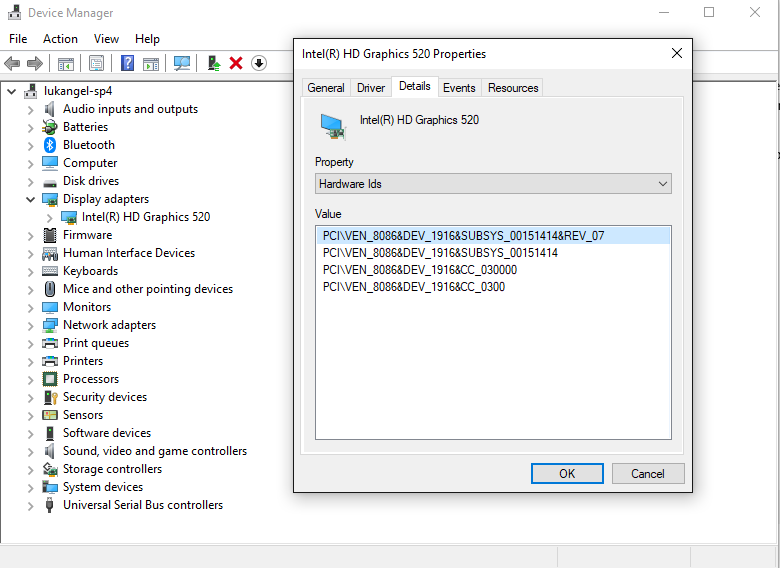


Figure 2: Hardware IDs for a Graphics Adapter

There are also Compatible IDs which represent even less specific device identities supported by the devices. For the same device, you can find these listed under the Compatible IDs property.

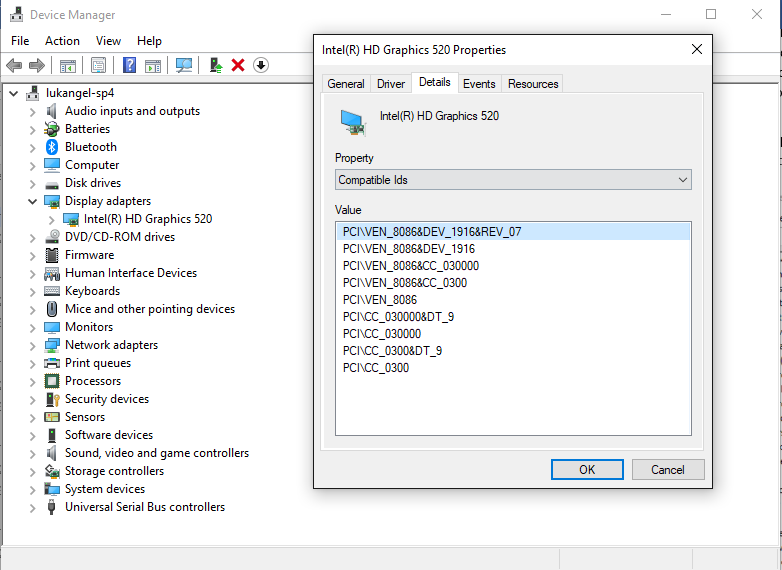


Figure 3: Compatible IDs for a Graphics adapter

The most specific Hardware ID will likely represent the device on a specific system. In the example of our Graphics Adapter, VEN refers to Intel, DEV refers the HD Graphics 520, and the SUBSYS represents the Surface Pro 4 on which this device is connected. You will notice as you go down the list of Hardware IDs and Compatible IDs, they get less and less specific. The second Compatible ID, for example, only refers to VEN Intel and DEV HD Graphics 520. The very last Compatible ID, “PCI\CC\_0300” is the PCI class code for VGA display controller. This is what the inbox driver matches on when no other driver exists.

## Task 2: Install a demo dev node for this lab

Now let us add a virtual device of our own to use while we work through the rest of this lab.

* Extract the supplied lab content to “C:\”
  + Right-click on “WinHEC 2017 Lab.zip” and select “Extract All…”
* Open an Administrator Command Prompt
* Navigate to “C:\WinHEC 2017 Lab\PlugInToaster\”
* Run “plug.exe”
* Open Device Manager
* Validate that there now exists an Unknown device

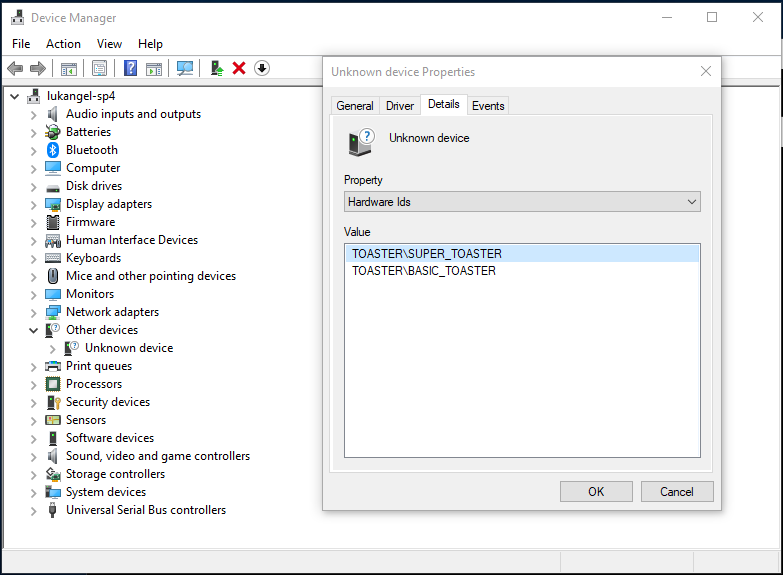


Figure 4: Our new device!

Notice that this device exposes a hardware id for the generic “BASIC\_TOASTER” as well as the hardware id for this specific “SUPER\_TOASTER”.

# Exercise 2: Creating a Basic Driver Package

Now that we have a device to install our new universal driver onto, let us get started building a driver package for it!

## Task 1: Open and build the supplied sample solution

* Double click on toaster.sln found in “C:\WinHEC 2017 Lab\Toaster Driver”
* Set the architecture to x64
* Click on the “Build” File menu and select “Rebuild Solution”

At this point the solution should have completed building successfully.

## Task 2: Add the driver so it is installed by the INF

It is great that we have our simple WDF toaster driver building, but it needs to be able to be installed as well. To do that, we need to add the appropriate information to the INF. INX files are used to generate INF files using stampinf. Stampinf is run as part of the build process. It updates the version and can substitute in some variables such as architecture.

* Open “Driver Files\toaster.inx”
* Add the [DestinationDirs] section. We will use 13 as our default destination. This refers to the driver store. Installing files to the driver store allows for more robust driver packages as they are isolated from other driver installations.

[DestinationDirs]

DefaultDestDir = 13

* Locate the “[Toaster\_Device.NT.Copy]” section and add “toaster.sys” to the list of files to be copied to the default destination.

[Toaster\_Device.NT.Copy]

toaster.sys

* Locate the “[SourceDisksFiles]” section and add “toaster.sys = 1” to the list of source files from which the file will be copied.

[SourceDisksFiles]

toaster.sys = 1

* Now that we have indicated where to get the file and where to put the file, now we need to add the declaration to use the file. Add the following service installation sections. Notice again the %13% directory as we are copying the file into the driver store rather than System32.

[Toaster\_Device.NT.Services]

AddService = toaster, 0x00000002, Toaster\_Service\_Inst

[Toaster\_Service\_Inst]

DisplayName = "Basic Toaster Device Driver"

ServiceType = 1 ; SERVICE\_KERNEL\_DRIVER

StartType = 3 ; SERVICE\_DEMAND\_START

ErrorControl = 1 ; SERVICE\_ERROR\_NORMAL

ServiceBinary = %13%\toaster.sys

## Task 3: Add some registry values

Most drivers will need some state stored in the registry. Our toaster driver is no different. It will contain a few keys indicating the minimum and maximum heat settings. We will store these relative to the device’s hardware key using HKR. This isolates the registry in the same way that using the driver store isolates the driver’s files. Not only does this isolation protect against global overwrites, it also will ensure multiple instances of the same device can maintain their own independent settings.

* Add the following sections and keys to your INF

[Toaster\_Device.NT.HW]

AddReg = Toaster\_AddReg

[Toaster\_AddReg]

HKR, Heat, Minimum, 0x10001, 0 ; FLG\_ADDREG\_TYPE\_DWORD

HKR, Heat, Maximum, 0x10001, 10 ; FLG\_ADDREG\_TYPE\_DWORD

## Task 4: Install your new driver

* Rebuild your solution with all the new changes
* Open an administrator command prompt (if you closed the one from earlier)
* Navigate to the output folder for your solution “C:\WinHEC 2017 Lab\Toaster Driver\x64\Debug\toaster”
* Run the command “pnputil /add-driver toaster.inf /install”

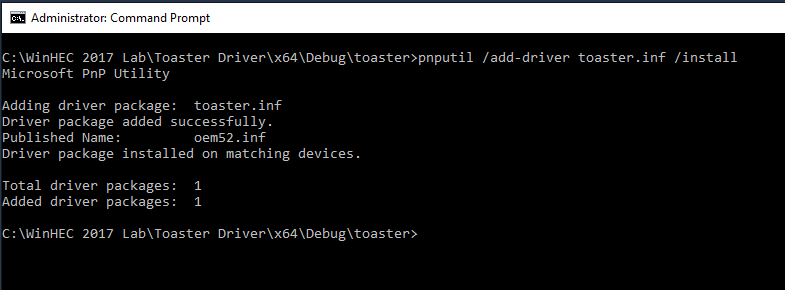


Figure 5: Successful Driver Addition and Installation

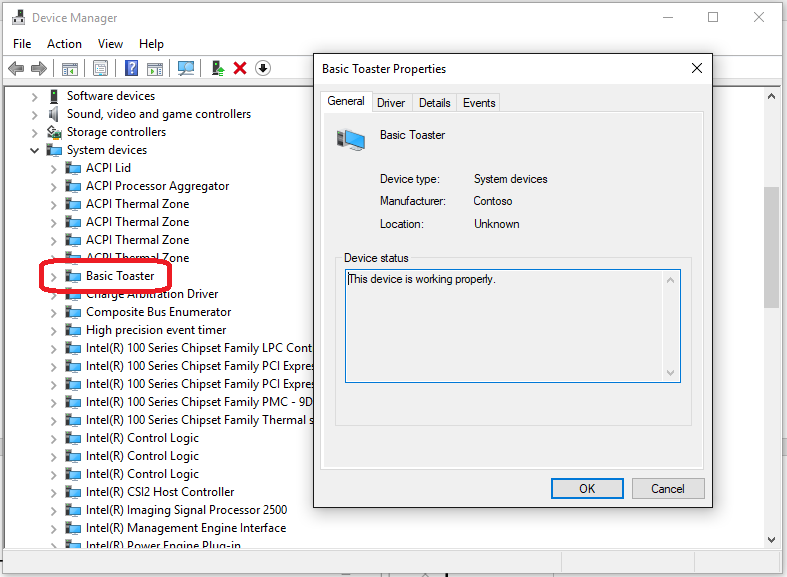


Figure 6: Our new toaster in Device Manager

# Exercise 3: Add a service

Sometimes, a driver will need to use a user mode NT service for additional functionality. Our demo driver is one of those drivers. Instead of using a co-installer to install this service, we will keep our driver package declarative and universal by installing the NT service directly from the INF.

## Task 1: Create a Service

Conveniently, we already have a service ready to go to add to our project. It is a simple RPC server.

* Right-click on the solution and select “Add >”
* Select “Existing Project…”

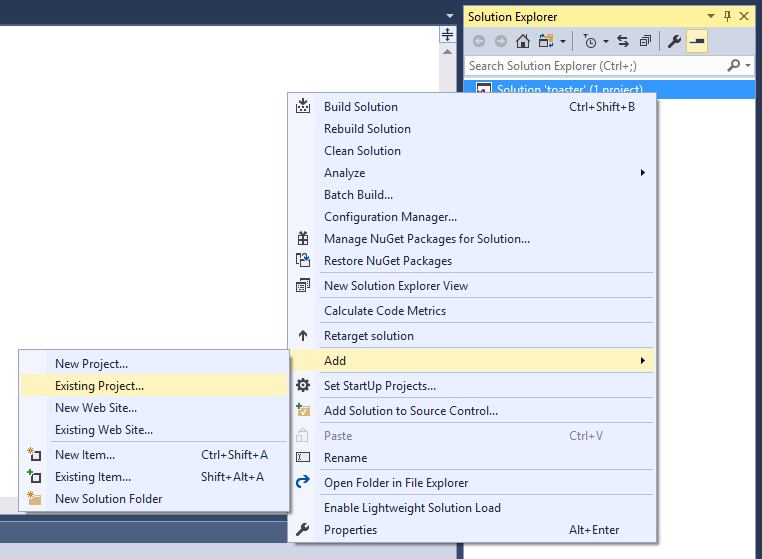


Figure 7: Add an existing project

* Navigate to “C:\WinHEC 2017 Lab\Toaster Driver\Service\”
* Open “RpcServer.vcxproj”

## Task 2: Add service installation logic to the INF

* Open “toaster.inx”
* Locate the “[Toaster\_Device.NT.Services]” section from Exercise 2, Task2
* Add the NT service using the AddService directive (note that 0x800 starts the service after install)

AddService = hsa\_usersrv, 0x00000800, Hsa\_Usersrv\_ServiceInstall

* Add the service install section. Note that we are putting this in the driver store as well.

[Hsa\_Usersrv\_ServiceInstall]

DisplayName = "HSA Toaster User Service"

ServiceType = 0x00000010 ; SERVICE\_WIN32\_SHARE\_PROCESS

StartType = 3 ; SERVICE\_DEMAND\_START

ErrorControl = 1 ; SERVICE\_ERROR\_NORMAL

ServiceBinary = %13%\RpcServer.exe

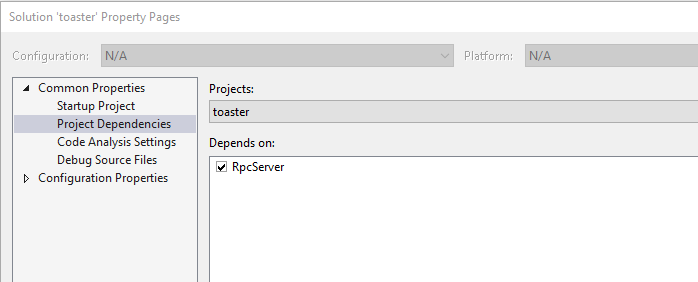
* Add the file to be copied, both destination and source
  + Add this to the [SourceDisksFiles] Section

RpcServer.exe = 1

* + Add this to the [Toaster\_Device.NT.Copy] Section

RpcServer.exe

* Add a dependency between the projects so that the RPCServer is built before the Driver and packaged correctly
  + Right-Click on the solution and select “Properties”
  + Select “Project Dependencies”
  + Under “Projects:” select “toaster”
  + Under “Depends on:” check the box for “RpcServer”
  + Click OK

Figure 8: Adding a project dependency

* Add the RpcServer project output to be included in the toaster driver package
  + Right-Click on the “toaster” project
  + Select “Unload Project”
  + Right-Click on the “toaster” project
  + Select “Edit toaster.vcxproj”
  + Locate the line “<FilesToPackage Include=”$(TargetPath)” />”
  + Add this below:

<FilesToPackage Include="$(SolutionDir)$(Platform)\$(Configuration)\rpcserver.exe" />

* + Right-Click on the “toaster” project
  + Select “Reload Project”

## Task 3: Build and Install

Now that we have added the NT service to our project and our INF, we should be able to build and install it!

* Rebuild your solution with all the new changes
* Open an administrator command prompt (if you closed the one from earlier)
* Navigate to the output folder for your solution “C:\WinHEC 2017 Lab\Toaster Driver\x64\Debug\toaster”
* Run the command “pnputil /add-driver toaster.inf /install”

If the driver installed successfully, you should now be able to find your service running in Task Manager!

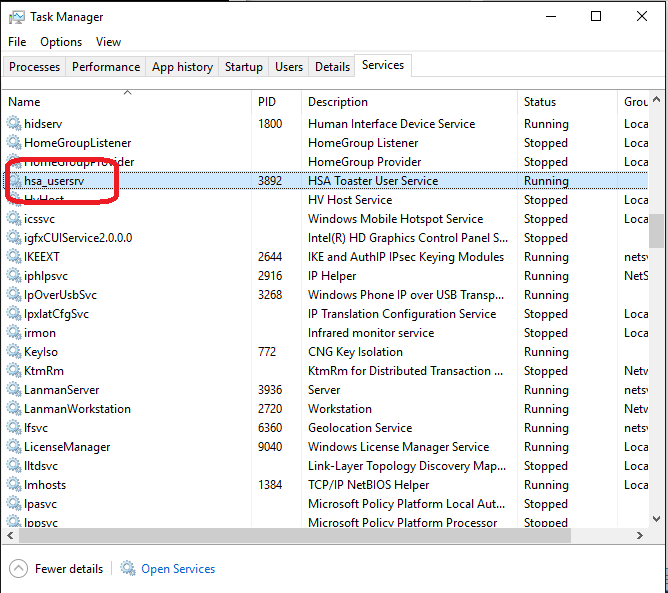


Figure 9: Our NT Service running!

# Exercise 4: Add a user interface with a Hardware Support App (HSA)

For a driver to be universal, both the driver and any applications it works with must work on all flavors of Windows 10. To enable this, a Universal Windows Platform (UWP) Hardware Support App (HSA) can be written. An HSA can communicate with either an NT Service or directly to a driver. In this exercise, we will link our new HSA with the service created in Exercise 3.

## Task 1: Build an App

This time we will start with a different solution rather than adding the app to our existing driver solution.

* Double click on app.sln found in “C:\WinHEC 2017 Lab\Toaster Support App”
* Set the architecture to x64
* Click on the “Build” File menu and select “Rebuild Solution”
* **NOTE**: “Developer Mode” is mandatory to run UWP apps from visual studio. Enable it if prompted.

Our app has all the functionality to talk to an RPC server, but it doesn’t have the permission to do so. You can try this for yourself by deploying the app as is (click the run on “Local Machine” button)

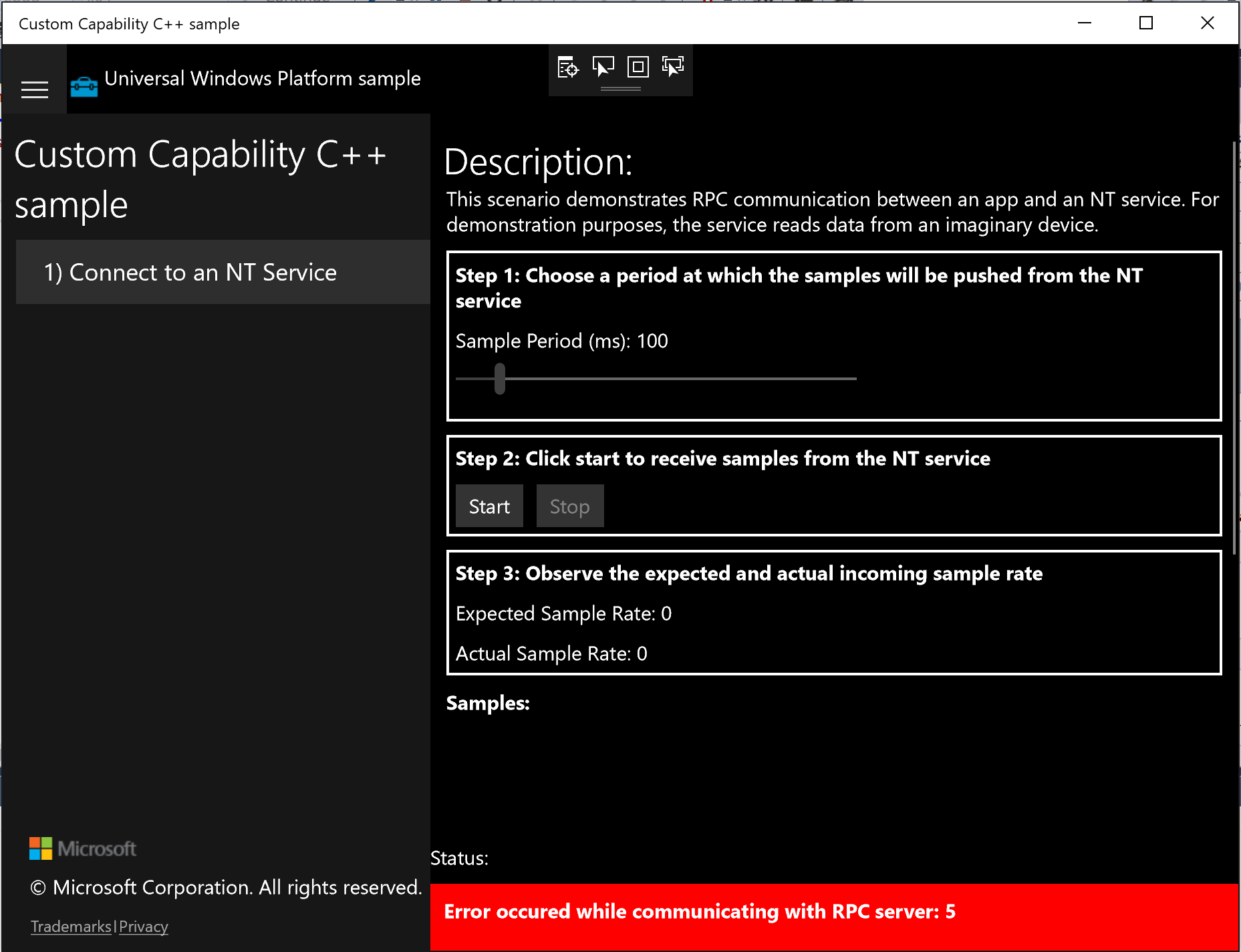


Figure 10: Failed RPC connection ☹

## Task 2: Add a custom capability to the App

A Signed Custom Capability Descriptor (SCCD) is used to describe a custom capability. These are generated by Microsoft and provided to partners. We have an SCCD ready for use in this lab.

* Add the SCCD to the project
  + Right-click on the project “CustomCapability”
  + Add -> Existing Item…
  + Choose “CustomCapability.SCCD”
* Add the SCCD to the deployed content
  + Select the file (CustomCapability.SCCD) in the Project
  + Set the “Content” field to “True”

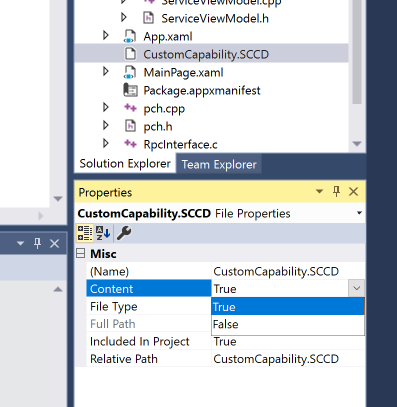


Figure 11: Including a file for deployment

* Add the custom capability to the app manifest
  + Right-click on “Package.appxmanifest”
  + Select “Open With…”
  + Choose “XML (Text) Editor” and click OK
  + Just before “</Package>” add the following (this capability name is from the SCCD)

<Capabilities>

<uap4:CustomCapability Name="microsoft.hsaTestCustomCapability\_q536wpkpf5cy2"/>

</Capabilities>

## Task 3: Grant access to the driver’s RPC server with the capability

Having a capability described in the app is only half of the story. To compete the connection, the RPC server needs to allow for it to be used.

* Return to the driver solution from Exercise 4 (“c:\WinHec 2017 Lab\Toaster Driver\toaster.sln”)
* Open “Source Files\RpcServer.cpp”
* Add the variable to define the custom capability name just before the line “bool ShutdownRequested”. Notice that this is the same name as above.

const WCHAR\* CustomCapabilityName = L"microsoft.hsaTestCustomCapability\_q536wpkpf5cy2";

bool ShutdownRequested;

* Next, we need to get the Custom Capability’s Security ID. We’ll insert this code in the RpcServerStart() function just after the variable defintions. (under comment //INSERT DERIVE CAPABILTY SIDS FROM NAME HERE)

if (!DeriveCapabilitySidsFromName(

CustomCapabilityName,

&capabilityGroupSids,

&capabilityGroupSidCount,

&capabilitySids,

&capabilitySidCount))

{

hResult = GetLastError();

goto end;

}

if (capabilitySidCount != 1)

{

// Unexpected sid count

hResult = ERROR\_INVALID\_PARAMETER;

goto end;

}

* And finally, add the Custom Capability to the Access Control List (ACL)
  + Replace

ea[1].Trustee.ptstrName = static\_cast<LPWSTR>(everyoneSid);

* + With

ea[1].Trustee.ptstrName = static\_cast<LPWSTR>(capabilitySids[0]);

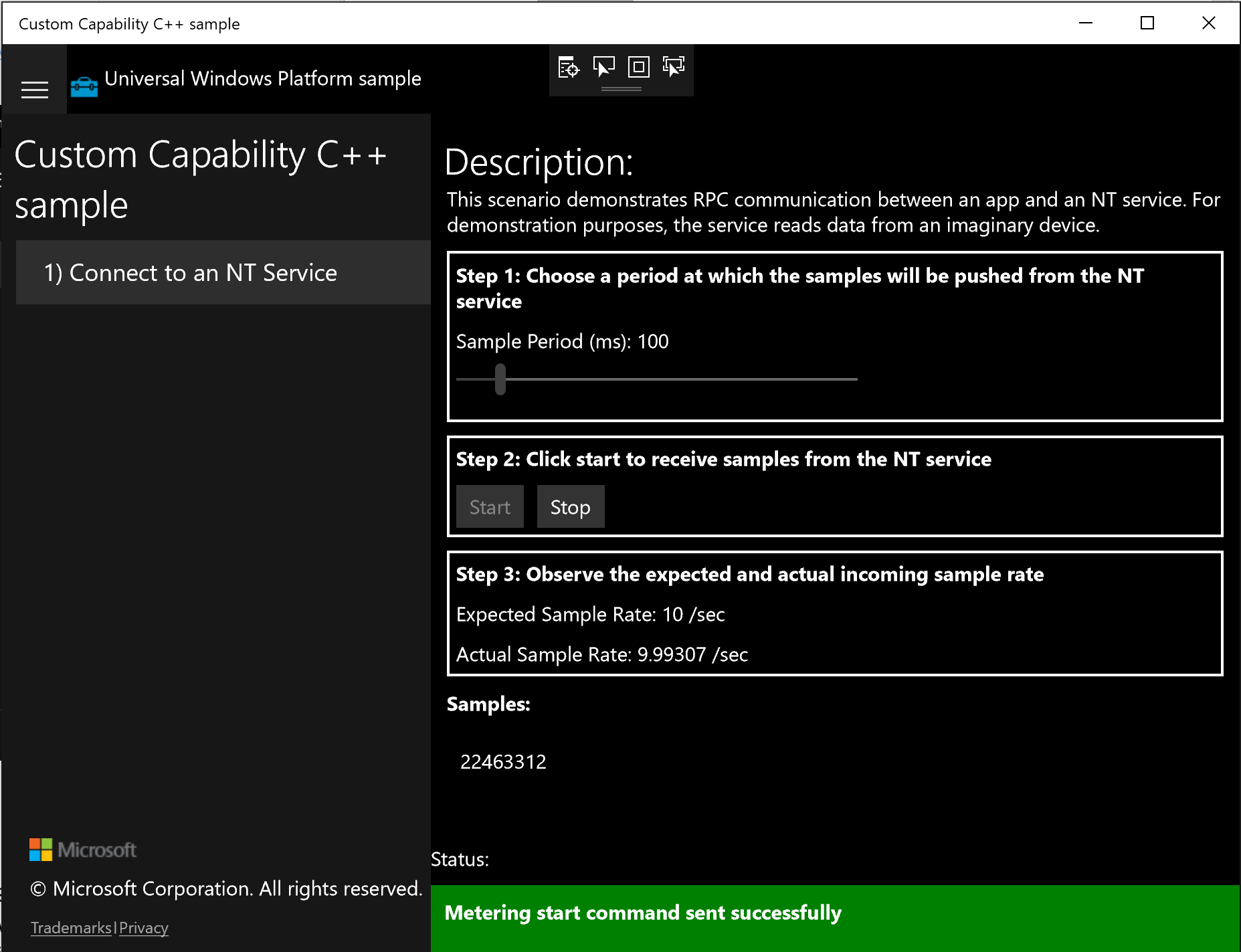
## Task 4: Build and Install the new Toaster Driver

Now we have all the code to enable our Toaster Support App to talk with the RPC server delivered by the Toaster driver.

* Rebuild the driver solution with all the new changes
* Open an administrator command prompt (if you closed the one from earlier)
* Navigate to the new output folder for your solution “C:\WinHEC 2017 Lab\Toaster Driver\x64\Debug\Toaster”
* Run the command “pnputil /add-driver toaster.inf /install”

## Task 5: Build and Deploy the Toaster Support App

* Rebuild the app solution with the new changes
* Click the run on “Local Machine” button
* Click the start button in the app

  
Figure 12: Successful RPC connection!

# Exercise 5: Add Componentized Customizations including HSA reference

A basic driver is great, but there are many situations when a driver needs to be customized for a use case or partner. To enable this scenario, Windows 10 supports Extension INFs. These are light weight INFs that can add these customizations.

## Task 1: Create Extension INF Project

* Return to your driver project
* Right click on the Solution ‘toaster’ and add a new project
  + Add -> New Project…
* Select “Driver Install Package”
  + Installed -> Visual C++ -> Windows Drivers -> Package
* Name the new project “Super Toaster”
* Click “OK”

## Task 2: Add a new INX file to the project

* Right-Click on “Driver Files” within your new project
* Add -> New Item…
* Select Text File (.txt)
  + Installed -> Visual C++ -> Utility -> Text File (.txt)
* Name to “super\_toaster\_extension.inx”

Now we have a blank file that we will mold into our Extension INF.

## Task 3: Add Version Information

* The version information of an INF indicates the important identity information about the driver.
* Copy this section into your new INX

[Version]

Signature = "$WINDOWS NT$"

Class = Extension

ClassGuid = {e2f84ce7-8efa-411c-aa69-97454ca4cb57}

Provider = Contoso

ExtensionId =

CatalogFile = super\_toaster\_extension.cat

DriverVer = 05/16/2017, 15.14.36.721

All extension INFs are of class “Extension” with the ClassGuid “{e2f84ce7-8efa-411c-aa69-97454ca4cb57}”. That allows the OS to know to treat this as an additional driver for a device, rather than the primary base driver.

## Task 4: Add a unique ExtensionId

The ExtensionId is used to identify an Extension driver. This allows versioning of extensions and replacements over time.

* Select “Tools” from the file menu
* Select “Create GUID”
* Select #4 Registry Format
* Click Copy
* Select Exit
* Paste your new GUID after “ExtensionId =”

## Task 5: Add some installation information

Add the following to your Extension INF to give it the essentials to install. Note that we are matching the extension INF on the “TOASTER\SUPER\_TOASTER” Hardware ID (HWID). Only the devices that express the more specific toaster design will be able to leverage the super toaster extension. The rest of the basic toasters will only get the base driver.

[Manufacturer]

Contoso = Contoso, NT$ARCH$

[Contoso.NT$ARCH$]

"Super Toaster" = SuperToaster\_Install, TOASTER\SUPER\_TOASTER

[SuperToaster\_Install.NT]

; empty

## Task 6: Add some customized registry keys

We added two registry keys with the basic toaster driver. The super toasters are capable of a higher heat setting so we will overwrite that key. We’ll also add an additional key to change the friendly name of the device. Extension INFs are always installed after the Base INF. That means that these configurations will always overwrite or add to the configuration added by the Base INF.

[SuperToaster\_Install.NT.HW]

AddReg = SuperToaster\_AddReg

[SuperToaster\_AddReg]

HKR, Heat, Maximum, 0x10001, 11 ; FLG\_ADDREG\_TYPE\_DWORD

HKR,,FriendlyName,,"Super Toaster"

## Task 7: Add a SoftwareComponent identity for our HSA

AddSoftware is the directive that is used to trigger the acquisition of a store app. AddSoftware can only be called from an INF of class “SoftwareComponent”. Since our base driver is of class “System” and our extension is of class “Extension” it is necessary to add another INF to the mix. The identity for the new INF is created using the AddComponent directive.

[SuperToaster\_Install.NT.Components]

AddComponent = SuperToasterHSA,,SuperToaster\_Component\_Install

[SuperToaster\_Component\_Install]

ComponentIDs = PROVIDER\_Contoso&&COMPONENT\_ToasterHSA

DisplayName = "Super Toaster Support Application"

In Exercise 6 we’ll complete the AddSoftware delivery of the HSA.

## Task 8: Build and Install

Now we have a completed extension INF! Let us see it in action.

* Rebuild your solution with all the new changes
* Open an administrator command prompt (if you closed the one from earlier)
* Navigate to the new output folder for your solution “C:\WinHEC 2017 Lab\Toaster Driver\x64\Debug\Super Toaster”
* Run the command “pnputil /add-driver super\_toaster\_extension.inf /install”

If we look in device manager now, you will see that there are new device properties for extension INFs.

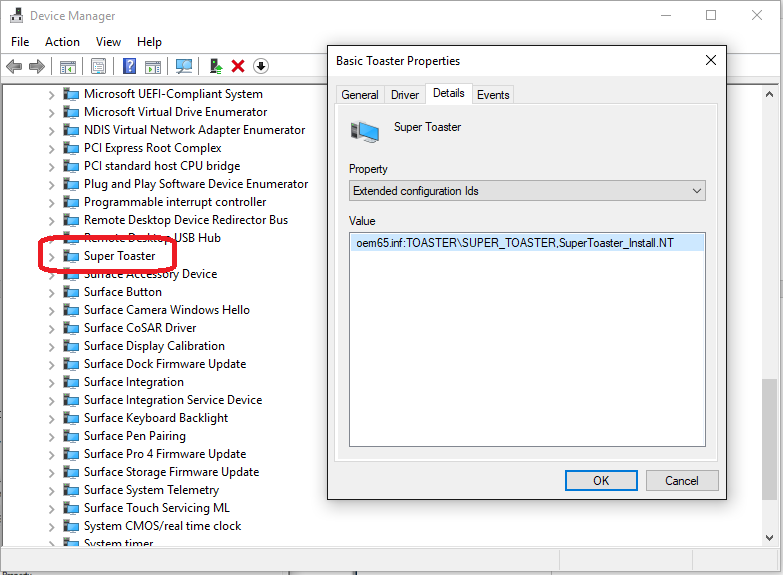


Figure 13: Extended configuration Ids showing extension INF installed on the Super\_Toaster Hardware ID

# Exercise 6: Create a Component INF to deliver the HSA

In Exercise 6, we had our extension INF add a software device node in order to deliver our HSA. Did you notice it under “Software Components” in device manager? Now we need a Component INF to match on that identity and actually deliver our application.

## Task 1: View your new component device

* Find and select the “Super Toaster” device under “System devices”
* With “Super Toaster” selected, view by connection
  + View -> Devices by connection
* Expand the children of “Super Toaster” by clicking on the “>”
* Right-click on the “Generic software component” and select “properties”
* View the “Hardware IDs” under details
* Notice that the HWID matches what we defined in the extension inf with an SWC\ prefix

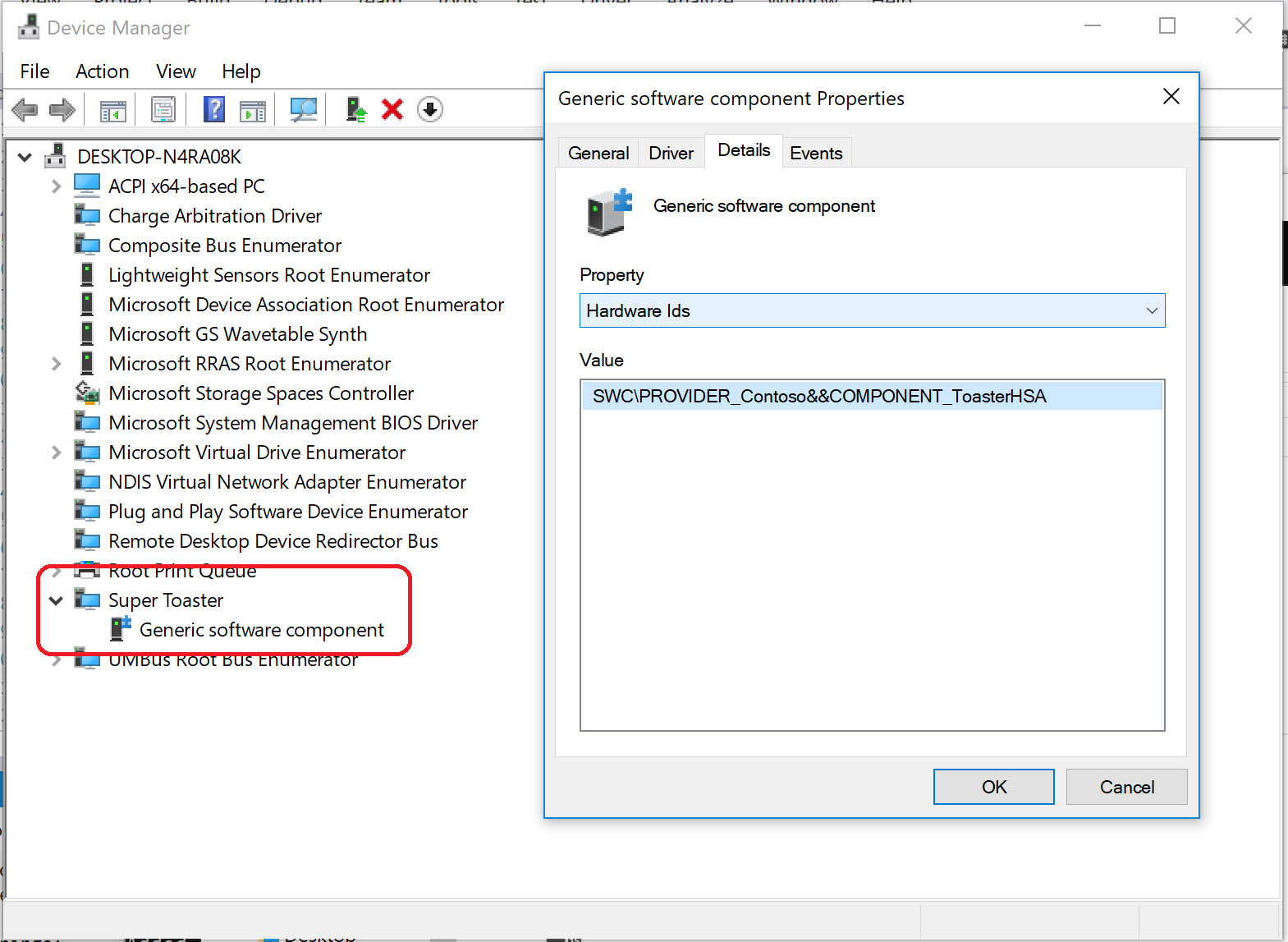


Figure 14: The software component and it’s HWID

## Task 2: Create Component INF Project and add a new INX file to the project

* Right click on the solution and add a new project
  + Add -> New Project…
* Select “Driver Install Package”
  + Installed -> Visual C++ -> Windows Drivers -> Package
* Name the new project “Toaster Component”
* Click “OK”
* Right-Click on “Driver Files” in the new project
* Add -> New Item…
* Select Text File (.txt)
  + Installed -> Visual C++ -> Utility -> Text File (.txt)
* Name to “toaster\_hsa\_component.inx”

Now we have a blank file that we will mold into our Software Component INF.

## Task 3: Add Version Information and installation information

* The version information of an INF indicates the important identity information about the driver.
* Copy this section into your new INX

[Version]

Signature = "$WINDOWS NT$"

Class = SoftwareComponent

ClassGuid = {5c4c3332-344d-483c-8739-259e934c9cc8}

Provider = Contoso

CatalogFile = toaster\_hsa\_component.cat

DriverVer = 11/09/2017, 1.0.01.03

[Manufacturer]

Contoso = Contoso, NT$ARCH$

[Contoso.NT$ARCH$]

"Toaster HSA Component" = Toaster\_HSA\_Install, SWC\PROVIDER\_Contoso&&COMPONENT\_ToasterHSA

[Toaster\_HSA\_Install.NT]

; empty

In order to use the AddSoftware directive, the inf must be of class “SoftwareComponent” with the ClassGuid “{5c4c3332-344d-483c-8739-259e934c9cc8}”.

Additionally, it must match on the HWID that was added by the extension inf. In this case, the HWID is “SWC\PROVIDER\_Contoso&&COMPONENT\_ToasterHSA”. You’ll notice that SWC\ has been prepended. This is done automatically when the AddComponent directive in the Extension INF is processed.

## Task 4: Use the AddSoftware directive

Now for the real action! We’ll use AddSoftware to trigger the store to install our HSA.

[Toaster\_HSA\_Install.NT.Software]

AddSoftware = ToasterHSA,,Toaster\_HSA\_SoftwareInstall

[Toaster\_HSA\_SoftwareInstall]

SoftwareType = 2

SoftwareID =

## Task 5: Add the App PFN

The software install section specified in Task 4 needs a value for the SoftwareID. This comes from your HSA.

* Return to your App Solution in Visual Studio
* Open the Package.appxmanifest file with the manifest viewer (double click on the file)
* Select the “Packaging” tab
* At the bottom, you’ll see the App’s PFN, in our case it’s:
  + Microsoft.SDKSamples.CustomCapability.CPP\_8wekyb3d8bbwe
* Add the pfn:// prefix to this and add it to the component INF
  + Replace “Software ID =” with this

SoftwareID = pfn://Microsoft.SDKSamples.CustomCapability.CPP\_8wekyb3d8bbwe

## Task 6: Build and Install

Now we have a completed component INF! Unfortunately, it would take too long to publish the application to the store and see it automatically install, however we can see the component INF install.

* Rebuild your solution with all the new changes
* Open an administrator command prompt (if you closed the one from earlier)
* Navigate to the new output folder for your solution “C:\WinHEC 2017 Lab\Toaster Driver\x64\Debug\Toaster Component”
* Run the command “pnputil /add-driver toaster\_hsa\_component.inf /install”

You’ll notice that PnPutil reports that the “Driver package matches devices that already have their drivers up to date” and if you go to device manager, you’ll see that the component device still reports “Generic software component”. The reason for this is because a generic inbox driver matches on the device and signed inbox drivers will be preferred over unsigned drivers (as is the case with our new component inf)

To fix this:

* Go to device manager
* Right click on the “Generic software component”
* Select “Properties”
* Select “Update Driver”
* Select “Browse my computer for driver software”
* Select “Let me pick from a list of available drivers on my computer”
* Choose “Toaster HAS Component”
* Click “Next” and then “Close”

And you’re done! In the production scenario, the component inf would be properly signed and therefore would out rank the generic driver and automatically be installed.

If our application had been published to the store, it would also be automatically installed at this time.

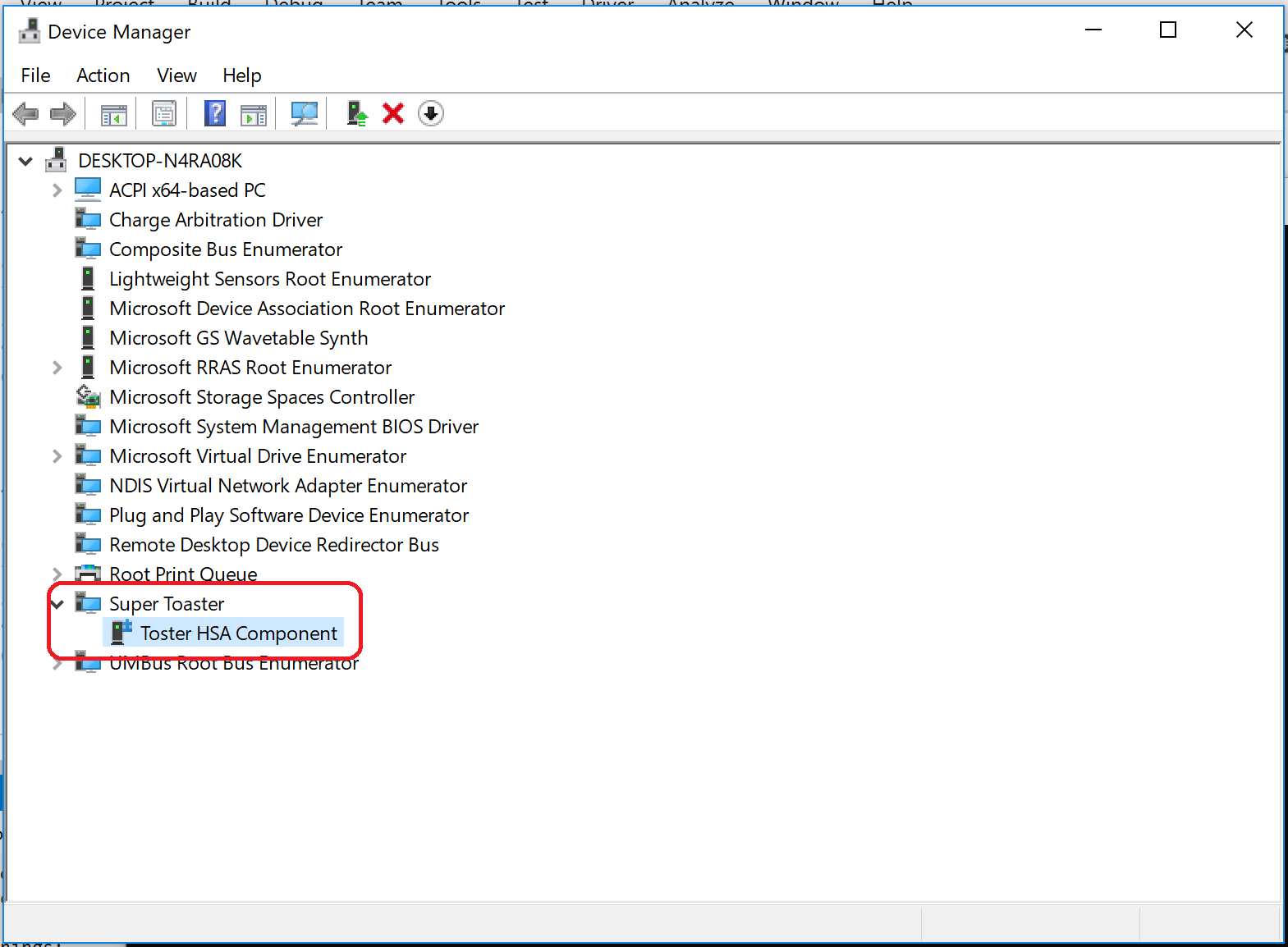


Figure 15: The Component INF installed

# Exercise 7: Validate it is Universal

Now we have a Universal driver! Or do we? There are some key tools that can be used to ensure that our driver is indeed ready for all flavors of Windows 10

If you look close at the output console, you might have noticed that it shows “Driver is a Universal Driver”.

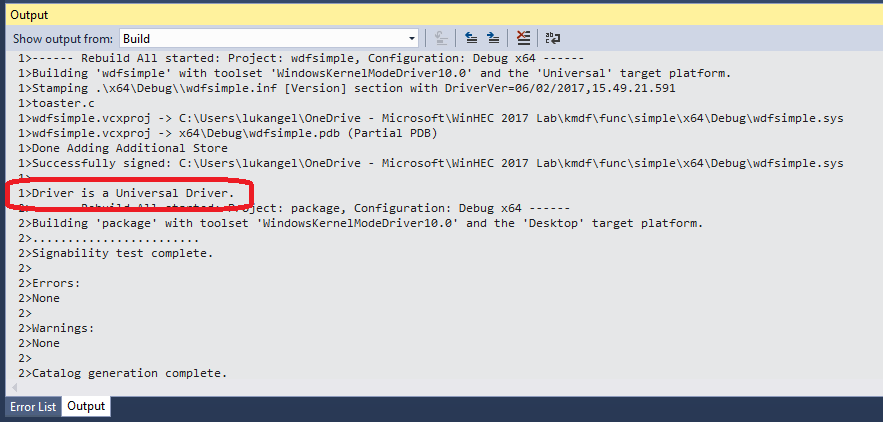


Figure 16: Universal Driver Testing During Build

When Visual Studio is configured properly to build universal drivers, it will run both “INFVerif.exe /u” to ensure that the INF meets the universal driver requirements as well as executes “APIValidator.exe” to ensure only Universal APIs are used in the building of the driver. You may, however, want to run these outside of visual studio. Here’s how.

## Task 1: Run INFVerif /u

* Open an administrator command prompt (if you closed the one from earlier)
* Navigate to “C:\Program Files (x86)\Windows Kits\10\Tools\x64\”
* Run:  
  infverif.exe “C:\WinHEC 2017 Lab\Toaster Driver\x64\Debug\toaster\toaster.inf” /u /v

## Task 2: Run API Validator

API validator runs against complete directories and you must include the whitelist of universal drivers when you run it.

* From your command prompt, navigate to: “"C:\Program Files (x86)\Windows Kits\10\bin\x64\”
* Run:

apivalidator.exe -DriverPackagePath:"c:\WinHEC 2017 Lab\Toaster Driver\x64\Debug\toaster" -SupportedApiXmlFiles:"C:\Program Files (x86)\Windows Kits\10\build\universalDDIs\x64\UniversalDDIs.xml" -ModuleWhiteListXmlFiles:"C:\Program Files (x86)\Windows Kits\10\build\universalDDIs\x64\ModuleWhiteList.xml"

# Conclusion

In this lab, we have been able to scratch the surface of Universal drivers. Drivers are certainly a very complex space but hopefully with the knowledge learned hear you will be able to start your journey building your own Universal driver that will work on all Windows 10 platforms. There is certainly so much more out there to learn. Keep an eye on MSDN for more information. In particularly these pages:

* [Getting Started with Universal Windows drivers](https://docs.microsoft.com/en-us/windows-hardware/drivers/develop/getting-started-with-universal-drivers)
* [Using a Universal INF File](https://docs.microsoft.com/en-us/windows-hardware/drivers/install/using-a-configurable-inf-file)
* [Hardware access for Universal Windows Platform apps](https://docs.microsoft.com/en-us/windows-hardware/drivers/devapps/hardware-access-for-universal-windows-platform-apps)
* [Creating a custom capability to pair a driver with a Hardware Support App (HSA)](https://docs.microsoft.com/en-us/windows-hardware/drivers/devapps/custom-capabilities-for-universal-windows-platform-apps)
* [Universal Windows Drivers for Audio](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-universal-drivers)
* [Device Identification Strings](https://docs.microsoft.com/en-us/windows-hardware/drivers/install/device-identification-strings)

Additional topics added in version 2:

* [Using an Extension INF File](https://docs.microsoft.com/en-us/windows-hardware/drivers/install/using-an-extension-inf-file)
* [Using a Component INF File](https://docs.microsoft.com/en-us/windows-hardware/drivers/install/using-a-component-inf-file)
* [Using a custom capability to pair a Hardware Support App (HSA) with a driver](https://docs.microsoft.com/en-us/windows-hardware/drivers/devapps/using-a-custom-capability-to-pair-hsa-with-driver)