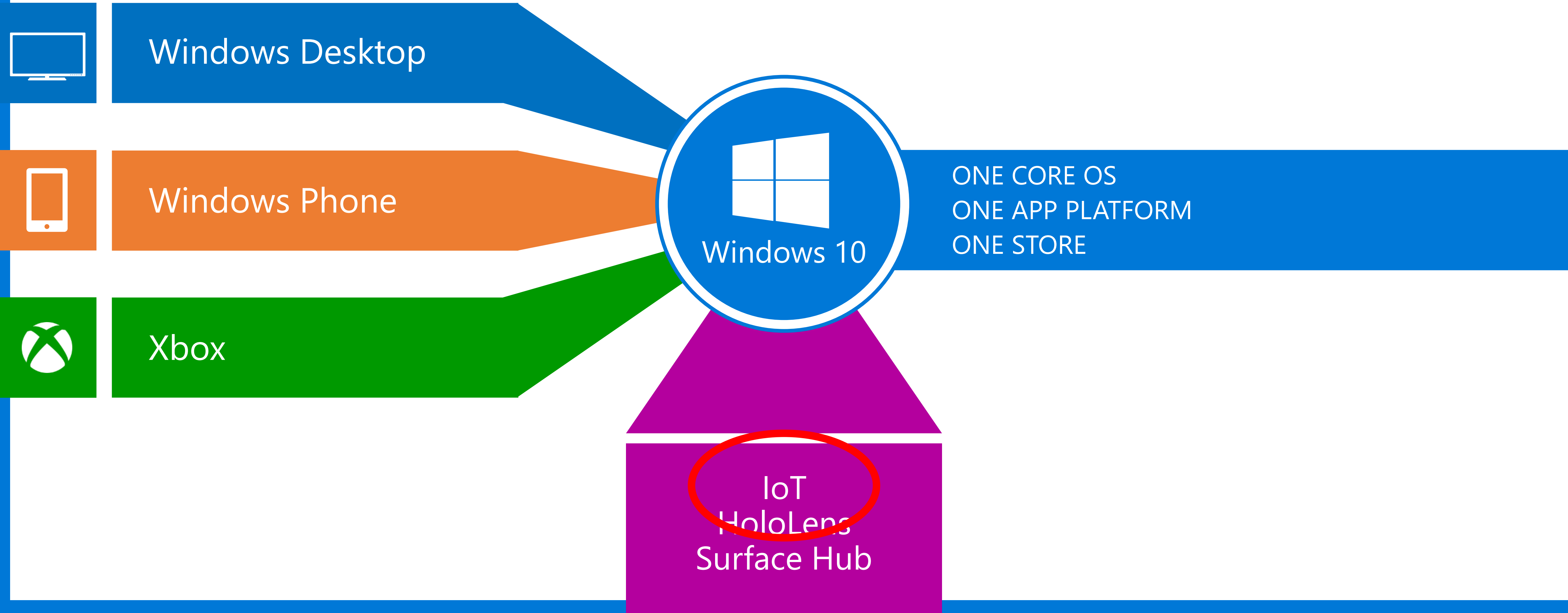
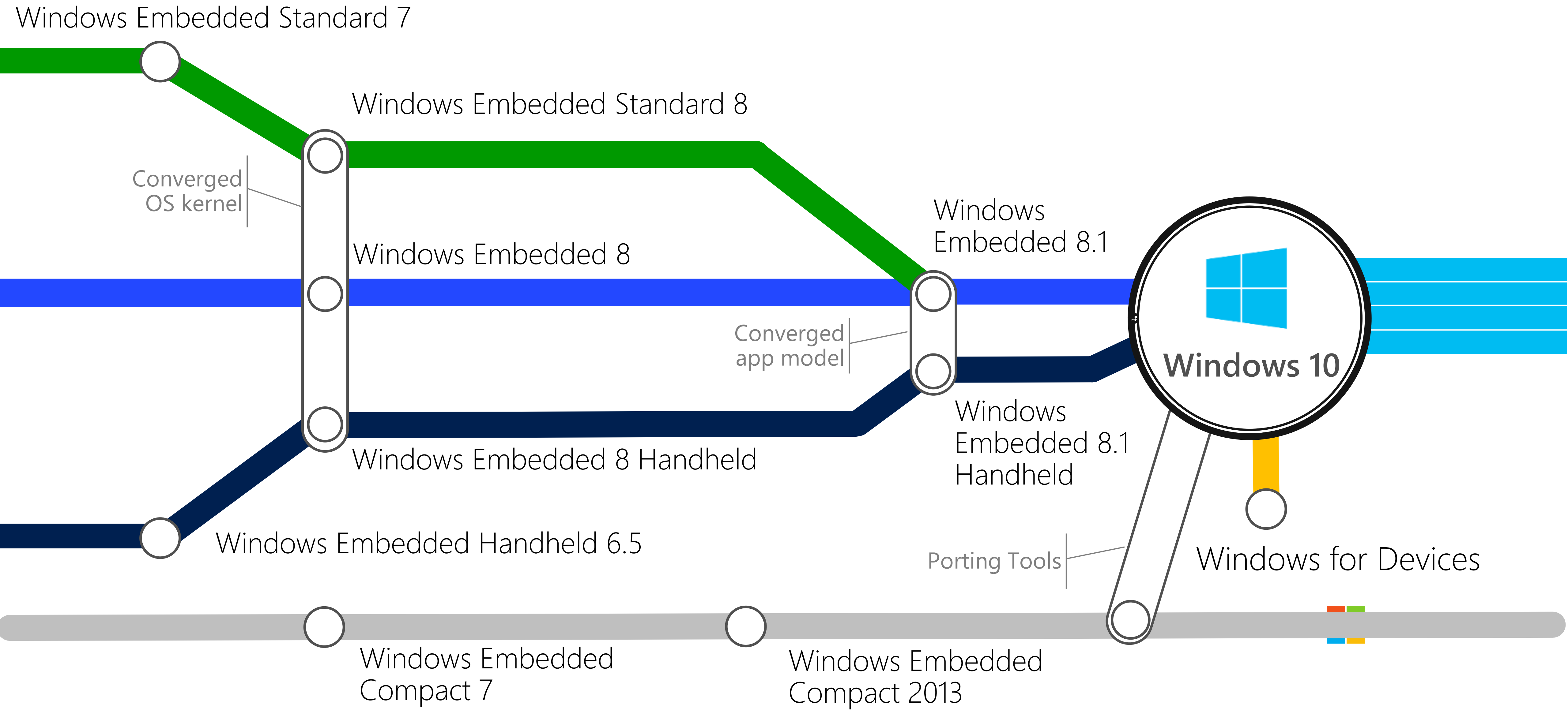
# Commercial Windows IOT

Gunter Logemann
Sr. Consultant

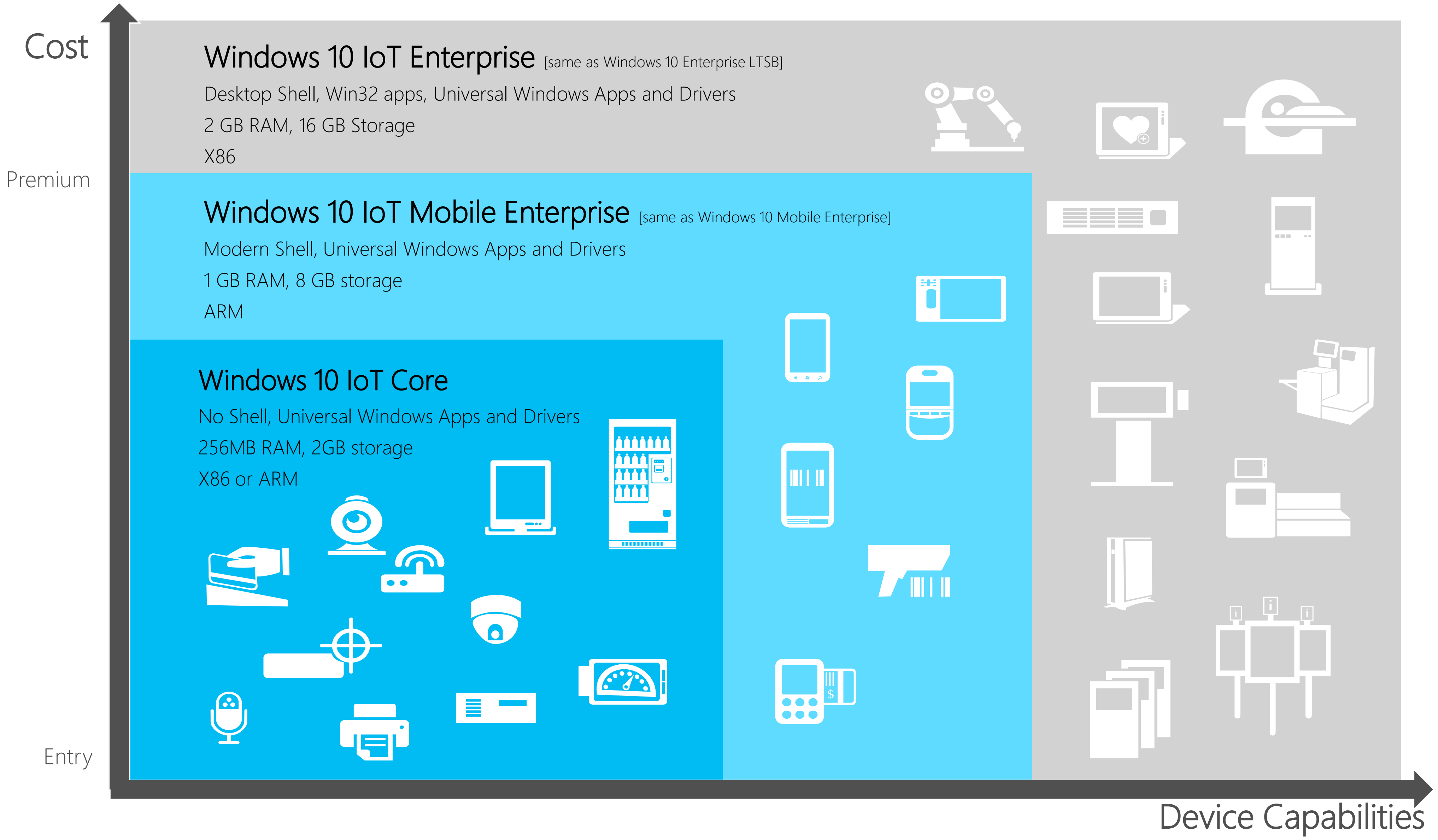
# The journey to one Windows…



# Embedded Platform Convergence Journey


Windows Embedded Standard 7
Windows Embedded Standard 8
Converged OS kernel
Windows Embedded 8
Windows Embedded 8.1
Converged app model
Windows 10
Windows Embedded 8.1 Handheld
Windows Embedded 8 Handheld
Windows Embedded Handheld 6.5
Porting Tools
Windows for Devices
Windows Embedded Compact 7
Windows Embedded Compact 2013

# Windows 10 IoT



### Windows 10 IoT Enterprise [same as Windows 10 Enterprise LTSB]

Desktop Shell, Win32 apps, Universal Windows Apps and Drivers
2 GB RAM, 16 GB Storage
X86

### Windows 10 IoT Mobile Enterprise [same as Windows 10 Mobile Enterprise]

Modern Shell, Universal Windows Apps and Drivers
1 GB RAM, 8 GB storage
ARM

### Windows 10 IoT Core

No Shell, Universal Windows Apps and Drivers
256MB RAM, 2GB storage
X86 or ARM

Microsoft

# Universal Windows Platform

UWP apps

Natural & rich user experience

One management & servicing approach

# UWP for different devices

- Full support of all App models, using C++, C#, Java Script
- Full support of 3rd party Appmodels that support UWP. E.g. Qt, Xamarin, Unity
- Adaptive Layout and Code allows you to build on App that scales to diffrent plattforms
- App framework for the Wearable availiable
- Samples to demonstrate diffrent IOT specific features (e.g. Process launcher)

https://technet.microsoft.com/en-us/itpro/windows/manage/lockdown-features-windows-10

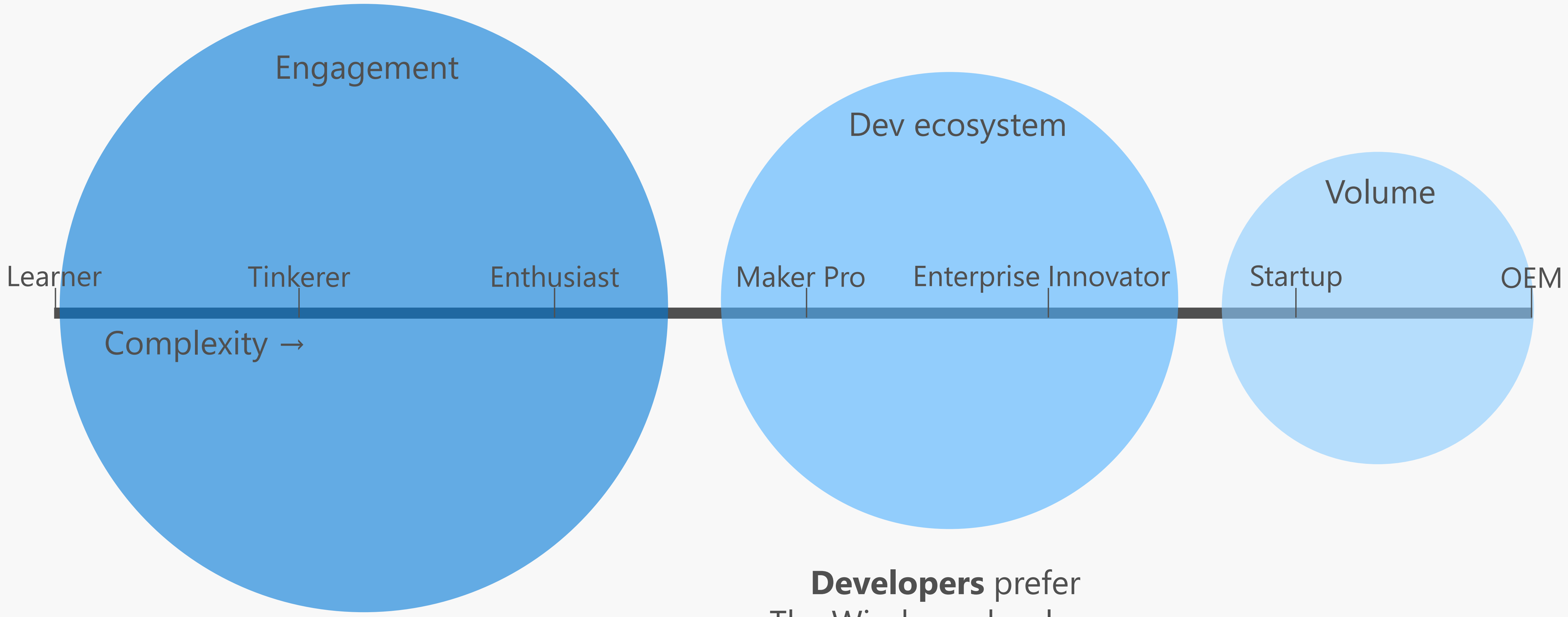
## Microsoft IoT

Comprehensive solutions from device to cloud

| Windows | Azure IoT |
|---|---|
| **IoT Editions Power a Broad Range of Devices** | **Cloud-Based IoT Services & Solutions** |
| 25 years of history in embedded devices | Easy to provision, use and manage |
| One Windows platform for all devices | Pay as you go, scale as you need |
| Enterprise-ready, OEM-ready, Maker-friendly | Global reach, hyper scale |
| Designed for today’s IoT environments | End-to-end security & privacy |
| Scalable solutions from free Windows IoT Core to Windows IoT Enterprise on PC-Like Devices | Windows, Mbed, Linux, iOS, Android, RTOS support |

# Any developer can build an IoT device

## Devices are the new apps!



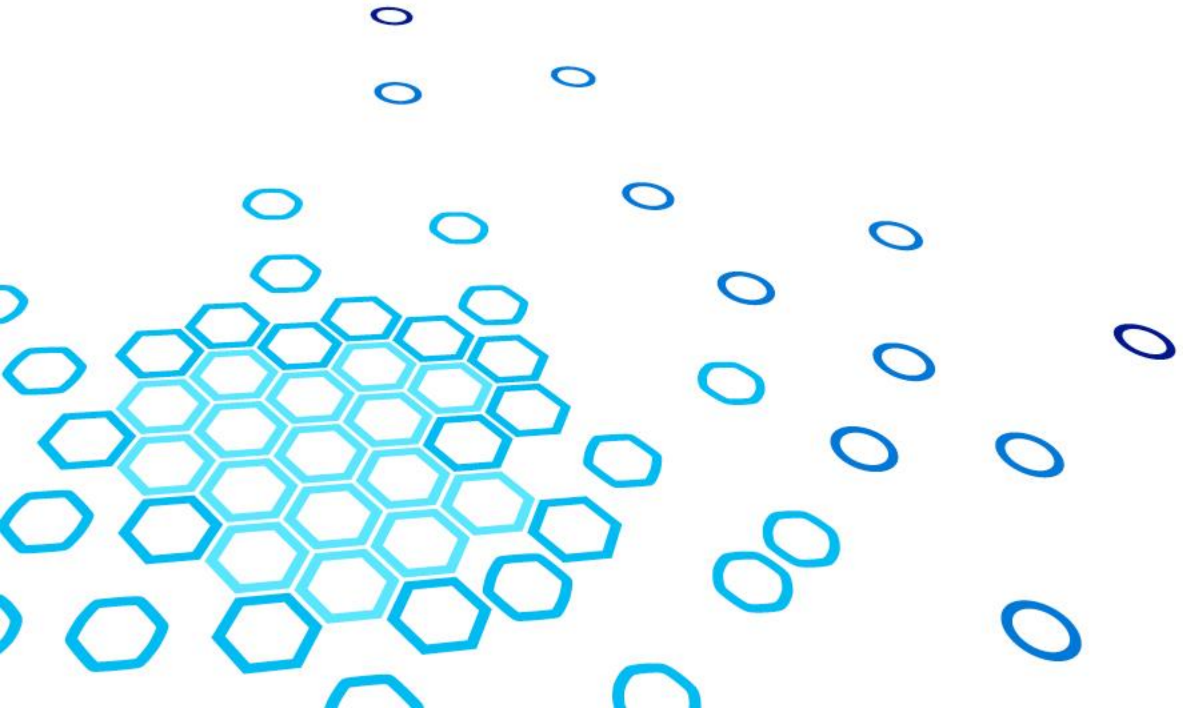
**Makers** find Windows IoT Core
easy for learning devices

**Developers** prefer
The Windows developer
experience
for devices

**Manufacturers** trust
Windows IoT Core to go to market

# What is Windows IOT

Enable Embedded Features



Microsoft

## Embedded features

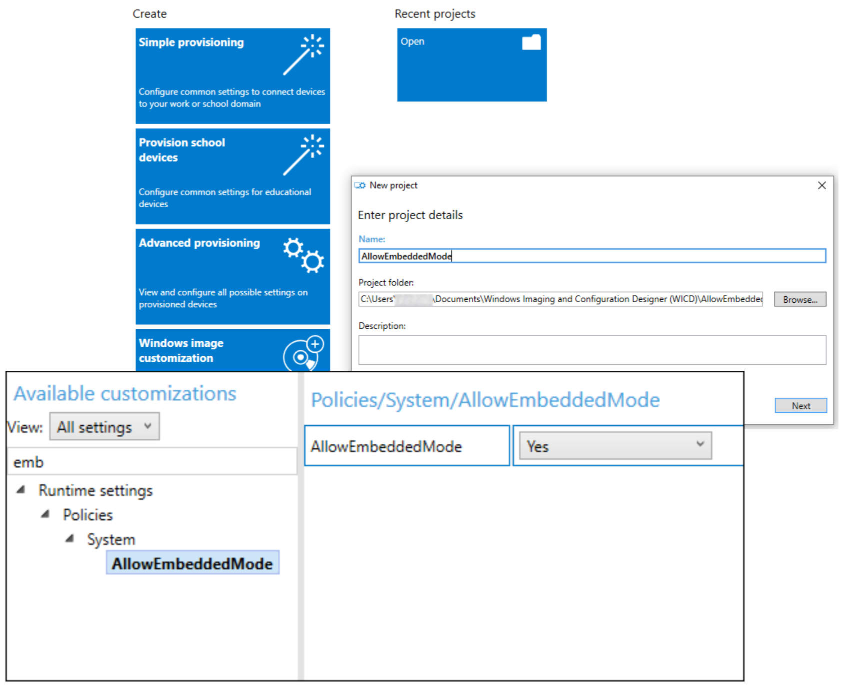
| Background Applications | Use of the lowlevelDevice capability | Use of the systemManagement capability |
|---|---|---|
| Remove limits enforced by the by the resource manager. | low-level hardware interfaces like GPIO, SPI, and I2C. (IOT Core only) | ProcessLauncher<br>TimeZoneSettings<br>ShutdownManager<br>AllJoyn loopback |

Embedded mode is only enabled by default on Window IoT Core and must be enabled on standard Windows and Windows Mobile

Microsoft

# How to enable features

1. Download and install the ADK
2. Install Immaging and Configuration Designer (ICD)
3. Create a provisioning Package that sets AllowEmbeddedMode=Yes
4. Install the package


Create
Simple provisioning
Configure common settings to connect devices to your work or school domain
Provision school devices
Configure common settings for educational devices
Advanced provisioning
View and configure all possible settings on provisioned devices
Windows image customization
Recent projects
Open
New project
Browse...
Next
Available customizations
View:
All settings
Runtime settings
Policies
System

Enter project details

Name: AllowEmbeddedMode

Project folder: C:\Users\[redacted]\Documents\Windows Imaging and Configuration Designer (WICD)\AllowEmbedded

Description:

emb

AllowEmbeddedMode

Policies/System/AllowEmbeddedMode

AllowEmbeddedMode: Yes

Microsoft

## More embedded features…
(also apply to windows 10 in general now)

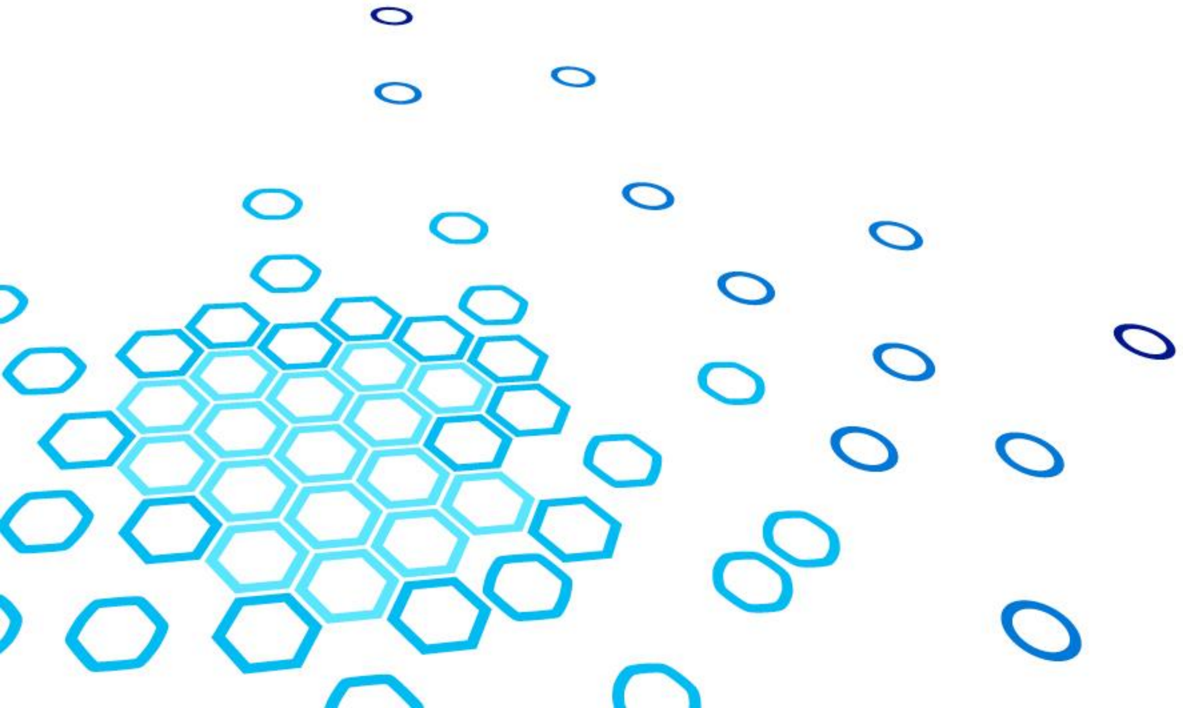
| Write Filters and Overlays | USB Filter | Dialog and Notification Filters | Input Filters | AppLocker and Layout Control | Shell and App Launcher |
|---|---|---|---|---|---|
| Easily create read only devices. Improve system uptime | Only allow approved USB peripherals | Block Pop-up Dialog Boxes and system notifications | Block hotkeys and edge gestures to prevent system access | Control which apps are visible and can run | Enable single Win32 or Modern app experience on device |

Microsoft

https://technet.microsoft.com/en-us/itpro/windows/manage/lockdown-features-windows-10

# IOT Security

Security is not optional for Devices!



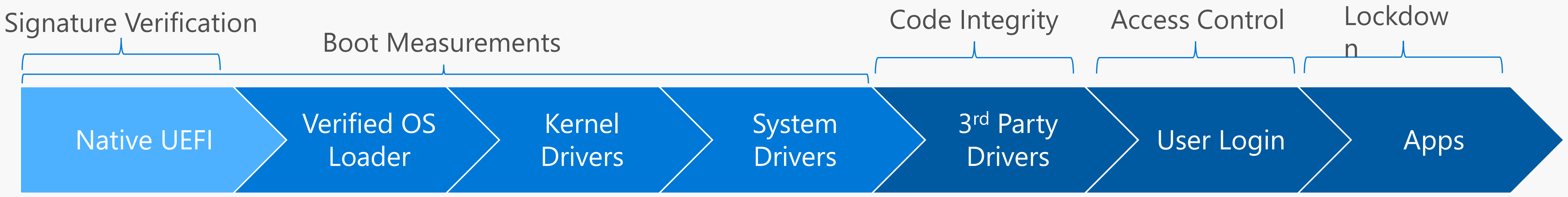
Microsoft

# Threats for IOT devices

| | | |
|---|---|---|
| Default Passwords and backdoors in devices | Devices that are not maintained | Unsafe storede credentials |
| Web portals to configure the device mostly come with a default password | Unpatched bugs in the device can be a problem for the device | Hardcoded device credentials or credentials that can be read out can be used to clone a device or harm your service |
| Open ports that are not known to the user can open a backdoor to the device | | |

Microsoft

# Securing Windows IoT Devices



## Boot malware resistance with UEFI Secure Boot

Firmware enforces policy and only starts signed OS loader

## Secure device identity and health attestation

TPM support across all IoT SKUs brings strong device identity, secure key management and health attestation

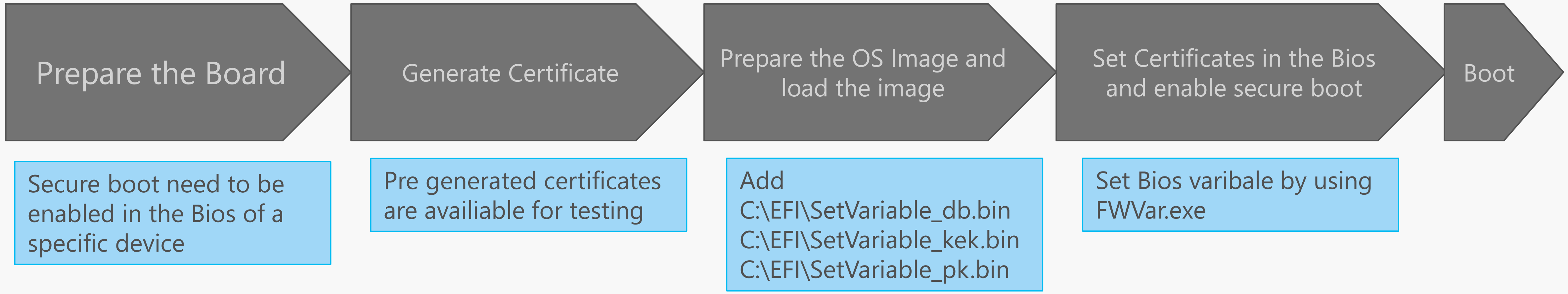
## Identity protection and access control

Supported by features like Microsoft Passport (2FA), Windows Hello, Credential Guard (virtualization-based security)

## Advanced lock-down capabilities

Supported in Windows 10 with AppLocker & Device Guard along with Enterprise Data Protection & BitLocker

# Enable Secure Boot


Prepare the Board
Generate Certificate
Prepare the OS Image and load the image
Set Certificates in the Bios and enable secure boot
Boot

Secure boot need to be enabled in the Bios of a specific device

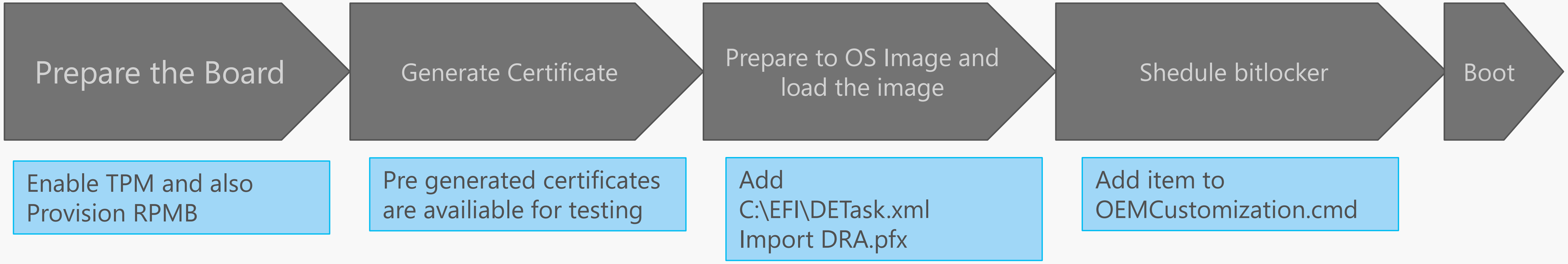
Pre generated certificates are availiable for testing

Add
C:\EFI\SetVariable_db.bin
C:\EFI\SetVariable_kek.bin
C:\EFI\SetVariable_pk.bin

Set Bios varibale by using FWVar.exe

https://developer.microsoft.com/en-us/windows/iot/docs/securebootandbitlocker

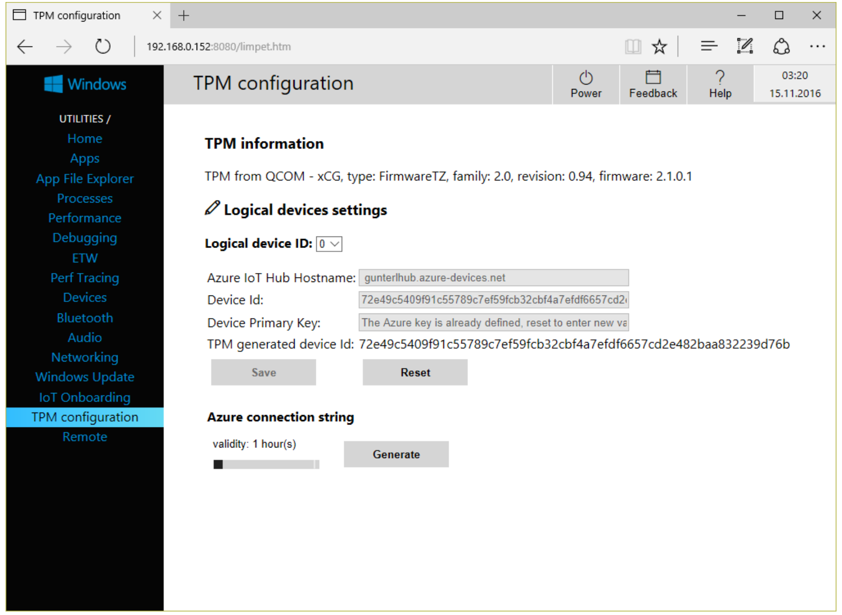
# Enable Bitlocker


Prepare the Board
Generate Certificate
Prepare to OS Image and load the image
Shedule bitlocker
Boot
Enable TPM and also Provision RPMB
Pre generated certificates are availiable for testing
Add C:\EFI\DETask.xml Import DRA.pfx
Add item to OEMCustomization.cmd

https://developer.microsoft.com/en-us/windows/iot/docs/securebootandbitlocker

# Connect to the cloud (Azure IOT Hub)

## Step 1: Provision the TPM



TPM configuration

**TPM information**

TPM from QCOM - xCG, type: FirmwareTZ, family: 2.0, revision: 0.94, firmware: 2.1.0.1

**Logical devices settings**

**Logical device ID:** 0

Azure IoT Hub Hostname: gunterlhub.azure-devices.net
Device Id: 72e49c5409f91c55789c7ef59fcb32cbf4a7efdf6657cd2
Device Primary Key: The Azure key is already defined, reset to enter new va
TPM generated device Id: 72e49c5409f91c55789c7ef59fcb32cbf4a7efdf6657cd2e482baa832239d76b

**Azure connection string**

validity: 1 hour(s)

```
tpm = new TpmDevice(0);
// reset TPM to clean previous
try
{
    Debug.WriteLine("Reset TPM...");
    tpm.Destroy();
}
catch (Exception ex)
{
    Debug.WriteLine("TPM was not initialized!");
}
Debug.WriteLine("TPM initialized");
string id = tpm.GetDeviceId();

//HWID is unique for this device.
string hwid = tpm.GetHardwareDeviceId();
Debug.WriteLine("TPM Hardware ID:" + hwid);

string hmackey = CryptoKeyGenerator.GenerateKey(32);
Debug.WriteLine("TPM hmackey:" + hmackey);

//provision the device.
tpm.Provision(hmackey, "gunterlhub.azure-devices.net", hwid);
```

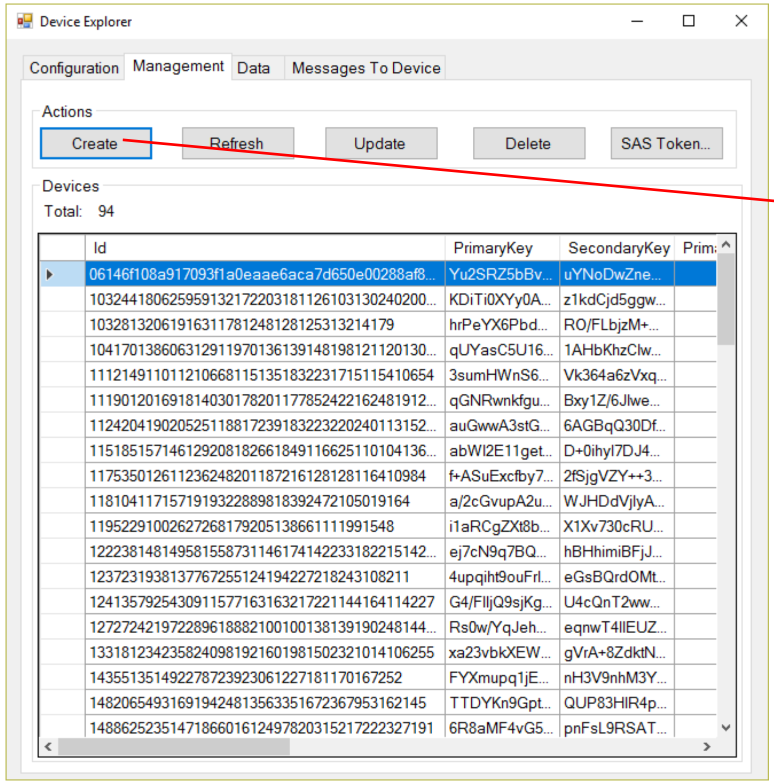
Use Portal or provisioning app

Microsoft

# Connect to the cloud (Azure IOT Hub)

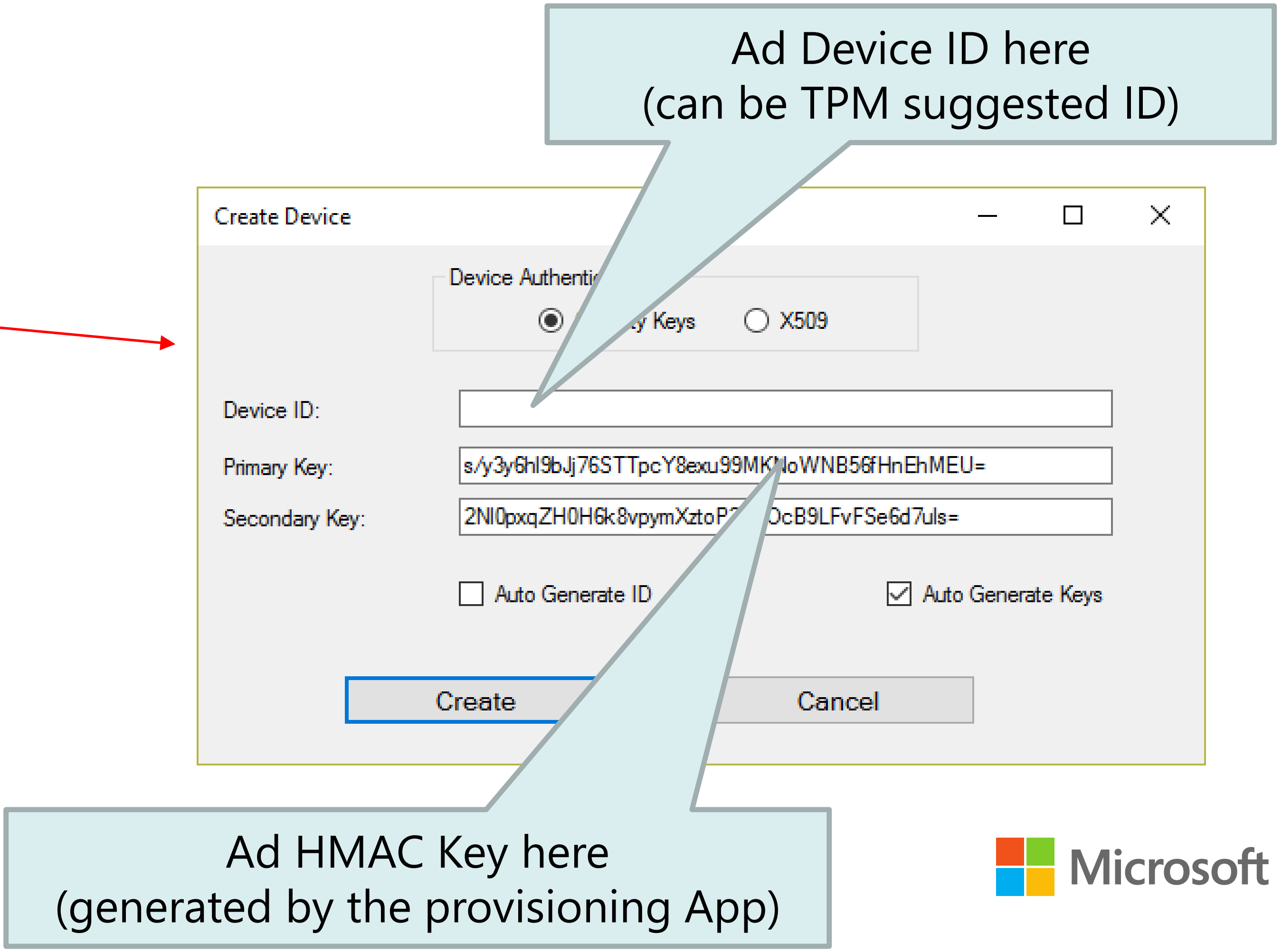
## Step 2: Create the device on IOT Hub

Manual Way using
Device Explorer



Devices
Total: 94

| Id | PrimaryKey | SecondaryKey | Prim: |
|---|---|---|---|
| 06146f108a917093f1a0eaae6aca7d650e00288af8... | Yu2SRZ5bBv... | uYNoDwZne... | |
| 103244180625959132172203181126103130240200... | KDiTi0XYy0A... | z1kdCjd5ggw... | |
| 10328132061916311781248128125313214179 | hrPeYX6Pbd... | RO/FLbjzM+... | |
| 104170138606312911970136139148198121120130... | qUYasC5U16... | 1AHbKhzClw... | |
| 1112149110112106681151351832231715115410654 | 3sumHWnS6... | Vk364a6zVxq... | |
| 111901201691814030178201177852422162481912... | qGNRwnkfgu... | Bxy1Z/6Jlwe... | |
| 112420419020525118817239183223220240113152... | auGwwA3stG... | 6AGBqQ30Df... | |
| 115185157146129208182661849116625110104136... | abWI2E11get... | D+0ihyl7DJ4... | |
| 117535012611236248201187216128128116410984 | f+ASuExcfby7... | 2fSjgVZY++3... | |
| 1181041171571919322889818392472105019164 | a/2cGvupA2u... | WJHDdVjlyA... | |
| 11952291002627268179205138661111991548 | i1aRCgZXt8b... | X1Xv730cRU... | |
| 122238148149581558731146174142233182215142... | ej7cN9q7BQ... | hBHhimiBFjJ... | |
| 1237231938137767255124194227218243108211 | 4upqiht9ouFrl... | eGsBQrdOMt... | |
| 1241357925430911577163163217221144164114227 | G4/FlljQ9sjKg... | U4cQnT2ww... | |
| 127272421972289618882100100138139190248144... | Rs0w/YqJeh... | eqnwT4lIEUZ... | |
| 1331812342358240981921601981502321014106255 | xa23vbkXEW... | gVrA+8ZdktN... | |
| 143551351492278723923061227181170167252 | FYXmupq1jE... | nH3V9nhM3Y... | |
| 148206549316919424813563351672367953162145 | TTDYKn9Gpt... | QUP83HIR4p... | |
| 1488625235147186601612497820315217222327191 | 6R8aMF4vG5... | pnFsL9RSAT... | |



Device ID:

Primary Key: s/y3y6hl9bJj76STTpcY8exu99MK[illegible]oWNB56fHnEhMEU=

Secondary Key: 2NI0pxqZH0H6k8vpymXztoP[illegible]OcB9LFvFSe6d7uls=

[ ] Auto Generate ID [✓] Auto Generate Keys

Ad Device ID here
(can be TPM suggested ID)

Ad HMAC Key here
(generated by the provisioning App)

Microsoft

# Connect to the cloud (Azure IOT Hub)

## Step 3: Use the IOT Hub in your application (test connection)

```
public static async Task<bool> TestHubConnection(bool sendRestartMessage, string restartMessage)
   {
       try
       {
           TpmDevice myDevice = new TpmDevice(0); // Use logical device 0 on the TPM
           string hubUri = myDevice.GetHostName();
           string deviceId = myDevice.GetDeviceId();
           string sasToken = myDevice.GetSASToken();
           if ((hubUri.Length == 0) || (sasToken.Length == 0)) return false;
       }
       catch (Exception ex)
       {
           return false;
       }

       if (sendRestartMessage)
       {
           return await SendDeviceToCloudMessageAsync(restartMessage);
       }
       return true;
   }
```

Microsoft

# Connect to the cloud (Azure IOT Hub)

## Step 3: Use the IOT Hub in your application (send message)

```csharp
public static async Task<bool> SendDeviceToCloudMessageAsync(string str)
{
    try
    {
        TpmDevice myDevice = new TpmDevice(0); // Use logical device 0 on the TPM
        string hubUri = myDevice.GetHostName();
        string deviceId = myDevice.GetDeviceId();
        string sasToken = myDevice.GetSASToken();

        var deviceClient = DeviceClient.Create(
            hubUri,
            AuthenticationMethodFactory.
                CreateAuthenticationWithToken(deviceId, sasToken), TransportType.Amqp);

        var message = new Message(Encoding.ASCII.GetBytes(str));

        await deviceClient.SendEventAsync(message);
        return true;
    }
    catch (Exception ex)
    {
        return false;
    }
}
```

Microsoft

# Connect to the cloud (Azure IOT Hub)

## Step 3: Use the IOT Hub in your application (receive message)

```
  public static async Task<string> ReceiveCloudToDeviceMessageAsync()
  {
      while (true)
      {
              TpmDevice myDevice = new TpmDevice(0); // Use logical device 0 on the TPM by default
              string hubUri = myDevice.GetHostName();
              string deviceId = myDevice.GetDeviceId();
              string sasToken = myDevice.GetSASToken();

              var deviceClient = DeviceClient.Create(hubUri,AuthenticationMethodFactory.CreateAuthenticationWithToken(deviceId, sasToken), TransportType.Amqp);

              Message receivedMessage = null;

              receivedMessage = await deviceClient.ReceiveAsync();

              if (receivedMessage != null)
              {
                  var messageData = Encoding.ASCII.GetString(receivedMessage.GetBytes());
                  await deviceClient.CompleteAsync(receivedMessage);
                  deviceClient.Dispose();
                  return messageData;
              }

              await Task.Delay(100);
          }
      }
}
```

Microsoft

# Unified Write Filter

- The Unified Write Filter (UWF) is a feature to protect physical storage media from data writes.
- UWF intercepts all write attempts to a protected volume and redirects those write attempts to a virtual overlay.
- improves the reliability and stability of your device and reduces

Enable UWF in the Image or Manual

Protect the volume

Protect Data Volume and add exclusions

Make sure that your app save data in an excluded data volume

Add **IOT_UNIFIED_WRITE_FILTER** feature id to the OEM Input file

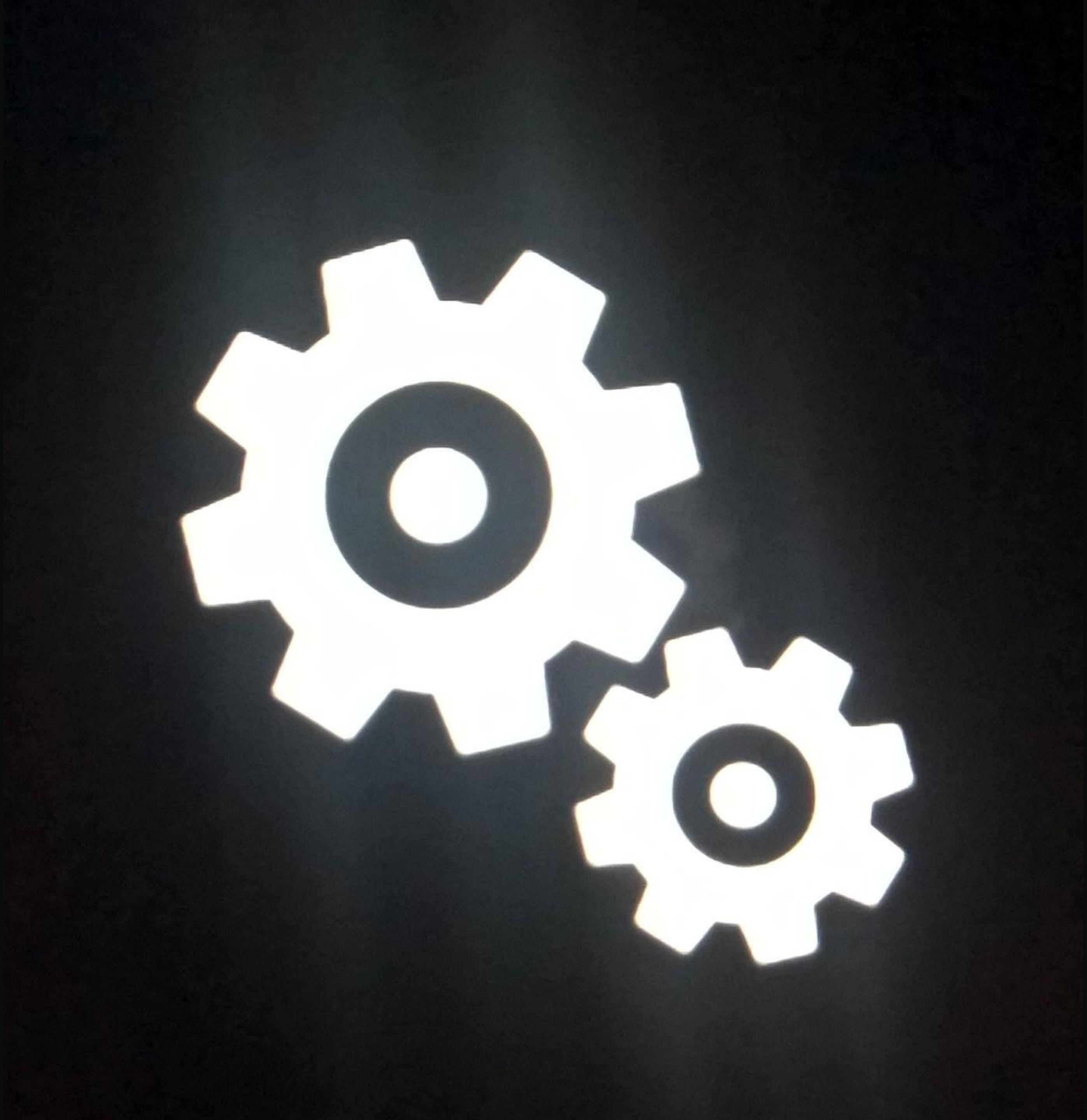
Create the image\FFU.
Read Create a basic image for instructions.

**uwfmgr.exe filter enable**
Enables the write filter
**uwfmgr.exe volume protect c:**
Protects the Volume C
**shutdown /r /t 0**
Restarts the device to make the write filter settings effective

https://developer.microsoft.com/en-us/windows/iot/docs/uwf

# Productization

Provisioning and Update

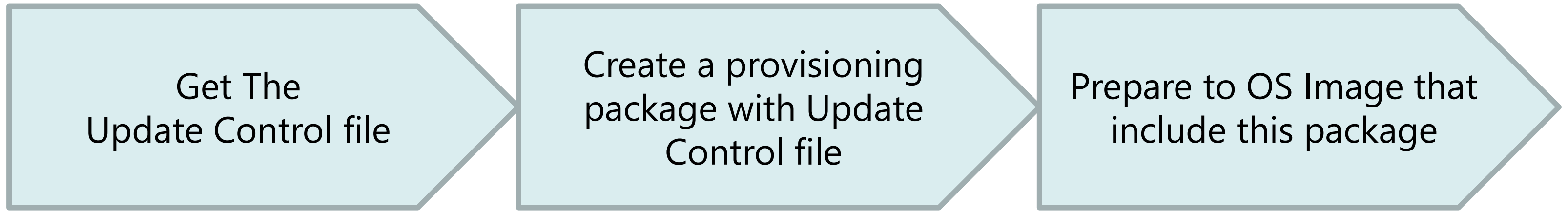


Microsoft

# Configure Windows Update

Windows 10 IoT (Core) Pro SKU provides the capability to control and schedule the windows update. This enables the policies related to Update such as:

Update/AllowAutoUpdate, Update/ScheduledInstallDay, Update/ScheduledInstallTime, Update/UpdateServiceUrl.


Get The Update Control file
Create a provisioning package with Update Control file
Prepare to OS Image that include this package

From *Windows 10 IoTCore 1607 version*, IoT Core is **free**.

download Windows 10 IoT Core Pro Update Control File.

https://developer.microsoft.com/en-us/windows/iot/docs/createiotcorepro

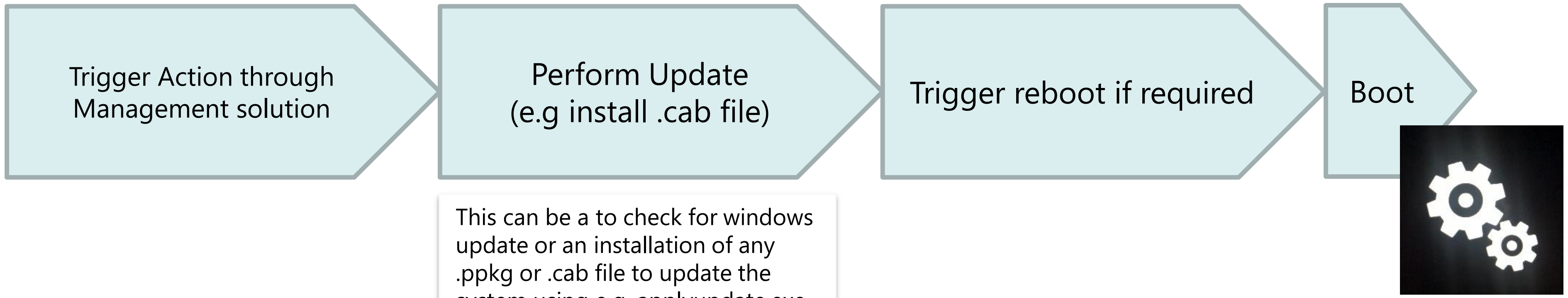
Microsoft

# Managing Windows 10 IoT Core Devices

Windows 10 IoT Core devices can be managed using a traditional OMA DM MDM server that supports certificate based enrollment or using Azure IoT Hub's Device Management (currently in preview).

*Learn more about MDM and Windows 10 here.*

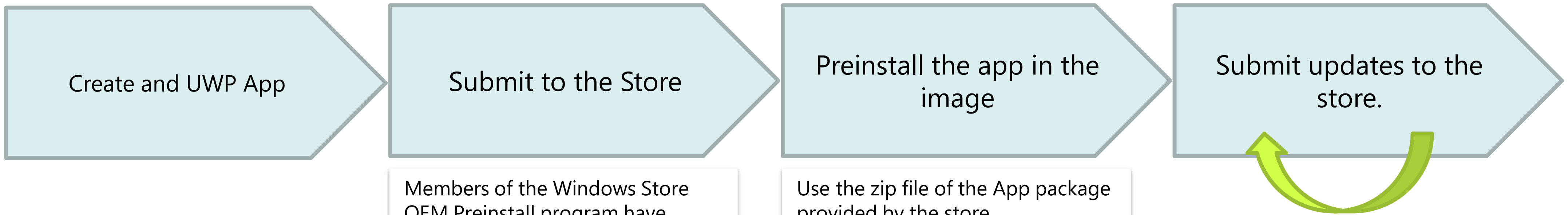
*Learn more about Azure IoT Hub Device Management here.*


Trigger Action through Management solution
Perform Update (e.g install .cab file)
Trigger reboot if required
Boot

This can be a to check for windows update or an installation of any .ppkg or .cab file to update the system using e.g. applyupdate.exe or even trigger a full FFU update.

## Installing and Servicing apps on Windows 10 IoT Core (Pro) using the Store

Microsoft makes it easy for OEMs to install and service UWP apps on Windows 10 IoT Core through the Universal Store.
All store signed apps on Windows 10 devices are capable of receiving updates directly from the store.



Members of the Windows Store OEM Preinstall program have special permissions in their Windows Dev Center account that allows a store signed version of an app to be downloaded.

Exceptions for apps using Headless mode or restricted Capabilities can be made.

Use the zip file of the App package provided by the store.

https://developer.microsoft.com/en-us/windows/iot/docs/store

# Create and Install Packages manually

## Install the Tools

- Windows Assessment and Deployment Kit(Windows ADK)
- IoT Core ADK Add-Ons

## Set up your environment

- Edit \IoT-ADK-AddonKit\Tools\setOEM.cmd to set the OEM_NAME
- Launch IoTCoreShell.cmd
  (this one launches in the elevated prompt )
- Select the required architecture in the Set Environment for Architecture prompt
- Install test signing certificates using InstallOEMCerts. This is required only once for the PC

## Create the Package

Create a **package definition xml file** (.pkg.xml file), and specify the files and reg keys you want to add.
Learn more at Specifying components in a package and Elements and Attributes of a package

Build the Package: buildpkg.cmd filename.pkg.xml

**The .cab file will be created** in the build directory \IoT-ADK-AddonKit\Build\<arch>\pkgs

Microsoft

# Create a package with files and reg keys

```xml
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
  Owner="OEMName"          OwnerType="OEM"
  ReleaseType="Test"       Platform="PlaformName"
  Component="ComponentName" SubComponent="SubName">
  <Components>
    <OSComponent>
      <Files>
        <File Source="$(_RELEASEDIR)\test_file1.dll"/>
        <File Source="$(_RELEASEDIR)\toBeRenamed.dat"
          DestinationDir="$(runtime.system32)\test" Name="test.dat"/>
      </Files>
      <RegKeys>
        <RegKey KeyName="$(hklm.software)\OEMName\test">
          <RegValue Name="StringValue" Value="Test string" Type="REG_SZ"/>
          <RegValue Name="DWordValue" Value="12AB34CD" Type="REG_DWORD"/>
          <RegValue Name="BinaryValue" Value="12,AB,CD,EF" Type="REG_BINARY"/>
        </RegKey>
        <RegKey KeyName="$(hklm.software)\OEMName\EmptyKey"/>
      </RegKeys>
    </OSComponent>
  </Components>
</Package>
```

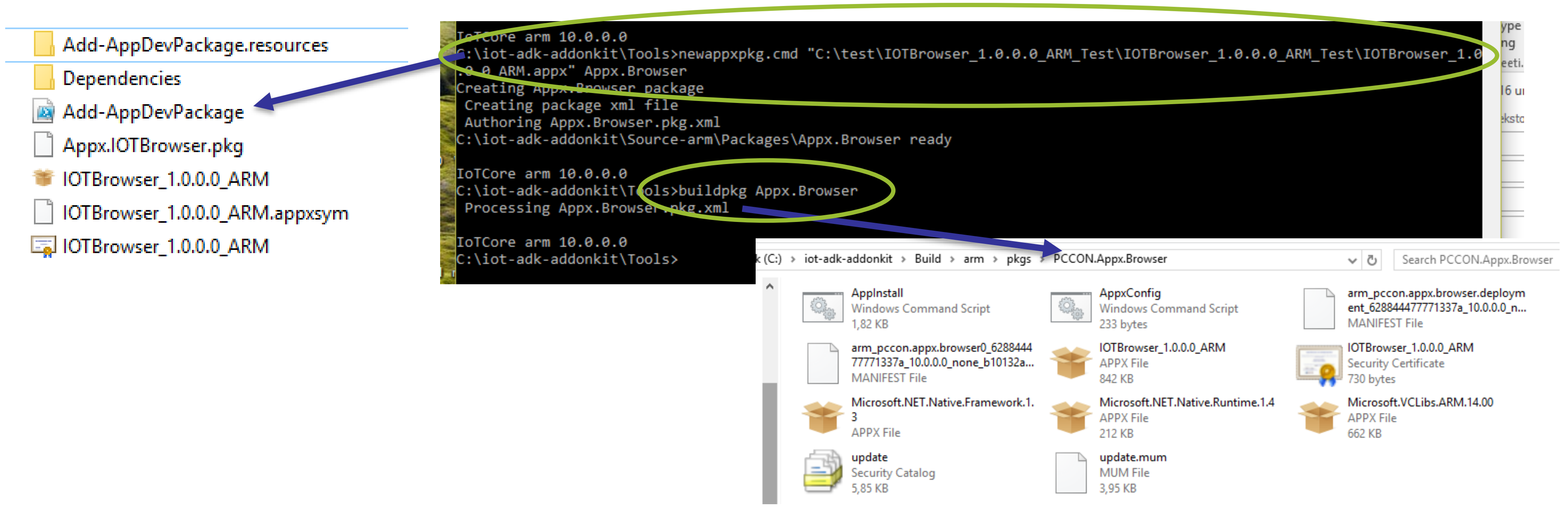
Microsoft

# Create an Appx package

Use appx2pkg.cmd or newappxpkg.cmd tool to generate the .pkg.xml file for a given appx file.

This tool expects the appx dependencies in the sub directory named "dependencies" in the folder containing the appx file.

https://msdn.microsoft.com/en-us/windows/hardware/commercialize/manufacture/iot/iot-core-manufacturing-guide



Add-AppDevPackage.resources
Dependencies
Add-AppDevPackage
Appx.IOTBrowser.pkg
IOTBrowser_1.0.0.0_ARM
IOTBrowser_1.0.0.0_ARM.appxsym
IOTBrowser_1.0.0.0_ARM

```
IoTCore arm 10.0.0.0
C:\iot-adk-addonkit\Tools>newappxpkg.cmd "C:\test\IOTBrowser_1.0.0.0_ARM_Test\IOTBrowser_1.0.0.0_ARM_Test\IOTBrowser_1.0
.0.0_ARM.appx" Appx.Browser
Creating Appx.Browser package
 Creating package xml file
 Authoring Appx.Browser.pkg.xml
C:\iot-adk-addonkit\Source-arm\Packages\Appx.Browser ready

IoTCore arm 10.0.0.0
C:\iot-adk-addonkit\Tools>buildpkg Appx.Browser
 Processing Appx.Browser.pkg.xml

IoTCore arm 10.0.0.0
C:\iot-adk-addonkit\Tools>
```

k (C:) › iot-adk-addonkit › Build › arm › pkgs › PCCON.Appx.Browser

| | | |
|---|---|---|
| AppInstall<br>Windows Command Script<br>1,82 KB | AppxConfig<br>Windows Command Script<br>233 bytes | arm_pccon.appx.browser.deployment_628844477771337a_10.0.0.0_n...<br>MANIFEST File |
| arm_pccon.appx.browser0_628844477771337a_10.0.0.0_none_b10132a...<br>MANIFEST File | IOTBrowser_1.0.0.0_ARM<br>APPX File<br>842 KB | IOTBrowser_1.0.0.0_ARM<br>Security Certificate<br>730 bytes |
| Microsoft.NET.Native.Framework.1.3<br>APPX File | Microsoft.NET.Native.Runtime.1.4<br>APPX File<br>212 KB | Microsoft.VCLibs.ARM.14.00<br>APPX File<br>662 KB |
| update<br>Security Catalog<br>5,85 KB | update.mum<br>MUM File<br>3,95 KB | |

# Download and install the Package

```
 public async Task StartDownloadandInstall(Uri url)
 {
    Uri source = url; //  new Uri("http://www.pccon.de/test/appupdate/update.main.zip");

    StorageFile destinationFile = await ApplicationData.Current.LocalFolder.CreateFileAsync(
                    "download.zip", CreationCollisionOption.GenerateUniqueName);

    BackgroundDownloader downloader = new BackgroundDownloader();
    DownloadOperation download = downloader.CreateDownload(source, destinationFile);
    await download.StartAsync();

    await UnzipFile(download.ResultFile.Path);

    StorageFolder localFolder = ApplicationData.Current.LocalFolder;
    StorageFolder t = null;
    try {
      t = await localFolder.GetFolderAsync("installer");
    }
    catch {
      t = null;
    }
    if (t!=null) {
      await t.DeleteAsync();
    }
    StorageFolder f = await localFolder.GetFolderAsync("update.main");
    await f.RenameAsync("installer");
    string path = localFolder.Path + "\\installer\\AppInstall\\appinstall.cmd";
    string s = "";

// REG ADD "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\EmbeddedMode\ProcessLauncher" /v AllowedExecutableFilesList /t REG_MULTI_SZ /d
"c:\windows\system32\applyupdate.exe\0c:\windows\system32\deployappx.exe\0c:\installer\appinstall.cmd\0c:\Data\Users\DefaultAccount\AppData\Local\Packages
\15c8ba7d-b8cc-46ee-84f1-ef0f27753fbe_0wy2ejr5nfw9j\LocalState\installer\AppInstall\appinstall.cmd\0"

    await App.ViewModel.RunProcess(path, s);
}
```

Microsoft

## OEM license requirements

The process of licensing Windows 10 IoT Core product and the OEM license agreement is provided at Windows 10 IoT Core Commercialization.

As part of signing the Windows 10 IoT Core OEM license agreement, you are required to meet these system requirements for the Windows 10 IoT Core device.

**SMBIOS Support**

The following are the minimum required fields in SMBIOS for IoTCore

- (Table 1, offset 04h) System Manufacturer
- (Table 1, offset 05h) System Product Name
- (Table 1, offset 19h) System SKU
- (Table 1, offset 1Bh) System Family

https://developer.microsoft.com/en-us/windows/iot/docs/oemlicenserequirements

## And finally..... Go To market

**Understand licensing requirements**

- See Windows 10 IoT Core Commercialization to understand the licensing requirements

**Create a retail image**

- Learn how to create a custom image at Windows 10 IoT Core manufacturing guide
- Learn how to lockdown your device at Building Secure Devices
- Learn how to configure your device for servicing at Service IoTCore

**Arrange for device manufacturing**

Get contact info for parts suppliers (SVs/IHVs/ODMs). Contact the supplier directly and follow their process to get components and licenses as necessary. Work directly with your SoC partner to locate an appropriate manufacturer.

- See Supported boards and SoCs

## Vielen Dank

Microsoft