Contents

Which AI solution is right for me?

12/3/2019 • 2 minutes to read • Edit Online

Microsoft offers several different AI solutions, which means you have several options at your disposal. But how do you choose which one to use for your application? Let's break it down.

I want to integrate a machine learning model into my application and run it on the device by taking full advantage of hardware acceleration

Windows Machine Learning is the right choice for you. These high-level WinRT APIs work on Windows 10 applications (UWP, desktop) and evaluate models directly on the device. You can even choose to take advantage of the device's GPU (if it has one) for better performance.

I want to integrate computer vision into my application and take advantage of platform optimizations

Windows Vision Skills is the way to go. This simple framework allows you to build custom vision applications that leverage hardware acceleration on edge devices. You can combine pre-built libraries to accomplish common image processing tasks and ML models for specialized tasks.

I want to have fuller control over resource utilization during model execution for high-intensive applications

DirectML is what you want. These DirectX-style APIs provide a programming paradigm that will feel familiar to C++ game developers, and allow you to take full advantage of the hardware.

I want to train, test, and deploy ML models with a framework that is familiar to a .NET developer

Check out ML.NET, a machine learning framework built for .NET developers.

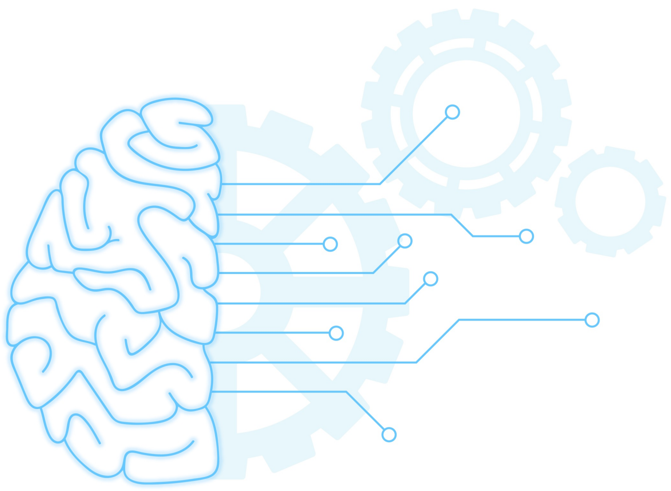
I want to leverage the power of the Azure cloud for training and deploying ML models

See What are the machine learning products at Microsoft? for a comprehensive list of the solutions available from Microsoft, including many products and services that run on Azure.

Windows Machine Learning

5/19/2020 • 4 minutes to read • Edit Online

Implement Machine Learning in your Windows apps using Windows ML — a high-performance, reliable API for deploying hardware-accelerated ML inferences on Windows devices.



Overview

Windows ML is built into the latest versions of Windows 10 and Windows Server 2019, and is also available as a NuGet package for down-level reach to Windows 8.1. Windows ML provides developers with the following advantages:

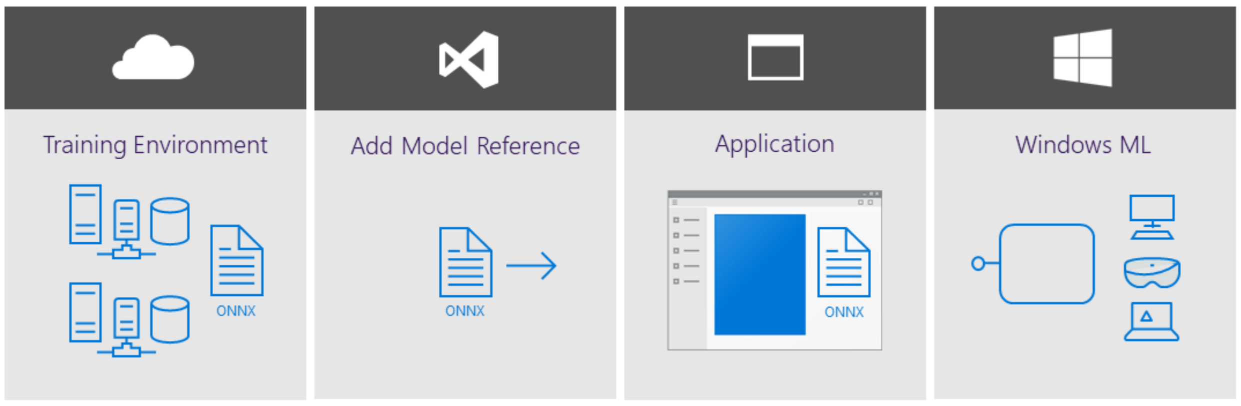
- **Ease of development:** With Windows ML built into the latest versions of Windows 10 and Windows Server 2019, all you need is Visual Studio and a trained ONNX model, which can be distributed along with the Windows application. Also, if you need to deliver your AI-based features to older versions of Windows (down to 8.1), Windows ML is also available as a NuGet package that you can distribute with your application.
- **Broad hardware support**: Windows ML allows you to write your ML workload once and automatically get highly optimized performance across different hardware vendors and silicon types, such as CPUs, GPUs, and AI accelerators. In addition, Windows ML guarantees consistent behavior across the range of supported hardware.
- **Low latency, real-time results:** ML models can be evaluated using the processing capabilities of the Windows device, enabling local, real-time analysis of large data volumes, such as images and video. Results are available quickly and efficiently for use in performance-intensive workloads like game engines, or background tasks such as indexing for search.
- **Increased flexibility:** The option to evaluate ML models locally on Windows devices lets you address a broader range of scenarios. For example, evaluation of ML models can run while the device is offline, or when faced with intermittent connectivity. This also lets you address scenarios where not all data can be sent to the cloud due to privacy or data sovereignty issues.
- **Reduced operational costs:** Training ML models in the cloud and then evaluating them locally on Windows devices can deliver significant savings in bandwidth costs, with only minimal data sent to the cloud—as might be needed for continual improvement of your ML model. Moreover, when deploying the ML model in a server scenario, developers can leverage Windows ML hardware acceleration to speed-up

model serving, reducing the number of machines needed in order to handle the workload.

Get Started

The process of incorporating trained ML models into your application code is simple, requiring just a few straightforward steps:

1. Get a trained Open Neural Network Exchange (ONNX) model, or convert models trained in other ML frameworks into ONNX with WinMLTools.
2. Add the ONNX model file to your application, or make it available in some other way on the target device.
3. Integrate the model into your application code, then build and deploy the application.



To start with the in-box Windows ML, go to Integrate a model into your app with Windows ML. You can also try out the sample apps in the Windows-Machine-Learning repo on GitHub.

If you want to use the NuGet package, please see Tutorial: Port an Existing WinML App to NuGet Package.

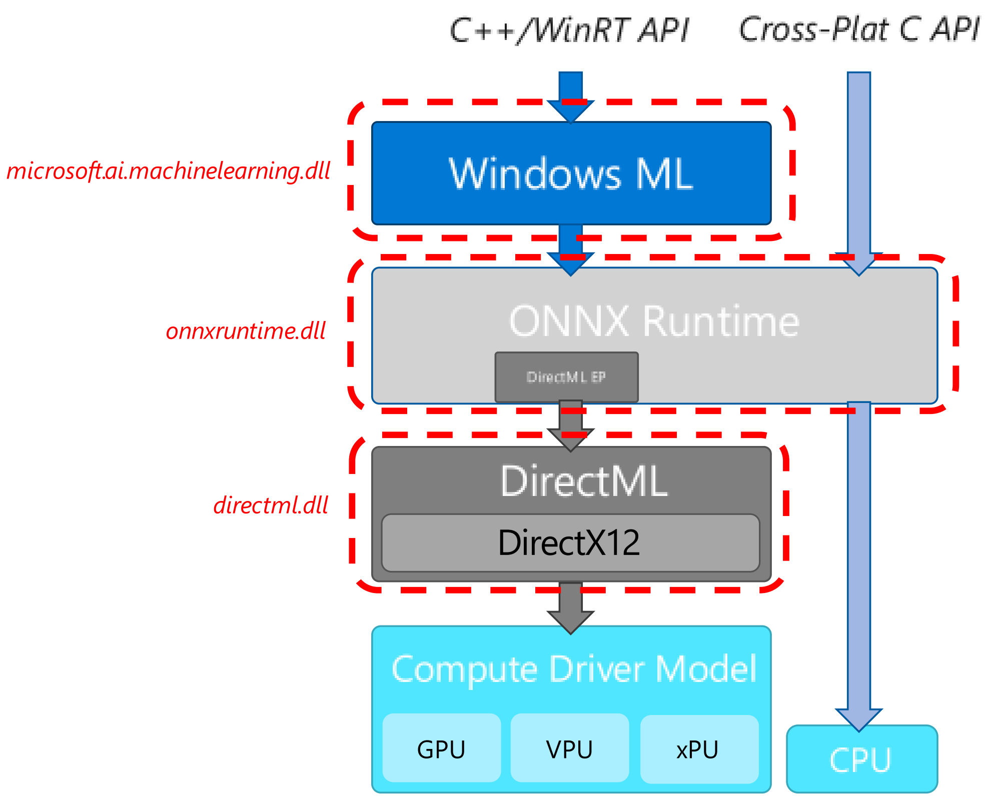
For the latest Windows ML features and fixes, see our release notes.

In-box vs NuGet WinML solutions

The table below highlights the availability, distribution, language support, servicing, and forward compatibility aspects of the In-Box and NuGet package for Windows ML.

	IN-BOX	NUGET
Availability	Windows 10 version 1809 or higher	Windows 8.1 or higher
Distribution	Built into the Windows SDK	Package and distribute as part of your application
Servicing	Microsoft-driven (customers benefit automatically)	Developer-driven
Forward compatibility	Automatically rolls forward with new features	Developer needs to update package manually

When your application runs with the in-box solution, the Windows ML runtime (which contains the ONNX Model Inference Engine) evaluates the trained model on the Windows 10 device (or Windows Server 2019 if targeting a server deployment). Windows ML handles the hardware abstraction, allowing developers to target a broad range of silicon—including CPUs, GPUs, and, in the future, AI accelerators. Windows ML hardware acceleration is built on top of DirectML, a high-performance, low-level API for running ML inferences that is part of the DirectX family.



For the NuGet package, these layers appear as binaries shown in the diagram below. Windows ML is built into the Microsoft.ai.machinelearning.dll. It does not contain an embedded ONNX runtime, instead the ONNX runtime is built into the file: onnxruntime.dll. The version included in the WindowsAI NuGet packages contains a DirectML EP embedded inside of it. The final binary, DirectML.dll, is the actual platform code as DirectML and is built on top of the Direct 3D and compute drivers that are built into Windows. All three of these binaries are included in the NuGet releases for you to distribute along with your applications.

Direct access to the onnxruntime.dll also allows you to target cross-platform scenarios while getting the same hardware agnostic acceleration that scales across all Windows devices.

Other machine learning solutions from Microsoft

Microsoft offers a variety of machine learning solutions to suit your needs. These solutions run in the cloud, on-premises, and locally on the device. See What are the machine learning product options from Microsoft? for more information.

NOTE

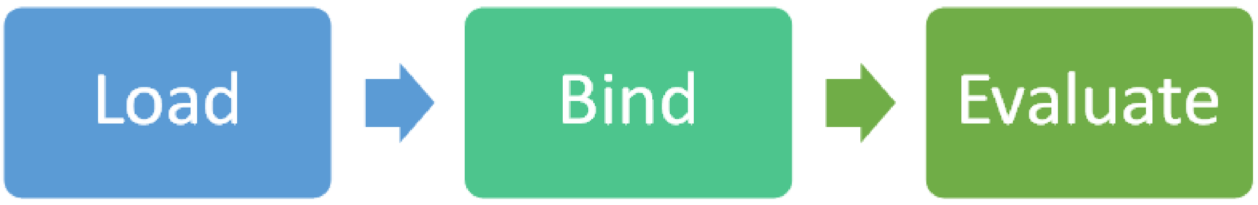
Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Tutorial: Create a Windows Machine Learning Desktop application (C++)

8/29/2019 • 8 minutes to read • Edit Online

Windows ML APIs can be leveraged to easily interact with machine learning models within C++ desktop (Win32) applications. Using the three steps of loading, binding, and evaluating, your application can benefit from the power of machine learning.



We will be creating a somewhat simplified version of the SqueezeNet Object Detection sample, which is available on GitHub. You can download the complete sample if you want to see what it will be like when you finish.

We'll be using C++/WinRT to access the WinML APIs. See C++/WinRT for more information.

In this tutorial, you'll learn how to:

- Load a machine learning model
- Load an image as a VideoFrame
- Bind the model's inputs and outputs
- Evaluate the model and print meaningful results

Prerequisites

- Visual Studio 2019 (or Visual Studio 2017, version 15.7.4 or later)
- Windows 10, version 1809 or later
- Windows SDK, build 17763 or later
- Visual Studio extension for C++/WinRT
  1. In Visual Studio, select **Tools > Extensions and Updates**.
  2. Select **Online** in the left pane and search for "WinRT" using the search box on the right.
  3. Select **C++/WinRT**, click **Download**, and close Visual Studio.
  4. Follow the installation instructions, then re-open Visual Studio.
- Windows-Machine-Learning Github repo (you can either download it as a ZIP file or clone to your machine)

Create the project

First, we will create the project in Visual Studio:

1. Select **File > New > Project** to open the **New Project** window.
2. In the left pane, select **Installed > Visual C++ > Windows Desktop**, and in the middle, select **Windows Console Application (C++/WinRT)**.
3. Give your project a **Name** and **Location**, then click **OK**.
4. In the **New Universal Windows Platform Project** window, set the **Target** and **Minimum Versions** both to

build 17763 or later, and click **OK**.

5. Make sure the dropdown menus in the top toolbar are set to **Debug** and either **x64** or **x86** depending on your computer's architecture.
6. Press **Ctrl+F5** to run the program without debugging. A terminal should open with some "Hello world" text. Press any key to close it.

Load the model

Next, we'll load the ONNX model into our program using LearningModel.LoadFromFilePath:

1. In **pch.h** (in the **Header Files** folder), add the following `include` statements (these give us access to all the APIs that we'll need):

```
#include <winrt/Windows.AI.MachineLearning.h>
#include <winrt/Windows.Foundation.Collections.h>
#include <winrt/Windows.Graphics.Imaging.h>
#include <winrt/Windows.Media.h>
#include <winrt/Windows.Storage.h>

#include <string>
#include <fstream>

#include <Windows.h>
```

2. In **main.cpp** (in the **Source Files** folder), add the following `using` statements:

```
using namespace Windows::AI::MachineLearning;
using namespace Windows::Foundation::Collections;
using namespace Windows::Graphics::Imaging;
using namespace Windows::Media;
using namespace Windows::Storage;

using namespace std;
```

3. Add the following variable declarations after the `using` statements:

```
// Global variables
hstring modelPath;
string deviceName = "default";
hstring imagePath;
LearningModel model = nullptr;
LearningModelDeviceKind deviceKind = LearningModelDeviceKind::Default;
LearningModelSession session = nullptr;
LearningModelBinding binding = nullptr;
VideoFrame imageFrame = nullptr;
string labelsFilePath;
vector<string> labels;
```

4. Add the following forward declarations after your global variables:

```
// Forward declarations
void LoadModel();
VideoFrame LoadImageFile(hstring filePath);
void BindModel();
void EvaluateModel();
void PrintResults(IVectorView<float> results);
void LoadLabels();
```

5. In **main.cpp**, remove the "Hello world" code (everything in the `main` function after `init_apartment`).

6. Find the **SqueezeNet.onnx** file in your local clone of the **Windows-Machine-Learning** repo. It should be located in **\Windows-Machine-Learning\SharedContent\models**.

7. Copy the file path and assign it to your `modelPath` variable where we defined it at the top. Remember to prefix the string with an `L` to make it a wide character string so that it works properly with `hstring`, and to escape any backslashes (`\`) with an extra backslash. For example:

```
hstring modelPath = L"C:\\Repos\\Windows-Machine-Learning\\SharedContent\\models\\SqueezeNet.onnx";
```

8. First, we'll implement the `LoadModel` method. Add the following method after the `main` method. This method loads the model and outputs how long it took:

```
void LoadModel()
{
     // load the model
     printf("Loading modelfile '%ws' on the '%s' device\n", modelPath.c_str(), deviceName.c_str());
     DWORD ticks = GetTickCount();
     model = LearningModel::LoadFromFilePath(modelPath);
     ticks = GetTickCount() - ticks;
     printf("model file loaded in %d ticks\n", ticks);
}
```

9. Finally, call this method from the `main` method:

```
LoadModel();
```

10. Run your program without debugging. You should see that your model loads successfully!

Load the image

Next, we'll load the image file into our program:

1. Add the following method. This method will load the image from the given path and create a VideoFrame from it:

```
VideoFrame LoadImageFile(hstring filePath)
{
    printf("Loading the image...\n");
    DWORD ticks = GetTickCount();
    VideoFrame inputImage = nullptr;

    try
    {
        // open the file
        StorageFile file = StorageFile::GetFileFromPathAsync(filePath).get();
        // get a stream on it
        auto stream = file.OpenAsync(FileAccessMode::Read).get();
        // Create the decoder from the stream
        BitmapDecoder decoder = BitmapDecoder::CreateAsync(stream).get();
        // get the bitmap
        SoftwareBitmap softwareBitmap = decoder.GetSoftwareBitmapAsync().get();
        // load a videoframe from it
        inputImage = VideoFrame::CreateWithSoftwareBitmap(softwareBitmap);
    }
    catch (...)
    {
        printf("failed to load the image file, make sure you are using fully qualified paths\r\n");
        exit(EXIT_FAILURE);
    }

    ticks = GetTickCount() - ticks;
    printf("image file loaded in %d ticks\n", ticks);
    // all done
    return inputImage;
}
```

2. Add a call to this method in the `main` method:

```
imageFrame = LoadImageFile(imagePath);
```

3. Find the **media** folder in your local clone of the **Windows-Machine-Learning** repo. It should be located at **\Windows-Machine-Learning\SharedContent\media**.

4. Choose one of the images in that folder, and assign its file path to the `imagePath` variable where we defined it at the top. Remember to prefix it with an `L` to make it a wide character string, and to escape any backslashes with another backslash. For example:

```
hstring imagePath = L"C:\\Repos\\Windows-Machine-Learning\\SharedContent\\media\\kitten_224.png";
```

5. Run the program without debugging. You should see the image loaded successfully!

Bind the input and output

Next, we'll create a session based on the model and bind the input and output from the session using LearningModelBinding.Bind. For more information on binding, see Bind a model.

1. Implement the `BindModel` method. This creates a session based on the model and device, and a binding based on that session. We then bind the inputs and outputs to variables we've created using their names. We happen to know ahead of time that the input feature is named "data_0" and the output feature is named "softmaxout_1". You can see these properties for any model by opening them in Netron, an online model visualization tool.

```
void BindModel()
{
    printf("Binding the model...\n");
    DWORD ticks = GetTickCount();

    // now create a session and binding
    session = LearningModelSession{ model, LearningModelDevice(deviceKind) };
    binding = LearningModelBinding{ session };
    // bind the intput image
    binding.Bind(L"data_0", ImageFeatureValue::CreateFromVideoFrame(imageFrame));
    // bind the output
    vector<int64_t> shape({ 1, 1000, 1, 1 });
    binding.Bind(L"softmaxout_1", TensorFloat::Create(shape));

    ticks = GetTickCount() - ticks;
    printf("Model bound in %d ticks\n", ticks);
}
```

2. Add a call to `BindModel` from the `main` method:

```
BindModel();
```

3. Run the program without debugging. The model's inputs and outputs should be bound successfully. We're almost there!

Evaluate the model

We're now on the last step in the diagram at the beginning of this tutorial, **Evaluate**. We'll evaluate the model using LearningModelSession.Evaluate:

1. Implement the `EvaluateModel` method. This method takes our session and evaluates it using our binding and a correlation ID. The correlation ID is something we could possibly use later to match a particular evaluation call to the output results. Again, we know ahead of time that the output's name is "softmaxout_1".

```
void EvaluateModel()
{
    // now run the model
    printf("Running the model...\n");
    DWORD ticks = GetTickCount();

    auto results = session.Evaluate(binding, L"RunId");

    ticks = GetTickCount() - ticks;
    printf("model run took %d ticks\n", ticks);

    // get the output
    auto resultTensor = results.Outputs().Lookup(L"softmaxout_1").as<TensorFloat>();
    auto resultVector = resultTensor.GetAsVectorView();
    PrintResults(resultVector);
}
```

2. Now let's implement `PrintResults`. This method gets the top three probabilities for what object could be in the image, and prints them:

```
void PrintResults(IVectorView<float> results)
{
    // load the labels
    LoadLabels();
    // Find the top 3 probabilities
    vector<float> topProbabilities(3);
    vector<int> topProbabilityLabelIndexes(3);
    // SqueezeNet returns a list of 1000 options, with probabilities for each, loop through all
    for (uint32_t i = 0; i < results.Size(); i++)
    {
        // is it one of the top 3?
        for (int j = 0; j < 3; j++)
        {
            if (results.GetAt(i) > topProbabilities[j])
            {
                topProbabilityLabelIndexes[j] = i;
                topProbabilities[j] = results.GetAt(i);
                break;
            }
        }
    }
    // Display the result
    for (int i = 0; i < 3; i++)
    {
        printf("%s with confidence of %f\n", labels[topProbabilityLabelIndexes[i]].c_str(),
topProbabilities[i]);
    }
}
```

3. We also need to implement `LoadLabels`. This method opens the labels file that contains all of the different objects that the model can recognize, and parses it:

```
void LoadLabels()
{
    // Parse labels from labels file.  We know the file's entries are already sorted in order.
    ifstream labelFile{ labelsFilePath, ifstream::in };
    if (labelFile.fail())
    {
        printf("failed to load the %s file.  Make sure it exists in the same folder as the app\r\n",
labelsFilePath.c_str());
        exit(EXIT_FAILURE);
    }

    std::string s;
    while (std::getline(labelFile, s, ','))
    {
        int labelValue = atoi(s.c_str());
        if (labelValue >= labels.size())
        {
            labels.resize(labelValue + 1);
        }
        std::getline(labelFile, s);
        labels[labelValue] = s;
    }
}
```

4. Locate the **Labels.txt** file in your local clone of the **Windows-Machine-Learning** repo. It should be in **\Windows-Machine-Learning\Samples\SqueezeNetObjectDetection\Desktop\cpp**.

5. Assign this file path to the `labelsFilePath` variable where we defined it at the top. Make sure to escape any backslashes with another backslash. For example:

```
string labelsFilePath = "C:\\Repos\\Windows-Machine-
Learning\\Samples\\SqueezeNetObjectDetection\\Desktop\\cpp\\Labels.txt";
```

6. Add a call to `EvaluateModel` in the `main` method:

```
EvaluateModel();
```

7. Run the program without debugging. It should now correctly recognize what's in the image! Here is an example of what it might output:

```
Loading modelfile 'C:\Repos\Windows-Machine-Learning\SharedContent\models\SqueezeNet.onnx' on the
'default' device
model file loaded in 250 ticks
Loading the image...
image file loaded in 78 ticks
Binding the model...Model bound in 15 ticks
Running the model...
model run took 16 ticks
tabby, tabby cat with confidence of 0.931461
Egyptian cat with confidence of 0.065307
Persian cat with confidence of 0.000193
```

Next steps

Hooray, you've got object detection working in a C++ desktop application! Next, you can try using command line arguments to input the model and image files rather than hardcoding them, similar to what the sample on GitHub does. You could also try running the evaluation on a different device, like the GPU, to see how the performance differs.

Play around with the other samples on GitHub and extend them however you like!

See also

- Windows ML samples (GitHub)
- Windows.AI.MachineLearning Namespace
- Windows ML

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Tutorial: Create a Windows Machine Learning Desktop application (Python)

8/29/2019 • 2 minutes to read • Edit Online

NOTE

Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

With Windows Machine Learning, you can create a Windows 10 desktop application in Python, and run evaluations with ONNX models locally on the device. The Python/WinRT projection allows you to use Windows Runtime APIs (including WinML) in a language you're already familiar with.

See Tutorial: Create a Windows Machine Learning application with Python/WinRT to see how to create a simple WinML application in Python. This tutorial corresponds to Tutorial: Create a Windows Machine Learning Desktop application (C++), focusing on the differences between the two.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Tutorial: Create a Windows Machine Learning UWP application (C#)

8/29/2019 • 6 minutes to read • Edit Online

In this tutorial, we'll build a simple Universal Windows Platform application that uses a trained machine learning model to recognize a numeric digit drawn by the user. This tutorial primarily focuses on how to load and use Windows ML in your UWP application.

The following video walks through the sample that this tutorial is based on.

If you'd prefer to simply look at the code of the finished tutorial, you can find it on the WinML GitHub repository. It's also available in C++/CX.

Prerequisites

- Windows 10 (Version 1809 or higher)
- Windows 10 SDK (Build 17763 or higher)
- Visual Studio 2019 (or Visual Studio 2017, version 15.7.4 or later)
- Windows Machine Learning Code Generator extension for Visual Studio 2019 or 2017
- Some basic UWP and C# knowledge

1. Open the project in Visual Studio

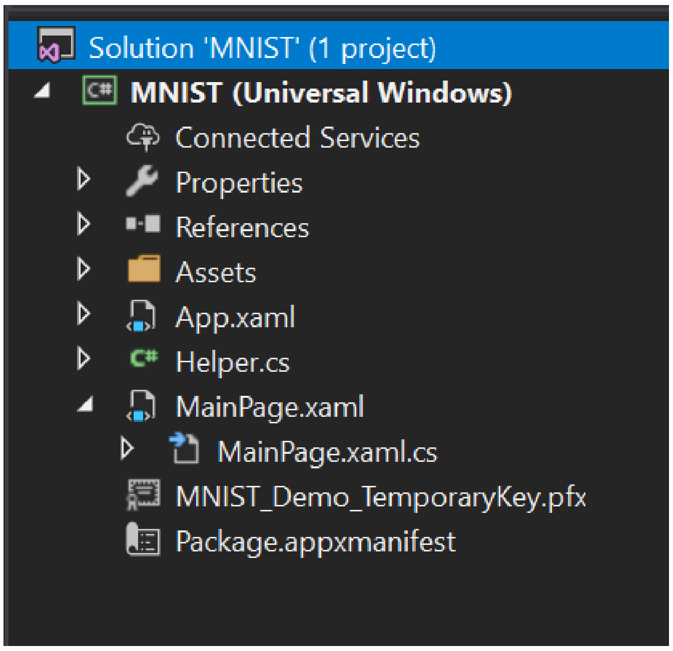
Once you've downloaded the project from GitHub, launch Visual Studio and open the **MNIST_Demo.sln** file (it should be located at **<Path to repo>\Windows-Machine-Learning\Samples\MNIST\Tutorial\cs**). If the solution is shown as unavailable, you'll need to right-click the project in the **Solution Explorer** and select **Reload Project**.

We've provided a template with implemented XAML controls and events, including:

- An InkCanvas to draw the digit.
- Buttons to interpret the digit and clear the canvas.
- Helper routines to convert the **InkCanvas** output to a VideoFrame.

Inside the **Solution Explorer**, the project has three main code files:

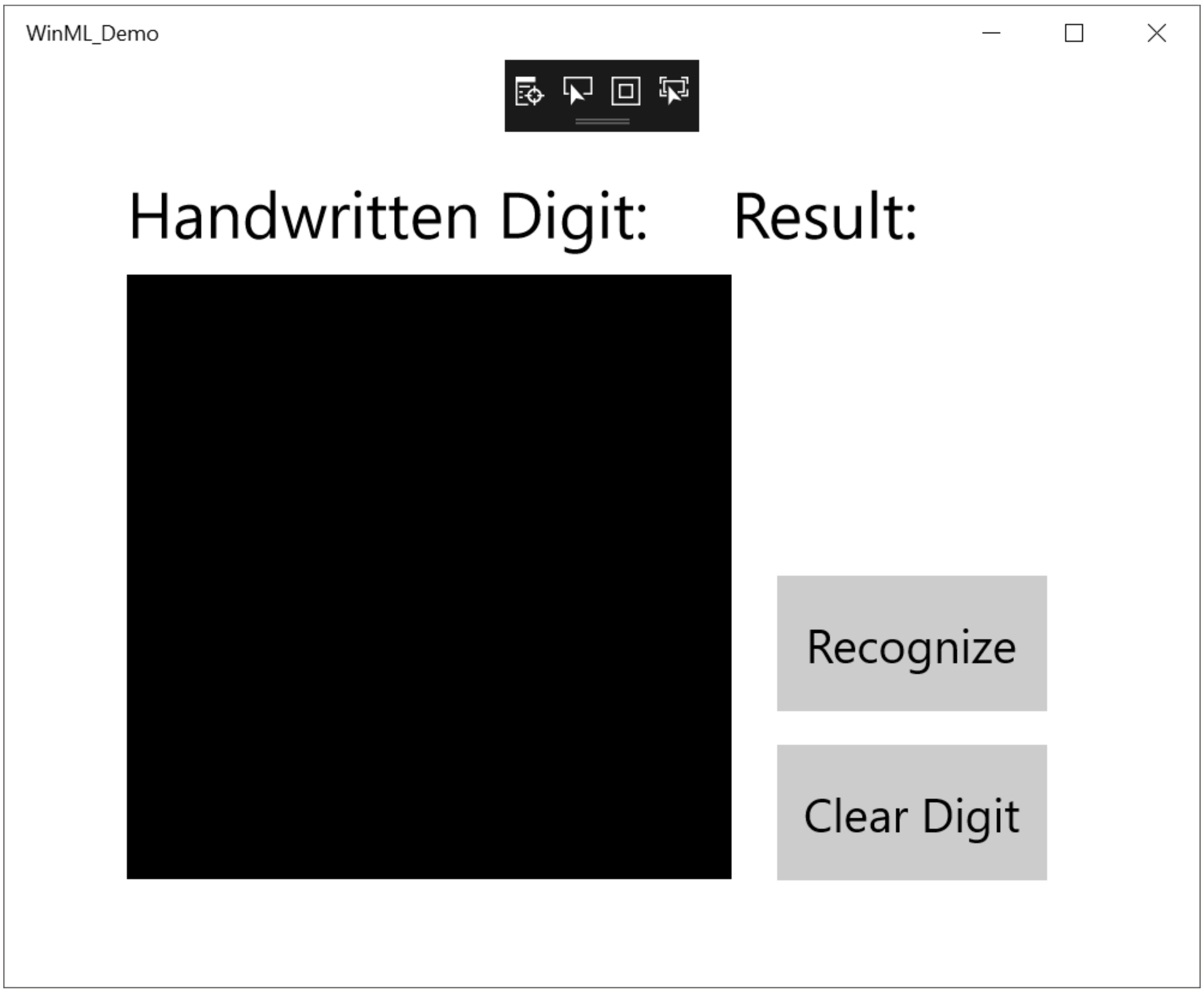
- **MainPage.xaml** - All of our XAML code to create the UI for the **InkCanvas**, buttons, and labels.
- **MainPage.xaml.cs** - Where our application code lives.
- **Helper.cs** - Helper routines to crop and convert image formats.


Solution 'MNIST' (1 project)
MNIST (Universal Windows)
Connected Services
Properties
References
Assets
App.xaml
Helper.cs
MainPage.xaml
MainPage.xaml.cs
MNIST_Demo_TemporaryKey.pfx
Package.appxmanifest

2. Build and run the project

In the Visual Studio toolbar, change the **Solution Platform** to **x64** to run the project on your local machine if your device is 64-bit, or **x86** if it's 32-bit. (You can check in the Windows Settings app: **System > About > Device specifications > System type**.)

To run the project, click the **Start Debugging** button on the toolbar, or press **F5**. The application should show an **InkCanvas** where users can write a digit, a **Recognize** button to interpret the number, an empty label field where the interpreted digit will be displayed as text, and a **Clear Digit** button to clear the **InkCanvas**.



> **NOTE**
>
> If the project won't build, you might need to change the project's deployment target version. Right-click the project in the **Solution Explorer** and select **Properties**. In the **Application** tab, set the **Target version** and **Min version** to match your OS and SDK.

> **NOTE**
>
> If you get a warning that the application is already installed, just select **Yes** to continue with deployment. You may need to close Visual Studio and re-open if it still doesn't work.

3. Download a model

Next, let's get a machine learning model to add to our application. For this tutorial, we'll use a pre-trained MNIST model that was trained with the Microsoft Cognitive Toolkit (CNTK) and exported to ONNX format.

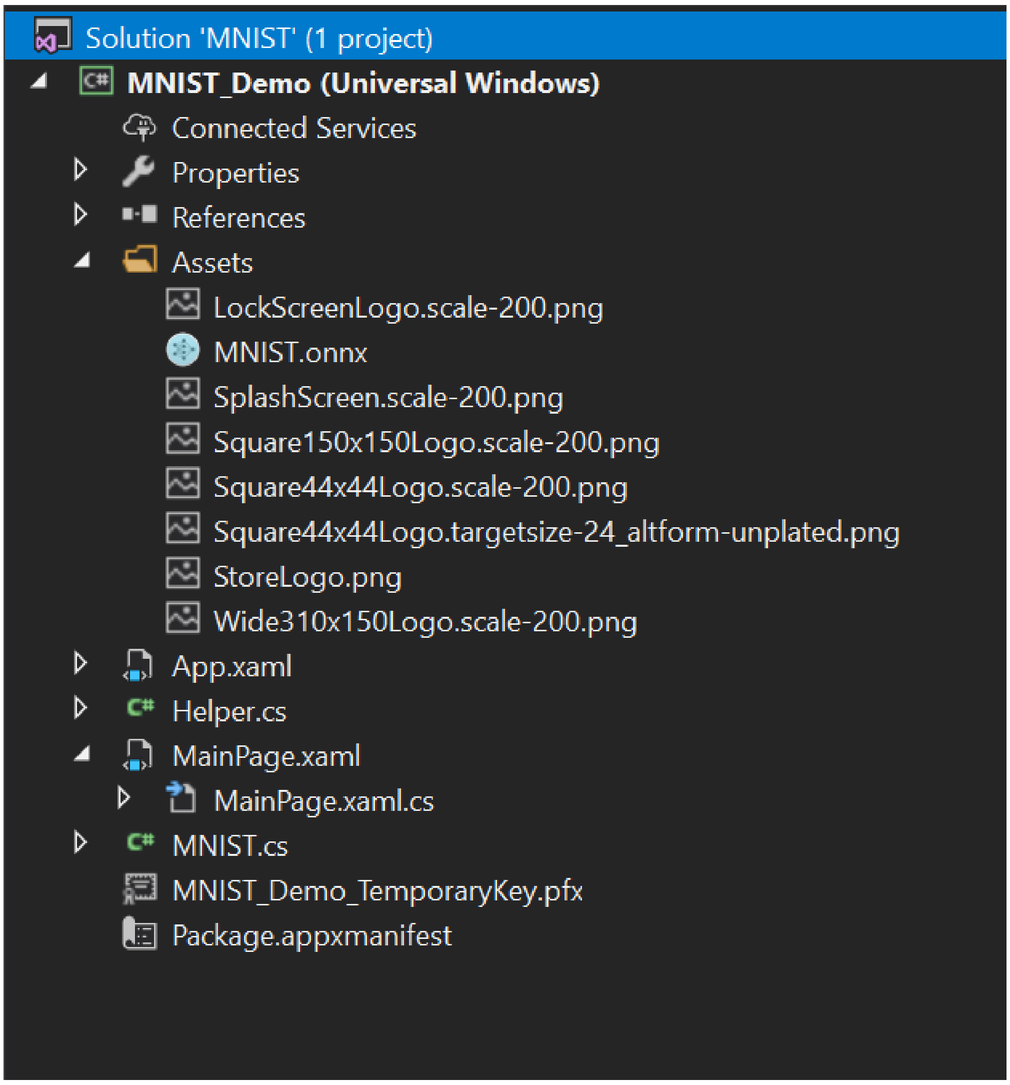
The MNIST model has already been included in your **Assets** folder, and you will need to add it to your application as an existing item. You can also download the pre-trained model from the ONNX Model Zoo on GitHub.

4. Add the model

Right click on the **Assets** folder in the **Solution Explorer**, and select **Add** > **Existing Item**. Point the file picker to the location of your ONNX model, and click **Add**.

The project should now have two new files:

- **mnist.onnx** - Your trained model.
- **mnist.cs** - The Windows ML-generated code.



```
Solution 'MNIST' (1 project)
◢ MNIST_Demo (Universal Windows)
    Connected Services
  ▷ Properties
  ▷ References
  ◢ Assets
      LockScreenLogo.scale-200.png
      MNIST.onnx
      SplashScreen.scale-200.png
      Square150x150Logo.scale-200.png
      Square44x44Logo.scale-200.png
      Square44x44Logo.targetsize-24_altform-unplated.png
      StoreLogo.png
      Wide310x150Logo.scale-200.png
  ▷ App.xaml
  ▷ Helper.cs
  ◢ MainPage.xaml
    ▷ MainPage.xaml.cs
  ▷ MNIST.cs
    MNIST_Demo_TemporaryKey.pfx
    Package.appxmanifest
```

To make sure the model builds when we compile our application, right click on the **mnist.onnx** file, and select **Properties**. For **Build Action**, select **Content**.

Now, let's take a look at the newly generated code in the **mnist.cs** file. We have three classes:

- **mnistModel** creates the machine learning model representation, creates a session on the system default device, binds the specific inputs and outputs to the model, and evaluates the model asynchronously.
- **mnistInput** initializes the input types that the model expects. In this case, the input expects an ImageFeatureValue.
- **mnistOutput** initializes the types that the model will output. In this case, the output will be a list called **Plus214_Output_0** of type TensorFloat.

We'll now use these classes to load, bind, and evaluate the model in our project.

5. Load, bind, and evaluate the model

For Windows ML applications, the pattern we want to follow is: Load > Bind > Evaluate.

1. Load the machine learning model.
2. Bind inputs and outputs to the model.
3. Evaluate the model and view results.

We'll use the interface code generated in **mnist.cs** to load, bind, and evaluate the model in our application.

First, in **MainPage.xaml.cs**, let's instantiate the model, inputs, and outputs. Add the following member variables to the **MainPage** class:

```
private mnistModel ModelGen;
private mnistInput ModelInput = new mnistInput();
private mnistOutput ModelOutput;
```

Then, in **LoadModelAsync**, we'll load the model. This method should be called before we use any of the model's methods (that is, on **MainPage**'s Loaded event, at an OnNavigatedTo override, or anywhere before **recognizeButton_Click** is called). The **mnistModel** class represents the MNIST model and creates the session on the system default device. To load the model, we call the **CreateFromStreamAsync** method, passing in the ONNX file as the parameter.

```
private async Task LoadModelAsync()
{
    // Load a machine learning model
    StorageFile modelFile = await StorageFile.GetFileFromApplicationUriAsync(new Uri($"ms-
appx:///Assets/mnist.onnx"));
    ModelGen = await mnistModel.CreateFromStreamAsync(modelFile as IRandomAccessStreamReference);
}
```

NOTE

If you get red underlines under **IRandomAccessStreamReference**, you need to include its namespace. Put your cursor over it, press **Ctrl + .** and select **using Windows.Storage.Streams** from the drop-down menu.

Next, we want to bind our inputs and outputs to the model. The generated code also includes **mnistInput** and **mnistOutput** wrapper classes. The **mnistInput** class represents the model's expected inputs, and the **mnistOutput** class represents the model's expected outputs.

To initialize the model's input object, call the **mnistInput** class constructor, passing in your application data, and make sure that your input data matches the input type that your model expects. The **mnistInput** class expects an **ImageFeatureValue**, so we use a helper method to get an **ImageFeatureValue** for the input.

Using our included helper functions in **helper.cs**, we will copy the contents of the **InkCanvas**, convert it to type **ImageFeatureValue**, and bind it to our model.

```
private async void recognizeButton_Click(object sender, RoutedEventArgs e)
{
    // Bind model input with contents from InkCanvas
    VideoFrame vf = await helper.GetHandWrittenImage(inkGrid);
    ModelInput.Input3 = ImageFeatureValue.CreateFromVideoFrame(vf);
}
```

For output, we simply call **EvaluateAsync** with the specified input. Once your inputs are initialized, call the model's **EvaluateAsync** method to evaluate your model on the input data. **EvaluateAsync** binds your inputs and outputs to the model object and evaluates the model on the inputs.

Since the model returns an output tensor, we'll first want to convert it to a friendly datatype, and then parse the returned list to determine which digit had the highest probability and display that one.

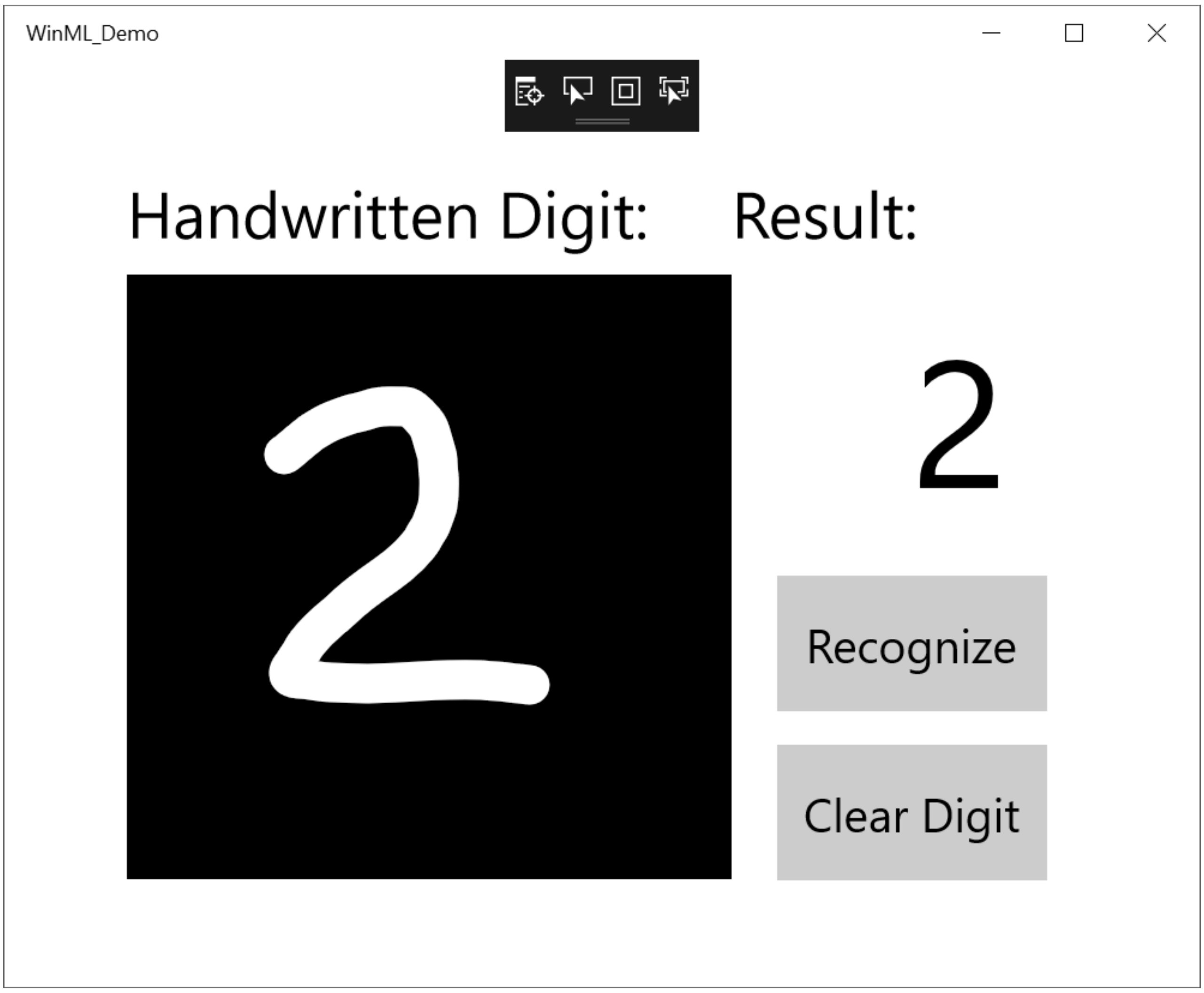
```
private async void recognizeButton_Click(object sender, RoutedEventArgs e)
{
    // Bind model input with contents from InkCanvas
    VideoFrame vf = await helper.GetHandWrittenImage(inkGrid);
    ModelInput.Input3 = ImageFeatureValue.CreateFromVideoFrame(vf);

    // Evaluate the model
    ModelOutput = await ModelGen.EvaluateAsync(ModelInput);

    // Convert output to datatype
    IReadOnlyList<float> vectorImage = ModelOutput.Plus214_Output_0.GetAsVectorView();
    IList<float> imageList = vectorImage.ToList();

    // Query to check for highest probability digit
    var maxIndex = imageList.IndexOf(imageList.Max());

    // Display the results
    numberLabel.Text = maxIndex.ToString();
}
```

Finally, we'll want to clear out the **InkCanvas** to allow users to draw another number.

```
private void clearButton_Click(object sender, RoutedEventArgs e)
{
    inkCanvas.InkPresenter.StrokeContainer.Clear();
    numberLabel.Text = "";
}
```

6. Launch the application

Once we build and launch the application (press **F5**), we'll be able to recognize a number drawn on the **InkCanvas**.



That's it - you've made your first Windows ML application! For more samples that demonstrate how to use Windows ML, check out our Windows-Machine-Learning repo on GitHub.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Port an existing WinML app to NuGet package (C++)

9/14/2020 • 2 minutes to read • Edit Online

In this tutorial, we'll take an existing WinML desktop application and port it to use the redistributable NuGet package.

Prerequisites

- A WinML application. If you are creating a new application, see Tutorial: Create a Windows Machine Learning Desktop application (C++)
- Windows 8.1 or higher
- Visual Studio 2019 (or Visual Studio 2017, version 15.7.4 or later)
- Download the CppWinRT NuGet package

Add the NuGet Package to your project

In the Visual Studio project for your existing application, navigate to the the Solution explorer and select **Manage NuGet Packages for Solution**. Choose the `Microsoft.AI.MachineLearning` NuGet package. Ensure you're adding to the correct project, and press **Install**.

Next, build your solution again. The C++/WinRT toolkit will parse the new headers and metadata from the `Microsoft.AI.MachineLearning` NuGet package, avoiding confusion in the next step.

Include the new header

For best practices, you should add a control flag to enable your app to swithc back and forth between using in-box Windows ML and the NuGet package.

```
#ifdef USE_WINML_NUGET
#include “winrt/Microsoft.AI.MachineLearning.h”
#endif
```

Change the namespace

Next, allow the `Windows::AI::Machinelearning` to switch over to the `Microsoft::AI::MachineLearning` namespace using a control flag. By making this change, your code will automatically use the NuGet package if applicable.

```
#ifdef USE_WINML_NUGET

Using namespace Microsoft::AI::MachineLearning

#else

Using namespace Windows::AI::MachineLearning

#endif
```

Change the Preprocessor Definitions

Now, right-click on the project in the **Solution Explorer** and select **Properties**. In the **Properties** window, choose the **Preprocessor** page. Edit the **Preprocessor Definitions**, and change it to `USE_WINML_NUGET:_DEBUG`.

Build and run

Your application now successfully uses the WinML NuGet Package.

What is a machine learning model?

8/29/2019 • 2 minutes to read • Edit Online

A machine learning model is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

Once you have trained the model, you can use it to reason over data that it hasn't seen before, and make predictions about those data. For example, let's say you want to build an application that can recognize a user's emotions based on their facial expressions. You can train a model by providing it with images of faces that are each tagged with a certain emotion, and then you can use that model in an application that can recognize any user's emotion. See the Emoji8 sample for an example of such an application.

Windows Machine Learning uses the Open Neural Network Exchange (ONNX) format for its models. You can download a pre-trained model, or you can train your own model. See Get ONNX models for Windows ML for more information.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ONNX models

1/8/2020 • 2 minutes to read • Edit Online

Windows Machine Learning supports models in the Open Neural Network Exchange (ONNX) format. ONNX is an open format for ML models, allowing you to interchange models between various ML frameworks and tools.

There are several ways in which you can obtain a model in the ONNX format, including:

- ONNX Model Zoo: Contains several pre-trained ONNX models for different types of tasks. Download a version that is supported by Windows ML and you are good to go!
- Native export from ML training frameworks: Several training frameworks support native export functionality to ONNX, like Chainer, Caffee2, and PyTorch, allowing you to save your trained model to specific versions of the ONNX format. In addition, services such as Azure Machine Learning and Azure Custom Vision also provide native ONNX export.
  - To learn how to train and export an ONNX model in the cloud using Custom Vision, check out Tutorial: Use an ONNX model from Custom Vision with Windows ML (preview).
- Convert existing models using WinMLTools: This Python package allows models to be converted from several training framework formats to ONNX. As a developer, you can specify which version of ONNX you would like to convert your model to, depending on which builds of Windows your application targets. If you are not familiar with Python, you can use Windows ML's UI-based Dashboard to easily convert your models with just a few clicks.

IMPORTANT

Not all ONNX versions are supported by Windows ML. In order to know which ONNX versions are officially supported in the Windows versions targeted by your application, please check ONNX versions and Windows builds.

Once you have an ONNX model, you'll integrate the model into your app's code, and then you'll be able use machine learning in your Windows apps and devices!

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ONNX versions and Windows builds

9/14/2020 • 2 minutes to read • Edit Online

Windows Machine Learning supports specific versions of the ONNX format in released Windows builds. In order for your model to work with Windows ML, you will need to make sure your ONNX model version is supported for the Windows release targeted by your application.

The below table summarizes all currently released versions of Windows ML and the corresponding ONNX versions supported.

WINDOWS RELEASE	ONNX VERSIONS SUPPORTED	ONNX OPSETS SUPPORTED
Windows 10, version 2004 (build 19041)	1.2.2, 1.3, and 1.4	7, 8, and 9
Windows 10, version 1909	1.2.2 and 1.3	7 and 8
Windows 10, version 1903 (build 18362)	1.2.2 and 1.3	7 and 8
Windows 10, version 1809 (build 17763)	1.2.2	7

ONNX opset 10 is supported in the NuGet package.

If you are developing using Windows Insider Flights builds, please check our release notes for the minimum and maximum supported ONNX versions in flights of the Windows 10 SDK.

ONNX opset converter

The ONNX API provides a library for converting ONNX models between different opset versions. This allows developers and data scientists to either upgrade an existing ONNX model to a newer version, or downgrade the model to an older version of the ONNX spec.

The version converter may be invoked either via C++ or Python APIs. There is also a tutorial that provides several examples on how to upgrade and downgrade an ONNX model to a new target opset.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Train a model with CNTK

5/28/2019 • 2 minutes to read • Edit Online

In this tutorial, we'll use Visual Studio Tools for AI, a development extension for building, testing, and deploying Deep Learning & AI solutions, to train a model.

We'll train the model with the Microsoft Cognitive Toolkit (CNTK) framework and the MNIST dataset, which has a training set of 60,000 examples and a test set of 10,000 examples of handwritten digits. We'll then save the model using the Open Neural Network Exchange (ONNX) format to use with Windows ML.

Prerequisites

Install Visual Studio Tools for AI

To get started, you'll need to download and install Visual Studio. Once you have Visual Studio open, activate the **Visual Studio Tools for AI** extension:

1. Click on the menu bar in Visual Studio and select "Extensions and Updates..."
2. Click on "Online" tab and select "Search Visual Studio Marketplace."
3. Search for "Visual Studio Tools for AI."
4. Click on the **Download** button.
5. After installation, restart Visual Studio.

The extension will be active once Visual Studio restarts. If you're having trouble, check out Finding Visual Studio extensions.

Download sample code

Download the Samples for AI repo on GitHub. The samples cover getting started with deep learning across TensorFlow, CNTK, Theano and more.

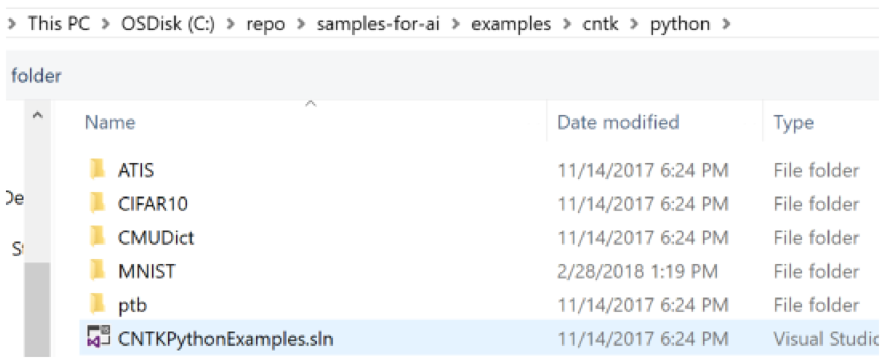
Install CNTK

Install CNTK for Python on Windows. Note that you'll also have to install Python if you haven't already.

Alternatively, to prepare your machine for deep learning model development, see Preparing your development environment for a simplified installer for installing Python, CNTK, TensorFlow, NVIDIA GPU drivers (optional) and more.

1. Open project

Launch Visual Studio and select **File > Open > Project/Solution**. From the Samples for AI repository, select the **examples\cntk\python** folder, and open the **CNTKPythonExamples.sln** file.



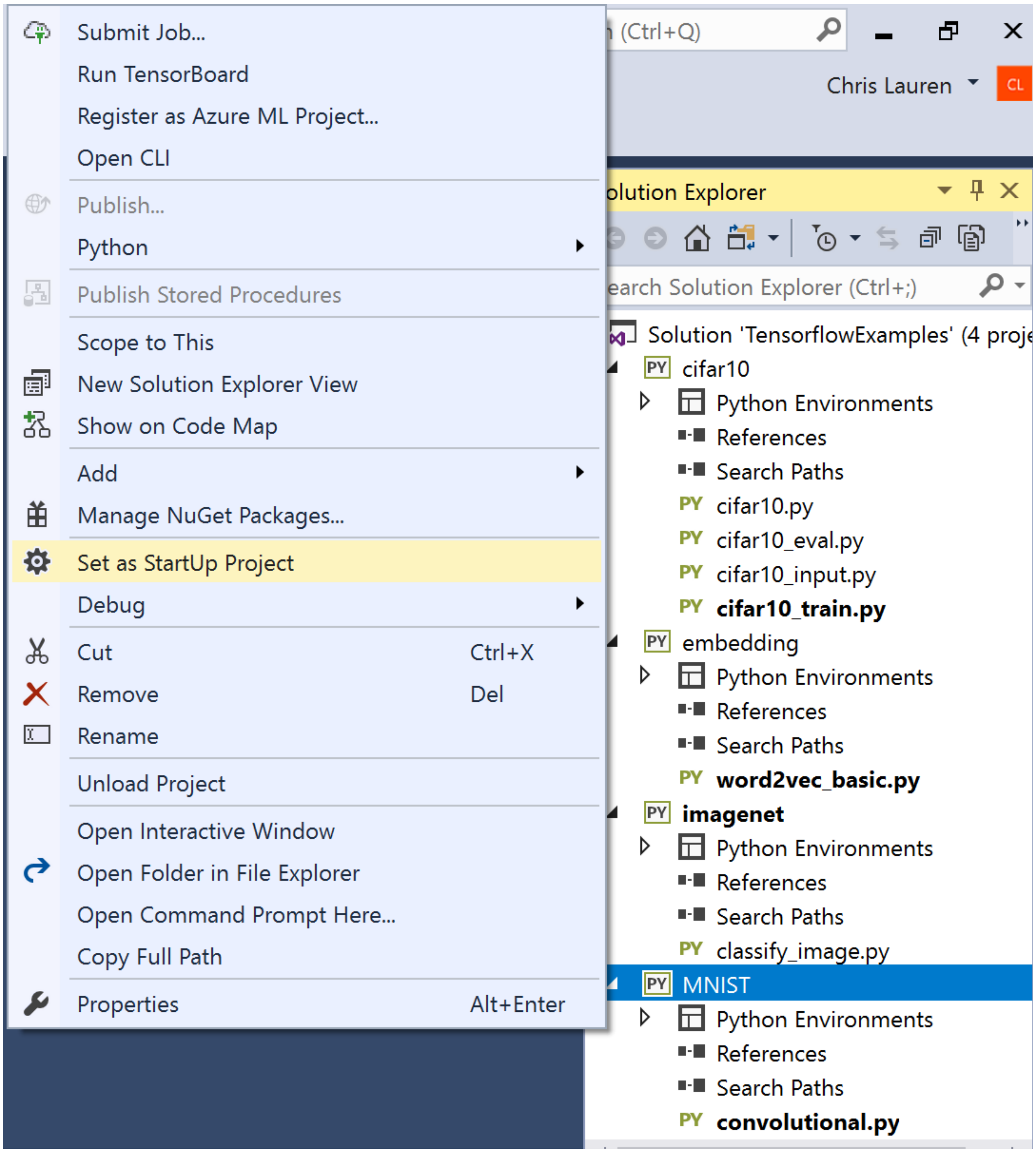
› This PC › OSDisk (C:) › repo › samples-for-ai › examples › cntk › python ›

folder

Name	Date modified	Type
ATIS	11/14/2017 6:24 PM	File folder
CIFAR10	11/14/2017 6:24 PM	File folder
CMUDict	11/14/2017 6:24 PM	File folder
MNIST	2/28/2018 1:19 PM	File folder
ptb	11/14/2017 6:24 PM	File folder
CNTKPythonExamples.sln	11/14/2017 6:24 PM	Visual Studio

2. Train the model

To set the MNIST project as the startup project, right-click on the python project and select **Set as Startup Project**.



Next, open the train_mnist_onnx.py file and **Run** the project by pressing **F5** or the green **Run** button.

3. View the model and add it to your app

Now, the trained **mnist.onnx** model file should be in the samples-for-ai/examples/cntk/python/MNIST folder.

4. Learn more

To learn how to speed up training deep learning models by using Azure GPU Virtual Machines and more, visit Artificial Intelligence at Microsoft and Microsoft Machine Learning Technologies.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Train a model with PyTorch and export to ONNX

9/14/2020 • 2 minutes to read • Edit Online

With the PyTorch framework and Azure Machine Learning, you can train a model in the cloud and download it as an ONNX file to run locally with Windows Machine Learning.

Train the model

With Azure ML, you can train a PyTorch model in the cloud, getting the benefits of rapid scale-out, deployment, and more. See Train and register PyTorch models at scale with Azure Machine Learning for more information.

Export to ONNX

Once you've trained the model, you can export it as an ONNX file so you can run it locally with Windows ML. See Export PyTorch models for Windows ML for instructions on how to natively export from PyTorch.

Integrate with Windows ML

After you've exported the model to ONNX, you're ready to integrate it into a Windows ML application. Windows ML is available in several different programming languages, so check out a tutorial in the language you're most comfortable with.

- C#: Create a Windows Machine Learning UWP application (C#)
- Python: Create a Windows Machine Learning application with Python
- C++: Create a Windows Machine Learning Desktop application (C++)

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Convert ML models to ONNX with WinMLTools

9/14/2020 • 12 minutes to read • Edit Online

WinMLTools enables you to convert machine learning models created with different training frameworks into ONNX. It is an extension of ONNXMLTools and TF2ONNX to convert models to ONNX for use with Windows ML.

WinMLTools currently supports conversion from the following frameworks:

- Apple Core ML
- Keras
- scikit-learn
- lightgbm
- xgboost
- libSVM
- TensorFlow (experimental)

To learn how to export from other ML frameworks, take a look at the ONNX tutorials on GitHub.

In this article, we demonstrate how to use WinMLTools to:

- Convert Core ML models into ONNX
- Convert scikit-learn models into ONNX
- Convert TensorFlow models into ONNX
- Apply post-training weight quantization to ONNX models
- Convert floating point models to 16-bit floating point precision models
- Create custom ONNX operators

NOTE

The latest version of WinMLTools supports conversion to ONNX versions 1.2.2, 1.3, and 1.4, as specified respectively by ONNX opsets 7 and 8, and 9. Previous versions of the tool do not have support for ONNX 1.4.

Install WinMLTools

WinMLTools is a Python package (**winmltools**) that supports Python versions 2.7 and 3.6. If you are working on a data science project, we recommend installing a scientific Python distribution such as Anaconda.

NOTE

WinMLTools does not currently support Python 3.7.

WinMLTools follows the standard Python package installation process. From your Python environment, run:

```
pip install winmltools
```

WinMLTools has the following dependencies:

- **numpy** v1.10.0+
- **protobuf** v.3.6.0+

- **onnx** v1.3.0+
- **onnxmltools** v1.3.0+
- **tf2onnx** v0.3.2+

To update the dependent packages, run the `pip` command with the `-U` argument.

```
pip install -U winmltools
```

For different converters, you will have to install different packages.

For **libsvm**, you can download **libsvm wheel** from various web sources. One example can be found at the University of California, Irvine's website.

For **coremltools**, currently Apple does not distribute Core ML packaging on Windows. You can install from source:

```
pip install git+https://github.com/apple/coremltools
```

Follow onnxmltools on GitHub for further information on **onnxmltools** dependencies.

Additional details on how to use WinMLTools can be found on the package-specific documentation with the `help` function.

```
help(winmltools)
```

Convert Core ML models

Here, we assume that the path of an example Core ML model file is *example.mlmodel*.

```
from coremltools.models.utils import load_spec
# Load model file
model_coreml = load_spec('example.mlmodel')
from winmltools import convert_coreml
# Convert it!
# The automatic code generator (mlgen) uses the name parameter to generate class names.
model_onnx = convert_coreml(model_coreml, 7, name='ExampleModel')
```

NOTE

The second parameter in the call to convert_coreml() is the target_opset, and it refers to the version number of the operators in the default namespace `ai.onnx`. See more details on these operators here. This parameter is only available on the latest version of WinMLTools, enabling developers to target different ONNX versions (currently versions 1.2.2 and 1.3 are supported). To convert models to run with the Windows 10 October 2018 update, use `target_opset` 7 (ONNX v1.2.2). For Windows 10 builds greater than 17763, WinML accepts models with `target_opset` 7 and 8 (ONNX v.1.3). The Release Notes section also contains the min and max ONNX versions supported by WinML in different builds.

`model_onnx` is an ONNX ModelProto object. We can save it in two different formats.

```
from winmltools.utils import save_model
# Save the produced ONNX model in binary format
save_model(model_onnx, 'example.onnx')
# Save the produced ONNX model in text format
from winmltools.utils import save_text
save_text(model_onnx, 'example.txt')
```

NOTE

Core MLTools is a Python package provided by Apple, but is not available on Windows. If you need to install the package on Windows, install the package directly from the repo:

```
pip install git+https://github.com/apple/coremltools
```

Convert Core ML models with image inputs or outputs

Because of the lack of image types in ONNX, converting Core ML image models (that is, models using images as inputs or outputs) requires some pre-processing and post-processing steps.

The objective of pre-processing is to make sure the input image is properly formatted as an ONNX tensor. Assume *X* is an image input with shape [C, H, W] in Core ML. In ONNX, the variable *X* would be a floating-point tensor with the same shape and *X*[0, :, :]/*X*[1, :, :]/*X*[2, :, :] stores the image's red/green/blue channel. For grayscale images in Core ML, their format is [1, H, W]-tensors in ONNX because we only have one channel.

If the original Core ML model outputs an image, manually convert ONNX's floating-point output tensors back into images. There are two basic steps. The first step is to truncate values greater than 255 to 255 and change all negative values to 0. The second step is to round all pixel values to integers (by adding 0.5 and then truncating the decimals). The output channel order (for example, RGB or BGR) is indicated in the Core ML model. To generate proper image output, we may need to transpose or shuffle to recover the desired format.

Here we consider a Core ML model, FNS-Candy, downloaded from GitHub, as a concrete conversion example to demonstrate the difference between ONNX and Core ML formats. Note that all the subsequent commands are executed in a Python environment.

First, we load the Core ML model and examine its input and output formats.

```
from coremltools.models.utils import load_spec
model_coreml = load_spec('FNS-Candy.mlmodel')
model_coreml.description # Print the content of Core ML model description
```

Screen output:

```
...
input {
    ...
      imageType {
      width: 720
      height: 720
      colorSpace: BGR
    ...
}
...
output {
    ...
      imageType {
      width: 720
      height: 720
      colorSpace: BGR
    ...
}
...
```

In this case, both the input and output are 720x720 BGR-images. Our next step is to convert the Core ML model with WinMLTools.

```
# The automatic code generator (mlgen) uses the name parameter to generate class names.
from winmltools import convert_coreml
model_onnx = convert_coreml(model_coreml, 7, name='FNSCandy')
```

An alternative method to view the model input and output formats in ONNX is to use the following command:

```
model_onnx.graph.input # Print out the ONNX input tensor's information
```

Screen output:

```
...
  tensor_type {
    elem_type: FLOAT
    shape {
      dim {
        dim_param: "None"
      }
      dim {
        dim_value: 3
      }
      dim {
        dim_value: 720
      }
      dim {
        dim_value: 720
      }
    }
  }
...
```

The produced input (denoted by X) in ONNX is a 4-D tensor. The last 3 axes are C-, H-, and W-axes, respectively. The first dimension is "None" which means that this ONNX model can be applied to any number of images. To apply this model to process a batch of 2 images, the first image corresponds to X[0, :, :, :] while X[1, :, :, :] corresponds to the second image. The blue/green/red channels of the first image are X[0, 0, :, :]/X[0, 1, :, :]/X[0, 2, :, :], and similar for the second image.

```
model_onnx.graph.output # Print out the ONNX output tensor's information
```

Screen output:

```
...
  tensor_type {
    elem_type: FLOAT
    shape {
      dim {
        dim_param: "None"
      }
      dim {
        dim_value: 3
      }
      dim {
        dim_value: 720
      }
      dim {
        dim_value: 720
      }
    }
  }
...
```

As you can see, the produced format is identical to the original model input format. However, in this case, it's not an image because the pixel values are integers, not floating-point numbers. To convert back to an image, truncate values greater than 255 to 255, change negative values to 0, and round all decimals to integers.

Convert scikit-learn models

The following code snippet trains a linear support vector machine in scikit-learn and converts the model into ONNX.

```
# First, we create a toy data set.
# The training matrix X contains three examples, with two features each.
# The label vector, y, stores the labels of all examples.
X = [[0.5, 1.], [-1., -1.5], [0., -2.]]
y = [1, -1, -1]

# Then, we create a linear classifier and train it.
from sklearn.svm import LinearSVC
linear_svc = LinearSVC()
linear_svc.fit(X, y)

# To convert scikit-learn models, we need to specify the input feature's name and type for our converter.
# The following line means we have a 2-D float feature vector, and its name is "input" in ONNX.
# The automatic code generator (mlgen) uses the name parameter to generate class names.
from winmltools import convert_sklearn
from winmltools.convert.common.data_types import FloatTensorType
linear_svc_onnx = convert_sklearn(linear_svc, 7, name='LinearSVC',
                                  initial_types=[('input', FloatTensorType([1, 2]))])

# Now, we save the ONNX model into binary format.
from winmltools.utils import save_model
save_model(linear_svc_onnx, 'linear_svc.onnx')

# If you'd like to load an ONNX binary file, our tool can also help.
from winmltools.utils import load_model
linear_svc_onnx = load_model('linear_svc.onnx')

# To see the produced ONNX model, we can print its contents or save it in text format.
print(linear_svc_onnx)
from winmltools.utils import save_text
save_text(linear_svc_onnx, 'linear_svc.txt')

# The conversion of linear regression is very similar. See the example below.
from sklearn.svm import LinearSVR
linear_svr = LinearSVR()
linear_svr.fit(X, y)
linear_svr_onnx = convert_sklearn(linear_svr, 7, name='LinearSVR',
                                  initial_types=[('input', FloatTensorType([1, 2]))])
```

As before `convert_sklearn` takes a scikit-learn model as its first argument, and the `target_opset` for the second argument. Users can replace `LinearSVC` with other scikit-learn models such as `RandomForestClassifier`. Please note that mlgen uses the `name` parameter to generate class names and variables. If `name` is not provided, then a GUID is generated, which will not comply with variable naming conventions for languages like C++/C#.

Convert scikit-learn pipelines

Next, we show how scikit-learn pipelines can be converted into ONNX.

```
# First, we create a toy data set.
# Notice that the first example's last feature value, 300, is large.
X = [[0.5, 1., 300.], [-1., -1.5, -4.], [0., -2., -1.]]
y = [1, -1, -1]

# Then, we declare a linear classifier.
from sklearn.svm import LinearSVC
linear_svc = LinearSVC()

# One common trick to improve a linear model's performance is to normalize the input data.
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# Here, we compose our scikit-learn pipeline.
# First, we apply our normalization.
# Then we feed the normalized data into the linear model.
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(normalizer, linear_svc)
pipeline.fit(X, y)

# Now, we convert the scikit-learn pipeline into ONNX format.
# Compared to the previous example, notice that the specified feature dimension becomes 3.
# The automatic code generator (mlgen) uses the name parameter to generate class names.
from winmltools import convert_sklearn
from winmltools.convert.common.data_types import FloatTensorType, Int64TensorType
pipeline_onnx = convert_sklearn(linear_svc, name='NormalizerLinearSVC',
                                input_features=[('input', FloatTensorType([1, 3]))])

# We can print the fresh ONNX model.
print(pipeline_onnx)

# We can also save the ONNX model into a binary file for later use.
from winmltools.utils import save_model
save_model(pipeline_onnx, 'pipeline.onnx')

# Our conversion framework provides limited support of heterogeneous feature values.
# We cannot have numerical types and string types in one feature vector.
# Let's assume that the first two features are floats and the last feature is integers (encoded a categorical
attribute).
X_heter = [[0.5, 1., 300], [-1., -1.5, 400], [0., -2., 100]]

# One popular way to represent categorical is one-hot encoding.
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder(categorical_features=[2])

# Let's initialize a classifier.
# It will be right after the one-hot encoder in our pipeline.
linear_svc = LinearSVC()

# Then, we form a two-stage pipeline.
another_pipeline = make_pipeline(one_hot_encoder, linear_svc)
another_pipeline.fit(X_heter, y)

# Now, we convert, save, and load the converted model.
# For heterogeneous feature vectors, we need to separately specify their types for
# all homogeneous segments.
# The automatic code generator (mlgen) uses the name parameter to generate class names.
another_pipeline_onnx = convert_sklearn(another_pipeline, name='OneHotLinearSVC',
                                        input_features=[('input', FloatTensorType([1, 2])),
                                        ('another_input', Int64TensorType([1, 1]))])
save_model(another_pipeline_onnx, 'another_pipeline.onnx')
from winmltools.utils import load_model
loaded_onnx_model = load_model('another_pipeline.onnx')

# Of course, we can print the ONNX model to see if anything went wrong.
print(another_pipeline_onnx)
```

Convert TensorFlow models

The following code is an example of how to convert a model from a frozen TensorFlow model. To get the possible output names of a TensorFlow model, you can use the summarize_graph tool.

```
import winmltools
import tensorflow

filename = 'frozen-model.pb'
output_names = ['output:0']

graph_def = graph_pb2.GraphDef()
with open(filename, 'rb') as file:
  graph_def.ParseFromString(file.read())
g = tf.import_graph_def(graph_def, name='')

with tf.Session(graph=g) as sess:
  converted_model = winmltools.convert_tensorflow(sess.graph, 7, output_names=['output:0'])
  winmltools.save_model(converted_model)
```

The WinMLTools converter uses `tf2onnx.tfonnx.process_tf_graph` in TF2ONNX.

Convert to floating point 16

WinMLTools supports the conversion of models represented in floating point 32 into a floating point 16 representation (IEEE 754 half), effectively compressing the model by reducing its size in half.

NOTE

Converting your model to floating point 16 could result in a loss of accuracy. Make sure you verify the model's accuracy before deploying into your application.

Below is a full example if you want to convert directly from an ONNX binary file.

```
from winmltools.utils import convert_float_to_float16
from winmltools.utils import load_model, save_model
onnx_model = load_model('model.onnx')
new_onnx_model = convert_float_to_float16(onnx_model)
save_model(new_onnx_model, 'model_fp16.onnx')
```

With `help(winmltools.utils.convert_float_to_float16)`, you can find more details about this tool. The floating point 16 in WinMLTools currently only complies with IEEE 754 floating point standard (2008).

Post-training weight quantization

WinMLTools also supports the compression of models represented in floating point 32 into 8-bit integer representations. This could yield a disk footprint reduction of up to 75% depending on the model. This reduction is done via a technique called post-training weight quantization, where the model is analyzed and the stored tensor weights are reduced from 32-bit floating point data into 8-bit data.

NOTE

Post-training weight quantization may result in loss of accuracy in the resulting model. Make sure you verify the model's accuracy before deploying into your application.

The following is a complete example that demonstrates how to convert directly from an ONNX binary file.

```
import winmltools

model = winmltools.load_model('model.onnx')
packed_model = winmltools.quantize(model, per_channel=True, nbits=8, use_dequantize_linear=True)
winmltools.save_model(packed_model, 'quantized.onnx')
```

Here is some information about the input parameters to `quantize`:

- `per_channel`: If set to `True`, this will linearly quantize each channel for each initialized tensor in [n,c,h,w] format. By default, this parameter is set to `True`.
- `nbits`: The number of bits to represent quantized values. Currently only 8 bits is supported.
- `use_dequantize_linear`: If set to `True`, this will linearly dequantize each channel in initialized tensors for Conv operators in [n,c,h,w] format. By default, this is set to `True`.

Create custom ONNX operators

When converting from a Keras or a Core ML model, you can write a custom operator function to embed custom operators into the ONNX graph. During the conversion, the converter invokes your function to translate the Keras layer or the Core ML LayerParameter to an ONNX operator, and then it connects the operator node into the whole graph.

1. Create the custom function for the ONNX sub-graph building.
2. Call `winmltools.convert_keras` or `winmltools.convert_coreml` with the map of the custom layer name to the custom function.
3. If applicable, implement the custom layer for the inference runtime.

The following example shows how it works in Keras.

```
# Define the activation layer.
class ParametricSoftplus(keras.layers.Layer):
    def __init__(self, alpha, beta, **kwargs):
    ...
    ...
    ...

# Create the convert function.
def convert_userPSoftplusLayer(scope, operator, container):
      return container.add_node('ParametricSoftplus', operator.input_full_names, operator.output_full_names,
        op_version=1, alpha=operator.original_operator.alpha, beta=operator.original_operator.beta)

winmltools.convert_keras(keras_model, 7,
    custom_conversion_functions={ParametricSoftplus: convert_userPSoftplusLayer })
```

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Integrate a model into your app with Windows ML

5/19/2020 • 2 minutes to read • Edit Online

In this guide, we'll cover how to use the Windows ML APIs to integrate a model into your Windows app. Alternatively, if you'd like to use Windows ML's automatic code generator, check out mlgen.

Important APIs: Windows.AI.MachineLearning

We'll go over the basic building blocks of Windows ML, which are:

- Models
- Sessions
- Devices
- Bindings

You'll use these to load, bind, and evaluate your models with Windows ML.

We also recommend taking a look at our sample apps on GitHub to see end-to-end Windows ML code examples.

The following video shows these APIs in action in a short demo.

Using WinML APIs in C++

While the WinML APIs are available in both C++/CX and C++/WinRT, we recommend using the C++/WinRT version, as it allows for more natural C++ coding and is where most development efforts will be focused going forward. You can follow the instructions below that pertain to your particular situation to use the C++/WinRT APIs:

- If you are targeting Windows 1803 or earlier, see Tutorial: Port an Existing WinML App to NuGet Package.
- If you are creating a new C++ application, see Tutorial: Create a Windows Machine Learning Desktop application (C++) and follow the steps up to **Load the model**.
- If you have an existing C++ application (which is not already set up for C++/WinRT), follow these steps to set up your application for C++/WinRT:
  1. Make sure you have the latest version of Visual Studio 2019 installed (any edition).
  2. Make sure you have the SDK for Windows 10, version 1803 or later.
  3. Download and install the C++/WinRT Visual Studio Extension (VSIX) from the Visual Studio Marketplace.
  4. Add the `<CppWinRTEnabled>true</CppWinRTEnabled>` property to the project's .vcxproj file:

     ```
     <Project ...>
         <PropertyGroup Label="Globals">
             <CppWinRTEnabled>true</CppWinRTEnabled>
     ...
     ```

  5. C++/WinRT requires features from the C++17 standard, so in your project properties, set **C/C++ > Language > C++ Language Standard > ISO C++17 Standard (/std:c++17)**.
  6. Set **Conformance mode: Yes (/permissive-)** in your project properties.
  7. Another project property to be aware of is **C/C++ > General > Treat Warnings As Errors**. Set this to **Yes (/WX)** or **No (/WX-)** to taste. Sometimes, source files generated by the **cppwinrt.exe** tool generate

warnings until you add your implementation to them.

8. The VSIX also gives you Visual Studio native debug visualization (natvis) of C++/WinRT projected types, providing an experience similar to C# debugging. Natvis is automatic for debug builds. You can opt into its release builds by defining the symbol **WINRT_NATVIS**.
9. Your project should now be setup for C++/WinRT. See C++/WinRT for more information.

Related topics

- mlgen
- Performance and memory
- API reference
- Code examples

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Load a model

8/29/2019 • 2 minutes to read • Edit Online

IMPORTANT

Windows Machine Learning requires ONNX models, version 1.2 or higher.

Once you get a trained ONNX model, you'll distribute the .onnx model file(s) with your app. You can include the .onnx file(s) in your APPX package, or, for desktop apps, they can be anywhere your app can access on the hard drive.

There are several ways to load a model using static methods on the LearningModel class:

- LearningModel.LoadFromStreamAsync
- LearningModel.LoadFromStream
- LearningModel.LoadFromStorageFileAsync
- LearningModel.LoadFromFilePath

The **LoadFromStream*** methods allow applications to have more control over where the model comes from. For example, an app could choose to have the model encrypted on disk and decrypt it only in memory prior to calling one of the **LoadFromStream*** methods. Other options include loading the model stream from a network share or other media.

TIP

Loading a model can take some time, so take care not to call a **Load*** method from your UI thread.

The following example shows how you can load a model into your application:

```
private async LearningModel LoadModelAsync(string modelPath)
{
    // Load and create the model
    var modelFile = await StorageFile.GetFileFromApplicationUriAsync(
        new Uri(modelPath));

    LearningModel model =
        await LearningModel.LoadFromStorageFileAsync(modelFile);

    return model;
}
```

See also

- Next: Create a session

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Create a session

3/24/2020 • 2 minutes to read • Edit Online

Once you load a LearningModel, you create a LearningModelSession, which binds the model to a device that runs and evaluates the model.

Choose a device

You can select a device when you create a session. You choose a device of type LearningModelDeviceKind:

- **Default**
  - Let the system decide which device to use. Currently, the default device is the CPU.
- **CPU**
  - Use the CPU, even if other devices are available.
- **DirectX**
  - Use a DirectX hardware acceleration device, specifically the first adapter enumerated by IDXGIFactory1::EnumAdapters1.
- **DirectXHighPerformance**
  - Same as **DirectX**, but will use DXGI_GPU_PREFERENCE_HIGH_PERFORMANCE when enumerating adapters.
- **DirectXMinPower**
  - Same as **DirectX**, but will use DXGI_GPU_PREFERENCE_MINIMUM_POWER when enumerating adapters.

If you don't specify a device, the system uses **Default**. We recommend using **Default** to get the flexibility of allowing the system to choose for you in the future.

The following video goes into more detail about each of the device kinds.

Advanced device creation

Windows AI supports using a device that the caller has already created. There are several options and considerations when doing this:

- CreateFromDirect3D11Device. Use this when you already have an existing IDirect3DDevice. Windows AI will use that same adapter to create a d3d12 device for its ML workloads. This is useful when you have a camera that is using a d3d11 device for VideoFrames and you want to use that same device for your LearningModelSession. In many cases it can avoid a memory copy. Note: VideoFrame tensorization is the only d3d11 workload Windows AI has. If you are not using that feature there is no advantage to sharing or creating a d3d11 device.
- CreateFromD3D12CommandQueue (native). Use this when you have a d3d12 device that you want to reuse. Windows AI will use that command queue for its ML workloads. It will also create an d3d11 device using D3D11On12CreateDevice. This is done only when needed and will be used for all d3d11 workloads such as VideoFrame tensorization. You can access this new device via the LearningModelDevice.Direct3D11Device property.

Example

The following example shows how to create a session from a model and a device:

```
private void CreateSession(LearningModel model, LearningModelDeviceKind kind)
{
    // Create the evaluation session with the model and device
    LearningModelSession session =
        new LearningModelSession(model, new LearningModelDevice(kind));
}
```

See also

- Previous: Load a model
- Next: Bind a model

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Bind a model

12/13/2019 • 5 minutes to read • Edit Online

A machine learning model has input and output features, which pass information into and out of the model.

After you load your model as a LearningModel, you can use LearningModel.InputFeatures and LearningModel.OutputFeatures to get ILearningModelFeatureDescriptor objects. These list the model's expected input and output feature types.

You use a LearningModelBinding to bind values to a feature, referencing the **ILearningModelFeatureDescriptor** by its Name property.

The following video gives a brief overview of binding features of machine learning models.

Types of features

Windows ML supports all ONNX feature types, which are enumerated in LearningModelFeatureKind. These are mapped to different feature descriptor classes:

- **Tensor**: TensorFeatureDescriptor
- **Sequence**: SequenceFeatureDescriptor
- **Map**: MapFeatureDescriptor
- **Image**: ImageFeatureDescriptor

Tensors

Tensors are multi-dimensional arrays, and the most common tensor is a tensor of 32-bit floats. The dimensions of tensors are row-major, with tightly packed contiguous data representing each dimension. The tensor's total size is the product of the sizes of each dimension.

Sequences

Sequences are vectors of values. A common use of sequence types is a vector of float probabilities, which some classification models return to indicate the accuracy rating for each prediction.

Maps

Maps are key/value pairs of information. Classification models commonly return a string/float map that describes the float probability for each labeled classification name. For example, the output of a model that tries to predict the breed of dog in a picture might be `["Boston terrier", 90.0], ["Golden retriever", 7.4], ["Poodle", 2.6]`.

Scalars

Most maps and sequences will have values that are scalars. These show up where **TensorFeatureDescriptor.Shape.Size** is zero (0). In this case, the map or sequence will be of the scalar type. The most common is `float`. For example, a string to float map would be:

```
MapFeatureDescriptor.KeyKind == TensorKind.String
MapFeatureDescriptor.ValueDescriptor.Kind == LearningModelFeatureKind.Tensor
MapFeatureDescriptor.ValueDescriptor.as<TensorFeatureDescriptor>().Shape.Size == 0
```

The actual map feature value will be an `IMap<string, float>`.

Sequence of maps

A sequence of maps is just a vector of key/value pairs. For example, a sequence of string-to-float maps would be of type `IVector<IMap<string, float>>`. The above output of dog breed predictions `["Boston terrier", 90.0], ["Golden retriever", 7.4], ["Poodle", 2.6]` is an example of a sequence of maps.

Images

When working with images, you'll need to be aware of image formats and tensorization.

Image formats

Models are trained with image training data, and the weights are saved and tailored for that training set. When you pass an image input into the model, its format must match the training images' format.

In many cases, the model describes the expected image format; ONNX models can use metadata to describe expected image formats.

Most models use the following formats, but this is not universal to all models:

- **Image.BitmapPixelFormat**: Bgr8
- **Image.ColorSpaceGamma**: SRGB
- **Image.NominalPixelRange**: NominalRange_0_255

Tensorization

Images are represented in Windows ML in a tensor format. Tensorization is the process of converting an image into a tensor and happens during bind.

Windows ML converts images into 4-dimensional tensors of 32-bit floats in the "NCHW tensor format":

- **N**: Batch size (or number of images). Windows ML currently supports a batch size N of 1.
- **C**: Channel count (1 for Gray8, 3 for Bgr8).
- **H**: Height.
- **W**: Width.

Each pixel of the image is an 8-bit color number that is stored in the range of 0-255 and packed into a 32-bit float.

How to pass images into the model

There are two ways you can pass images into models:

- ImageFeatureValue

  We recommend using **ImageFeatureValue** to bind images as inputs and outputs, as it takes care of both conversion and tensorization, so the images match the model's required image format. The currently supported model format types are **Gray8**, **Rgb8**, and **Bgr8**, and the currently supported pixel range is 0-255.

  You can create an **ImageFeatureValue** using the static method ImageFeatureValue.CreateFromVideoFrame.

  To find out what format the model needs, WinML uses the following logic and precedence order:

  1. Bind(String, Object, IPropertySet) will override all image settings.
  2. Model metadata will then be checked and used if available.
  3. If no model metadata is provided, and no caller supplied properties, the runtime will attempt to make a best match.
     - If the tensor looks like NCHW (4-dimensional float32, N==1), the runtime will assume either **Gray8** (C==1) or **Bgr8** (C==3) depending on the channel count.
     - NominalRange_0_255 will be assumed
     - SRGB will be assumed

There are several optional properties that you can pass into Bind(String, Object, IPropertySet):

  - BitmapBounds: If specified, these are the cropping boundaries to apply prior to sending the image to the model.
  - BitmapPixelFormat: If specified, this is the pixel format that will be used as the model pixel format during image conversion.

  For image shapes, the model can specify either a specific shape that it takes (for example, SqueezeNet takes 224,224), or the model can specify free dimensions for any shape image (many StyleTransfer-type models can take variable sized images). The caller can use **BitmapBounds** to choose which section of the image they would like to use. If not specified, the runtime will scale the image to the model size (respecting aspect ratio) and then center crop.

- TensorFloat

  If Windows ML does not support your model's color format or pixel range, then you can implement conversions and tensorization. You'll create an NCHW four-dimensional tensor for 32-bit floats for your input value. See the Custom Tensorization Sample for an example of how to do this.

  When this method is used, any image metadata on the model is ignored.

Example

The following example shows how to bind to a model's input. In this case, we create a binding from *session*, create an **ImageFeatureValue** from *inputFrame*, and bind the image to the model's input, *inputName*.

```
private void BindModel(
    LearningModelSession session,
    VideoFrame inputFrame,
    string inputName)
{
    // Create a binding object from the session
    LearningModelBinding binding = new LearningModelBinding(session);

    // Create an image tensor from a video frame
    ImageFeatureValue image =
        ImageFeatureValue.CreateFromVideoFrame(inputFrame);

    // Bind the image to the input
    binding.Bind(inputName, image);
}
```

See also

- Previous: Create a session
- Next: Evaluate the model inputs

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Evaluate the model inputs

8/29/2019 • 2 minutes to read • Edit Online

Once you have bound values to a model's inputs and outputs, you are ready to evaluate the model's inputs and get its predictions.

To run the model, you call any of the **Evaluate*** methods on your LearningModelSession. You can use the LearningModelEvaluationResult to look at the output features.

Example

In the following example, we run an evaluation on the session, passing in the binding and a unique correlation ID. Then we parse the output as a list of probabilities, match it to a list of labels for the different things our model can recognize, and write the results to the console:

```
// How many times an evaluation has been run
private int runCount = 0;

private void EvaluateModel(
    LearningModelSession session,
    LearningModelBinding binding,
    string outputName,
    List<string> labels)
{
    // Process the frame with the model
    var results =
        await session.EvaluateAsync(binding, $"Run {++runCount}");

    // Retrieve the results of evaluation
    var resultTensor = results.Outputs[outputName] as TensorFloat;
    var resultVector = resultTensor.GetAsVectorView();

    // Find the top 3 probabilities
    List<(int index, float probability)> indexedResults = new List<(int, float)>();

    for (int i = 0; i < resultVector.Count; i++)
    {
        indexedResults.Add((index: i, probability: resultVector.ElementAt(i)));
    }

    // Sort the results in order of highest probability
    indexedResults.Sort((a, b) =>
    {
        if (a.probability < b.probability)
        {
            return 1;
        }
        else if (a.probability > b.probability)
        {
            return -1;
        }
        else
        {
            return 0;
        }
    });

    // Display the results
    for (int i = 0; i < 3; i++)
    {
        Debug.WriteLine(
            $"\"{labels[indexedResults[i].index]}\" with confidence of {indexedResults[i].probability}");
    }
}
```

Device removal

If the device becomes unavailable, or if you'd like to use a different device, you must close the session and create a new session.

In some cases, graphics devices might need to be unloaded and reloaded, as explained in the DirectX documentation.

When using Windows ML, you'll need to detect this case and close the session. To recover from a device removal or re-initialization, you'll create a new session, which triggers the device selection logic to run again.

The most common case where you will see this error is during LearningModelSession.Evaluate. In the case of device removal or reset, LearningModelEvaluationResult.ErrorStatus will be DXGI_ERROR_DEVICE_REMOVED or DXGI_ERROR_DEVICE_RESET.

See also

- Previous: Bind a model

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Automatic code generation with mlgen

5/28/2019 • 2 minutes to read • Edit Online

Windows Machine Learning's code generator **mlgen** creates an interface (C#, C++/WinRT, and C++/CX) with wrapper classes that call the Windows ML API for you, allowing you to easily load, bind, and evaluate a model in your project.

mlgen is provided as a Visual Studio extension for developers creating WinML applications in VS 2017 or later.

In Windows 10, version 1903 and later, **mlgen** is no longer included in the Windows 10 SDK, so you must download and install the extension. There is one for Visual Studio 2017 and one for Visual Studio 2019.

Once you have **mlgen** installed, inside your Visual Studio project, add your ONNX file to your project's **Assets** folder, and VS will generate Windows ML wrapper classes in a new interface file. You can use these classes and methods to integrate your model into your application. See Tutorial: Create a Windows Machine Learning UWP application (C#) for a tutorial that integrates a model using this method.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Windows ML performance and memory

3/24/2020 • 3 minutes to read • Edit Online

In this article, we'll cover how to manage your application's performance when using Windows Machine Learning.

Threading and concurrency

Every object exposed from the runtime is *agile*, meaning that they can be accessed from any thread. See Agile objects in C++/WinRT for more on agile.

One key object you will work with is the LearningModelSession. This object is always safe to call from any thread.

- **For GPU sessions**: The object will lock and synchronize concurrent calls. If you require concurrency you need to create multiple sessions in order to achieve it.
- **For CPU sessions**: The object will not lock and will allow concurrent calls on a single session. You must take care to manage your own state, buffers, and binding objects.

You should take care and measure your goal for your scenario. Modern GPU architectures work differently than CPUs. For example, if low latency is your goal you might want to manage how you schedule work across your CPU and GPU engines using pipelining, not concurrency. This article on multi-engine synchronization is a great place to get started. If throughput is your goal (like processing as many images at a time as possible) you often want to use multiple threads and concurrency in order to saturate the CPU.

When it comes to threading and concurrency you want to run experiments and measure timings. Your performance will change significantly based on your goals and scenario.

Memory utilization

Each instance of LearningModel and LearningModelSession has a copy of the model in memory. If you're working with small models, you might not be concerned, but if you're working with very large models, this becomes important.

To release the memory, call **Dispose** on either the model or session. Don't just delete them, as some languages perform lazy garbage collection.

LearningModel keeps a copy in memory to enable new session creation. When you dispose the **LearningModel**, all existing sessions will continue to work. However, you will no longer be able to create new sessions with that **LearningModel** instance. For large models, you can create a model and session, and then dispose the model. By using a single session for all calls to Evaluate, you'll have a single copy of the large model in memory.

Float16 support

For better performance and reduced model footprint, you can use WinMLTools to convert your model to float16.

Once converted, all weights and inputs are float16. Here's how you can work with float16 inputs and outputs:

- ImageFeatureValue
  - Recommended usage.
  - Converts colors and tensorizes into float16.
  - Supports bgr8 and 8-bit image formats, which can be converted safely into float16 without data loss.
- TensorFloat

  - Advanced path.
  - Float32 cast into float16.
  - For images, this is safe to cast, as bgr8 is small and fits.
  - For non-images, Bind will fail, and you will need to pass in a TensorFloat16Bit instead.
- TensorFloat16Bit
  - Advanced path.
  - You need to convert to float16 and pass the inputs as float32, which will be cast down into float16.

NOTE

Most of the time, the operator is still performing 32-bit math. There is less risk for overflow, and the result is truncated to float16. However, if the hardware advertises float16 support, then the runtime will take advantage of it.

Pre-processing input data

WinML performs some pre-processing steps under the covers to make processing input data simpler and more efficient. For example, given input images could be in various color formats and shapes, and may be different than what the model expects. WinML performs conversions on the images to match them up, reducing the load on the developer.

WinML also leverages the entire hardware stack (CPU, GPU, and so on) to provide the most efficient conversions for a particular device and scenario.

However, in certain cases you may want to manually tensorize your input data due to some specific requirements you have. For example, maybe you don't want to use VideoFrame for your images, or you want to normalize the pixel values from range 0-255 to range 0-1. In these cases, you can perform your own custom tensorization on the data. See the Custom Tensorization Sample for an example of this.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Executing multiple ML models in a chain

5/28/2019 • 3 minutes to read • Edit Online

Windows ML supports high-performance load and execution of model chains by carefully optimizing its GPU path. Model chains are defined by two or more models that execute sequentially, where the outputs of one model become the inputs to the next model down the chain.

In order to explain how to efficiently chain models with Windows ML, let's use a FNS-Candy Style Transfer ONNX model as a toy example. You can find this type of model in the FNS-Candy Style Transfer sample folder in our GitHub.

Let's say we want to execute a chain that is composed of two instances of the same FNS-Candy model, here called **mosaic.onnx**. The application code would pass an image to the first model in the chain, let it compute the outputs, and then pass that transformed image to another instance of FNS-Candy, producing a final image.

The following steps illustrate how to accomplish that using Windows ML.

NOTE

In a real word scenario you most likely would use two different models, but this should be enough to illustrate the concepts.

1. First, let's load the **mosaic.onnx** model so that we can use it.

```
std::wstring filePath = L"path\\to\\mosaic.onnx";
LearningModel model = LearningModel::LoadFromFilePath(filePath);
```

```
string filePath = "path\\to\\mosaic.onnx";
LearningModel model = LearningModel.LoadFromFilePath(filePath);
```

2. Then, let's create two identical sessions on the device's default GPU using the same model as input parameter.

```
LearningModelSession session1(model, LearningModelDevice(LearningModelDeviceKind::DirectX));
LearningModelSession session2(model, LearningModelDevice(LearningModelDeviceKind::DirectX));
```

```
LearningModelSession session1 =
  new LearningModelSession(model, new LearningModelDevice(LearningModelDeviceKind.DirectX));
LearningModelSession session2 =
  new LearningModelSession(model, new LearningModelDevice(LearningModelDeviceKind.DirectX));
```

NOTE

In order to reap the performance benefits of chaining, you need to create identical GPU sessions for all of your models. Not doing so would result in additional data movement out of the GPU and into the CPU, which would reduce performance.

3. The following lines of code will create bindings for each session:

```
LearningModelBinding binding1(session1);
LearningModelBinding binding2(session2);
```

```
LearningModelBinding binding1 = new LearningModelBinding(session1);
LearningModelBinding binding2 = new LearningModelBinding(session2);
```

4. Next, we will bind an input for our first model. We will pass in an image that is located in the same path as our model. In this example the image is called "fish_720.png".

```
//get the input descriptor
ILearningModelFeatureDescriptor input = model.InputFeatures().GetAt(0);
//load a SoftwareBitmap
hstring imagePath = L"path\\to\\fish_720.png";

// Get the image and bind it to the model's input
try
{
  StorageFile file = StorageFile::GetFileFromPathAsync(imagePath).get();
  IRandomAccessStream stream = file.OpenAsync(FileAccessMode::Read).get();
  BitmapDecoder decoder = BitmapDecoder::CreateAsync(stream).get();
  SoftwareBitmap softwareBitmap = decoder.GetSoftwareBitmapAsync().get();
  VideoFrame videoFrame = VideoFrame::CreateWithSoftwareBitmap(softwareBitmap);
  ImageFeatureValue image = ImageFeatureValue::CreateFromVideoFrame(videoFrame);
  binding1.Bind(input.Name(), image);
}
catch (...)
{
  printf("Failed to load/bind image\n");
}
```

```
//get the input descriptor
ILearningModelFeatureDescriptor input = model.InputFeatures[0];
//load a SoftwareBitmap
string imagePath = "path\\to\\fish_720.png";

// Get the image and bind it to the model's input
try
{
    StorageFile file = await StorageFile.GetFileFromPathAsync(imagePath);
    IRandomAccessStream stream = await file.OpenAsync(FileAccessMode.Read);
    BitmapDecoder decoder = await BitmapDecoder.CreateAsync(stream);
    SoftwareBitmap softwareBitmap = await decoder.GetSoftwareBitmapAsync();
    VideoFrame videoFrame = VideoFrame.CreateWithSoftwareBitmap(softwareBitmap);
    ImageFeatureValue image = ImageFeatureValue.CreateFromVideoFrame(videoFrame);
    binding1.Bind(input.Name, image);
}
catch
{
    Console.WriteLine("Failed to load/bind image");
}
```

5. In order for the next model in the chain to use the outputs of the evaluation of the first model, we need to create an empty output tensor and bind the output so we have a marker to chain with:

```
//get the output descriptor
ILearningModelFeatureDescriptor output = model.OutputFeatures().GetAt(0);
//create an empty output tensor
std::vector<int64_t> shape = {1, 3, 720, 720};
TensorFloat outputValue = TensorFloat::Create(shape);
//bind the (empty) output
binding1.Bind(output.Name(), outputValue);
```

```
//get the output descriptor
ILearningModelFeatureDescriptor output = model.OutputFeatures[0];
//create an empty output tensor
List<long> shape = new List<long> { 1, 3, 720, 720 };
TensorFloat outputValue = TensorFloat.Create(shape);
//bind the (empty) output
binding1.Bind(output.Name, outputValue);
```

NOTE

You must use the **TensorFloat** data type when binding the output. This will prevent de-tensorization from occurring once evaluation for the first model is completed, therefore also avoiding additional GPU queueing for load and bind operations for the second model.

6. Now, we run the evaluation of the first model, and bind its outputs to the next model's input:

```
//run session1 evaluation
session1.EvaluateAsync(binding1, L"");
//bind the output to the next model input
binding2.Bind(input.Name(), outputValue);
//run session2 evaluation
auto session2AsyncOp = session2.EvaluateAsync(binding2, L"");
```

```
//run session1 evaluation
await session1.EvaluateAsync(binding1, "");
//bind the output to the next model input
binding2.Bind(input.Name, outputValue);
//run session2 evaluation
LearningModelEvaluationResult results = await session2.EvaluateAsync(binding2, "");
```

7. Finally, let's retrieve the final output produced after running both models by using the following line of code.

```
auto finalOutput = session2AsyncOp.get().Outputs().First().Current().Value();
```

```
var finalOutput = results.Outputs.First().Value;
```

That's it! Both your models now can execute sequentially by making the most of the available GPU resources.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Release notes

9/14/2020 • 2 minutes to read • Edit Online

This page records updates to Windows ML in the latest builds of the Windows 10 SDK and NuGet Package.

Windows ML NuGet Package – Version 1.4

- Download NuGet here
- Built on ONNX Runtime 1.4
- Support for ONNX 1.6 and opset 11
- General usability and performance improvements

Windows ML NuGet Package - Version 1.3

- Download NuGet here
- Built on ONNX Runtime 1.3
- Corresponds to MachineLearningContract v3
- Support for ONNX 1.6 and opset 11
- CPU execution supported down to Windows 8.1; GPU execution supported down to Windows 10 version 1709
- Certified known tested paths are Desktop Applications using C++. Store applications and the Windows Application Certification Kit are not yet supported.

Build 19041 (Windows 10, version 2004)

Support for ONNX 1.4 and opset 9 (CPU and GPU)

API Surface additions:

- CloseModelOnSessionCreation: new **LearningModelSessionOptions** parameter to configure to reduce working memory.

Tooling:

- WinMLTools converters supports new ONNX versions and opset
- Optimizations to WinMLRunner exposing new performance metrics

Build 18362 (Windows 10, version 1903)

All features and updates from previous flighted builds:

- ONNX 1.3 support
- Support for model size reduction via post-training weight quantization. You can use the latest version of WinMLTools to pack the weights of your model down to int8.
- Removal of mlgen from the Windows 10 SDK—use one of the following Visual Studio extensions instead:
  - Visual Studio 2017: Windows Machine Learning Code Generator VS 2017
  - Visual Studio 2019: Windows Machine Learning Code Generator

Build 18829

- mlgen has been removed from the Windows 10 SDK. Instead, install one of the following Visual Studio

extensions depending on your version:
  - Visual Studio 2017: Windows Machine Learning Code Generator VS 2017
  - Visual Studio 2019: Windows Machine Learning Code Generator

Build 18290

- Min supported ONNX version = 1.2.2 (opset 7)
- Max supported ONNX version = 1.3 (opset 8)
- Supports model size reduction via post-training weight quantization. You can use the latest version of WinMLTools to pack the weights of your model down to int8.

Build 17763 (Windows 10, version 1809)

- First official release of Windows Machine Learning.
- Requires ONNX v1.2.
- Windows.AI.MachineLearning.Preview namespace deprecated in favor of Windows.AI.MachineLearning namespace.

Known issues

- For models containing sequences, MLGen generates an **IList<Dictionary<*key*, *value*>>** instead of the proper **IList<IDictionary<*key*, *value*>>**, leading to empty results. To fix this issue, simply replace the automatically generated code with the appropriate **IList<IDictionary<*key*, *value*>>** instead.

Build 17723

- Requires ONNX v1.2.
- Supports F16 datatypes with GPU-based model inferences for better performance and reduced model footprint. You can use WinMLTools to convert your models from FP32 to FP16.
- Allows desktop apps to consume Windows.AI.MachineLearning APIs with WinRT/C++.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Windows Machine Learning API reference

9/14/2020 • 2 minutes to read • Edit Online

The WinML APIs are divided roughly into three areas, which are listed below.

NAME	DESCRIPTION
Core APIs	The main WinML APIs that are used to load, bind, and evaluate models. Located in the **Windows.AI.MachineLearning** namespace.
Custom operators	APIs that handle custom operators in WinML. Located in **MLOperatorAuthor.h**.
Native APIs	Native WinML APIs that let you interact with Direct3D. Located in **windows.ai.machinelearning.native.h**.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Custom operators

8/29/2019 • 2 minutes to read • Edit Online

The Windows Machine Learning custom operator Win32 APIs are located in **MLOperatorAuthor.h**.

APIs

The following is a list of the custom operator APIs with their syntax and descriptions.

Enumerations

NAME	DESCRIPTION
MLOperatorAttributeType	Specifies the type of an attribute. Each attribute type numerically matches the corresponding ONNX type.
MLOperatorEdgeType	Specifies the types of an input or output edge of an operator.
MLOperatorExecutionType	Specifies whether a kernel uses the CPU or GPU for computation.
MLOperatorKernelOptions	Specifies options used when registering custom operator kernels.
MLOperatorParameterOptions	Specifies option flags of input and output edges of operators.
MLOperatorSchemaEdgeTypeFormat	Specifies the manner in which types of input and output edges are described.
MLOperatorTensorDataType	Specifies the data type of a tensor. Each data type numerically matches the corresponding ONNX type.

Functions

NAME	DESCRIPTION
MLCreateOperatorRegistry	Creates an instance of IMLOperatorRegistry which may be used to register a custom operator kernel and custom operator schema.

Interfaces

NAME	DESCRIPTION
IMLOperatorAttributes	Represents the values of an operator's attributes, as determined by a model using the operator.
IMLOperatorKernel	Implemented by custom operator kernels.
IMLOperatorKernelContext	Provides information about an operator's usage while kernels are being computed.

NAME	DESCRIPTION
IMLOperatorKernelCreationContext	Provides information about an operator's usage while kernels are being created.
IMLOperatorKernelFactory	Implemented by the author of a custom operator kernel to create instances of that kernel.
IMLOperatorRegistry	Represents an instance of a registry for the custom operator kernel and schema.
IMLOperatorShapeInferenceContext	Provides information about an operator's usage while shape inferrers are being invoked.
IMLOperatorShapeInferrer	Implemented by shape inferrers to infer shapes of an operator's output tensor edges.
IMLOperatorTensor	Representation of a tensor used during computation of custom operator kernels.
IMLOperatorTensorShapeDescription	Represents the set of input and output tensor shapes of an operator.
IMLOperatorTypeInferenceContext	Provides information about an operator's usage while type inferrers are being invoked.
IMLOperatorTypeInferrer	Implemented by type inferrers to infer the types of an operator's output edges.

Structures

NAME	DESCRIPTION
MLOperatorAttribute	Specifies the name and properties of an attribute of a custom operator.
MLOperatorAttributeNameValue	Specifies the name and value(s) of an attribute of a custom operator.
MLOperatorEdgeDescription	Specifies the properties of an input or output edge of an operator.
MLOperatorEdgeTypeConstraint	Specifies constraints upon the types of edges supported in custom operator kernels and schema.
MLOperatorKernelDescription	Description of a custom operator kernel used to register that schema.
MLOperatorSchemaDescription	Description of a custom operator schema used to register that schema.
MLOperatorSchemaEdgeDescription	Specifies information about an input or output edge of an operator.
MLOperatorSetId	Specifies the identity of an operator set.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorAttributeType enum

8/29/2019 • 2 minutes to read • Edit Online

Specifies the type of an attribute. Each attribute type numerically matches the corresponding ONNX type.

Fields

NAME	VALUE	DESCRIPTION
Undefined	0	Undefined (unused).
Float	2	32-bit floating point.
Int	3	64-bit integer.
String	4	String value.
FloatArray	7	Array of 32-bit floating point values.
IntArray	8	Array of 64-bit integer values.
StringArray	9	Array of string values.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorEdgeType enum

9/14/2020 • 2 minutes to read • Edit Online

Specifies the types of an input or output edge of an operator.

Fields

NAME	VALUE	DESCRIPTION
Undefined	0	
Tensor	1	

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

MLOperatorExecutionType enum

9/14/2020 • 2 minutes to read • Edit Online

Specifies whether a kernel uses the CPU or GPU for computation.

Fields

NAME	VALUE	DESCRIPTION
Undefined	0	
Cpu	1	
D3D12	2	

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorKernelOptions enum

9/14/2020 • 2 minutes to read • Edit Online

Specifies options used when registering custom operator kernels.

Fields

NAME	VALUE	DESCRIPTION
None	0	
AllowDynamicInputShapes	1	Specifies whether the shapes of input tensors are allowed to vary among invocations of an operator kernel instance. If this is not set, kernel instances may query input tensor shapes during creation, and front-load initialization work which depends on those shapes. Setting this may improve performance if shapes vary dynamically between inference operations, and the kernel implementation handles this efficiently.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorParameterOptions enum

9/14/2020 • 2 minutes to read • Edit Online

Specifies option flags of input and output edges of operators. These options are used while defining custom operator schema.

Fields

NAME	VALUE	DESCRIPTION
Single	0	There is a single instance of the input or output.
Optional	1	The input or output may be omitted.
Variadic	2	The number of instances of the operator is variable. Variadic parameters must be last among the set of inputs or outputs.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorSchemaEdgeTypeFormat enum

8/29/2019 • 2 minutes to read • Edit Online

Specifies the manner in which types of input and output edges are described. This is used within MLOperatorSchemaEdgeDescription while defining custom operator schema.

Fields

NAME	VALUE	DESCRIPTION
EdgeDescription	0	The type is defined using MLOperatorEdgeDescription.
Label	1	The type is defined by a type string constructed as in ONNX operator schema.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorTensorDataType enum

8/29/2019 • 2 minutes to read • Edit Online

Specifies the data type of a tensor. Each data type numerically matches the corresponding ONNX type.

Fields

NAME	VALUE	DESCRIPTION
Undefined	0	Undefined (unused).
Float	1	IEEE 32-bit floating point.
UInt8	2	8-bit unsigned integer.
Int8	3	8-bit signed integer.
UInt16	4	16-bit unsigned integer.
Int16	5	16-bit signed integer.
Int32	6	32-bit signed integer.
Int64	7	64-bit signed integer.
String	8	String (unsupported).
Bool	9	8-bit boolean. Values other than zero and one result in undefined behavior.
Float16	10	IEEE 16-bit floating point.
Double	11	64-bit double-precision floating point.
UInt32	12	32-bit unsigned integer.
UInt64	13	64-bit unsigned integer.
Complex64	14	64-bit complex type (unsupported).
Complex128	15	128-bit complex type (unsupported).

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience

Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLCreateOperatorRegistry function

8/29/2019 • 2 minutes to read • Edit Online

Creates an instance of IMLOperatorRegistry which may be used to register a custom operator kernel and custom operator schema.

```
HRESULT MLCreateOperatorRegistry(
    _COM_Outptr_ IMLOperatorRegistry** registry)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorAttributes interface

8/29/2019 • 2 minutes to read • Edit Online

Represents the values of an operator's attributes, as determined by a model using the operator. This interface is called by implementations of custom operator kernels, and by implementations of shape and type inferrers.

Methods

NAME	DESCRIPTION
GetAttribute	Gets the value of an attribute element which is of a numeric type.
GetAttributeElementCount	Gets the count of elements in an attribute.
GetStringAttributeElement	Gets the value of an attribute element which is of a string type.
GetStringAttributeElementLength	Gets the length of an attribute element which is of a string type.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorAttributes.GetAttribute method

9/14/2020 • 2 minutes to read • Edit Online

Gets the value of an attribute element which is of a numeric type. For attributes which are of array types, this method queries an individual element within the attribute at the specified index.

```
void GetAttribute(
    _In_z_ const char* name,
    MLOperatorAttributeType type,
    uint32_t elementCount,
    size_t elementByteSize,
    _Out_writes_bytes_(elementCount * elementByteSize) void* value)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorAttributes.GetAttributeElementCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the count of elements in an attribute. This may be used to determine if an attribute exists, and to determine the count of elements within an attribute of an array type.

```
void GetAttributeElementCount(
    _In_z_ const char* name,
    MLOperatorAttributeType type,
    _Out_ uint32_t* elementCount)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorAttributes.GetStringAttributeElement method

9/14/2020 • 2 minutes to read • Edit Online

Gets the value of an attribute element which is of a string type. For attributes which are string arrays, this method queries the value of an individual element within the attribute at the specified index. The string is in UTF-8 format. The size includes the null termination character.

```
void GetStringAttributeElement(
    _In_z_ const char* name,
    uint32_t elementIndex,
    uint32_t attributeElementByteSize,
    _Out_writes_(uint32_t) char* attributeElement)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorAttributes.GetStringAttributeElementLength method

9/14/2020 • 2 minutes to read • Edit Online

Gets the length of an attribute element which is of a string type. For attributes which are string arrays, this method queries the size of an individual element within the attribute at the specified index. The string is in UTF-8 format. The size includes the null termination character.

```
void GetStringAttributeElementLength(
    _In_z_ const char* name,
    uint32_t elementIndex,
    _Out_ uint32_t* attributeElementByteSize)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernel interface

9/14/2020 • 2 minutes to read • Edit Online

Implemented by custom operator kernels. A factory which creates instances of this interface is supplied when registering custom operator kernels using IMLOperatorRegistry::RegisterOperatorKernel.

Methods

NAME	DESCRIPTION
Compute	Computes the outputs of the kernel.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernel.Compute method

9/14/2020 • 2 minutes to read • Edit Online

Computes the outputs of the kernel. The implementation of this method should be thread-safe. The same instance of the kernel may be computed simultaneously on different threads.

```
void Compute(
    IMLOperatorKernelContext* context)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelContext interface

8/29/2019 • 2 minutes to read • Edit Online

Provides information about an operator's usage while kernels are being computed.

Methods

NAME	DESCRIPTION
AllocateTemporaryData	Allocates temporary data which will be usable as intermediate memory for the duration of a call to IMLOperatorKernel::Compute.
GetExecutionInterface	Returns an object whose supported interfaces vary based on the kernel type.
GetInputTensor	Gets the input tensor of the operator at the specified index.
GetOutputTensor(uint32_t, IMLOperatorTensor**)	Gets the output tensor of the operator at the specified index.
GetOutputTensor(uint32_t, uint32_t, const uint32_t*, IMLOperatorTensor**)	Gets the output tensor of the operator at the specified index, while declaring its shape.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelContext.AllocateTemporaryData method

8/29/2019 • 2 minutes to read • Edit Online

Allocates temporary data which will be usable as intermediate memory for the duration of a call to IMLOperatorKernel::Compute. This may be used by kernels registered using MLOperatorExecutionType::D3D12. The data object supports the ID3D12Resource interface, and is a GPU buffer.

```
void AllocateTemporaryData(
    size_t size,
    _COM_Outptr_ IUnknown** data)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelContext.GetExecutionInterface method

8/29/2019 • 2 minutes to read • Edit Online

Returns an object whose supported interfaces vary based on the kernel type. For kernels registered with MLOperatorExecutionType::Cpu, *executionObject* will be set to **nullptr**. For kernels registered with **MLOperatorExecutionType::D3D12**, *executionObject* will support the ID3D12GraphicsCommandList interface. This may be a different object than was provided to IMLOperatorKernelCreationContext::GetExecutionInterface when the kernel instance was created.

```
void GetExecutionInterface(
    _COM_Outptr_result_maybenull_ IUnknown** executionObject)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelContext.GetInputTensor method

9/14/2020 • 2 minutes to read • Edit Online

Gets the input tensor of the operator at the specified index. This sets the tensor to **nullptr** for optional inputs which do not exist. Returns an error if the input at the specified index is not a tensor.

```
void GetInputTensor(
    uint32_t inputIndex,
    _COM_Outptr_result_maybenull_ IMLOperatorTensor** tensor)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelContext.GetOutputTensor method

9/14/2020 • 2 minutes to read • Edit Online

Overloads

NAME	DESCRIPTION
GetOutputTensor(uint32_t, IMLOperatorTensor**)	Gets the output tensor of the operator at the specified index.
GetOutputTensor(uint32_t, uint32_t, const uint32_t*, IMLOperatorTensor**)	Gets the output tensor of the operator at the specified index, while declaring its shape.

GetOutputTensor(uint32_t, IMLOperatorTensor**)

Gets the output tensor of the operator at the specified index. This sets the tensor to **nullptr** for optional outputs which do not exist. If the operator kernel was registered without a shape inference method, then the overload of **GetOutputTensor** which consumes the tensor's shape must be called instead. Returns an error if the output at the specified index is not a tensor.

```
void GetOutputTensor(
    uint32_t outputIndex,
    _COM_Outptr_result_maybenull_ IMLOperatorTensor** tensor)
```

GetOutputTensor(uint32_t, uint32_t, const uint32_t*, IMLOperatorTensor**)

Gets the output tensor of the operator at the specified index, while declaring its shape. This returns **nullptr** for optional outputs which do not exist. If the operator kernel was registered with a shape inference method, then the overload of **GetOutputTensor** which doesn't consume a shape may also be called. Returns an error if the output at the specified index is not a tensor.

```
void GetOutputTensor(
    uint32_t outputIndex,
    uint32_t dimensionCount,
    _In_reads_(dimensionCount) const uint32_t* dimensionSizes,
    _COM_Outptr_result_maybenull_ IMLOperatorTensor** tensor)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelContext.GetOutputTensor method

9/14/2020 • 2 minutes to read • Edit Online

Overloads

NAME	DESCRIPTION
GetOutputTensor(uint32_t, IMLOperatorTensor**)	Gets the output tensor of the operator at the specified index.
GetOutputTensor(uint32_t, uint32_t, const uint32_t*, IMLOperatorTensor**)	Gets the output tensor of the operator at the specified index, while declaring its shape.

GetOutputTensor(uint32_t, IMLOperatorTensor**)

Gets the output tensor of the operator at the specified index. This sets the tensor to **nullptr** for optional outputs which do not exist. If the operator kernel was registered without a shape inference method, then the overload of **GetOutputTensor** which consumes the tensor's shape must be called instead. Returns an error if the output at the specified index is not a tensor.

```
void GetOutputTensor(
    uint32_t outputIndex,
    _COM_Outptr_result_maybenull_ IMLOperatorTensor** tensor)
```

GetOutputTensor(uint32_t, uint32_t, const uint32_t*, IMLOperatorTensor**)

Gets the output tensor of the operator at the specified index, while declaring its shape. This returns **nullptr** for optional outputs which do not exist. If the operator kernel was registered with a shape inference method, then the overload of **GetOutputTensor** which doesn't consume a shape may also be called. Returns an error if the output at the specified index is not a tensor.

```
void GetOutputTensor(
    uint32_t outputIndex,
    uint32_t dimensionCount,
    _In_reads_(dimensionCount) const uint32_t* dimensionSizes,
    _COM_Outptr_result_maybenull_ IMLOperatorTensor** tensor)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext interface

8/29/2019 • 2 minutes to read • Edit Online

Provides information about an operator's usage while kernels are being created.

Methods

NAME	DESCRIPTION
GetExecutionInterface	Returns an object whose supported interfaces vary based on the kernel type.
GetInputCount	Gets the number of inputs to the operator.
GetInputEdgeDescription	Gets the description of the specified input edge of the operator.
GetOutputCount	Gets the number of outputs to the operator.
GetOutputEdgeDescription	Gets the description of the specified output edge of the operator.
GetTensorShapeDescription	Gets the description of input and output shapes connected to operator edges.
HasTensorShapeDescription	Returns true if the description of input and output shapes connected to operator edges may be queried using **GetTensorShapeDescription**.
IsInputValid	Returns true if an input to the operator is valid. This always returns true except for optional inputs.
IsOutputValid	Returns true if an output to the operator is valid. This always returns true except for optional outputs.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.GetExecutionInterface method

8/29/2019 • 2 minutes to read • Edit Online

Returns an object whose supported interfaces vary based on the kernel type. For kernels registered with MLOperatorExecutionType::Cpu, *executionObject* will be set to **nullptr**. For kernels registered with **MLOperatorExecutionType::D3D12**, *executionObject* will support the ID3D12GraphicsCommandList interface.

```
void GetExecutionInterface(
    _COM_Outptr_result_maybenull_ IUnknown** executionObject)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.GetInputCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of inputs to the operator.

```
uint32_t GetInputCount()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.GetInputEdgeDescription method

9/14/2020 • 2 minutes to read • Edit Online

Gets the description of the specified input edge of the operator.

```
void GetInputEdgeDescription(
    uint32_t inputIndex,
    _Out_ MLOperatorEdgeDescription* edgeDescription)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.GetOutputCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of outputs to the operator.

```
uint32_t GetOutputCount()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.GetOutputEdgeDescription method

9/14/2020 • 2 minutes to read • Edit Online

Gets the description of the specified output edge of the operator.

```
void GetOutputEdgeDescription(
    uint32_t outputIndex,
    _Out_ MLOperatorEdgeDescription* edgeDescription)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.GetTensorShapeDescription method

8/29/2019 • 2 minutes to read • Edit Online

Gets the description of input and output shapes connected to operator edges.

```
void GetTensorShapeDescription(
    _COM_Outptr_ IMLOperatorTensorShapeDescription** shapeDescription)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.HasTensorShapeDescription method

8/29/2019 • 2 minutes to read • Edit Online

Returns true if the description of input and output shapes connected to operator edges may be queried using IMLOperatorKernelCreationContext::GetTensorShapeDescription. This returns true unless the operator was registered using the MLOperatorKernelOptions::AllowDynamicInputShapes flag.

```
bool HasTensorShapeDescription()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.IsInputValid method

9/14/2020 • 2 minutes to read • Edit Online

Returns true if an input to the operator is valid. This always returns true except for optional inputs.

```
bool IsInputValid(uint32_t inputIndex)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelCreationContext.IsOutputValid method

9/14/2020 • 2 minutes to read • Edit Online

Returns true if an output to the operator is valid. This always returns true except for optional outputs.

```
bool IsOutputValid(uint32_t outputIndex)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelFactory interface

8/29/2019 • 2 minutes to read • Edit Online

Implemented by the author of a custom operator kernel to create instances of that kernel.

Methods

NAME	DESCRIPTION
CreateKernel	Creates an instance of the associated operator kernel, given information about the operator's usage within a model described in the provided context object.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorKernelFactory.CreateKernel method

8/29/2019 • 2 minutes to read • Edit Online

Creates an instance of the associated operator kernel, given information about the operator's usage within a model described in the provided context object.

```
void CreateKernel(
    IMLOperatorKernelCreationContext* context,
    _COM_Outptr_ IMLOperatorKernel** kernel)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorRegistry interface

8/29/2019 • 2 minutes to read • Edit Online

Represents an instance of a registry for the custom operator kernel and schema. Custom operators may be used with Windows.AI.MachineLearning APIs by returning instances of **IMLOperatorRegistry** through **ILearningModelOperatorProviderNative**.

Methods

NAME	DESCRIPTION
RegisterOperatorKernel	Registers a custom operator kernel.
RegisterOperatorSetSchema	Registers a set of custom operator schema comprising an operator set.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorRegistry.RegisterOperatorKernel method

9/14/2020 • 2 minutes to read • Edit Online

Registers a custom operator kernel. A shape inferrer may optionally be provided. This may improve performance and enables the kernel to query the shape of its output tensors when it is created and computed.

```
void RegisterOperatorKernel(
    const MLOperatorKernelDescription* operatorKernel,
    IMLOperatorKernelFactory* operatorKernelFactory,
    _In_opt_ IMLOperatorShapeInferrer* shapeInferrer)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorRegistry.RegisterOperatorSetSchema method

9/14/2020 • 2 minutes to read • Edit Online

Registers a set of custom operator schema comprising an operator set. Operator sets follow the ONNX versioning design. Callers should provide schema for all operators that have changed between the specified baseline version and the version specified within *operatorSetId*. This prevents older versions of kernels from being used in models which import the newer operator set version. A type inferrer must be provided if the MLOperatorSchemaDescription structure cannot express how output types are determined. A shape inferrer may optionally be provided to enable model validation.

```
void RegisterOperatorSetSchema(
    const MLOperatorSetId* operatorSetId,
    int32_t baselineVersion,
    _In_reads_opt_(schemaCount) const MLOperatorSchemaDescription* const* schema,
    uint32_t schemaCount,
    _In_opt_ IMLOperatorTypeInferrer* typeInferrer,
    _In_opt_ IMLOperatorShapeInferrer* shapeInferrer)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext interface

8/29/2019 • 2 minutes to read • Edit Online

Provides information about an operator's usage while shape inferrers are being invoked.

Methods

NAME	DESCRIPTION
GetInputCount	Gets the number of inputs to the operator.
GetInputEdgeDescription	Gets the description of the specified input edge of the operator.
GetInputTensorDimensionCount	Gets the number of dimensions of a tensor output of the operator.
GetInputTensorShape	Gets the sizes of dimensions of an input tensor of the operator.
GetOutputCount	Gets the number of outputs to the operator.
IsInputValid	Returns true if an input to the operator is valid.
IsOutputValid	Returns true if an output to the operator is valid.
SetOutputTensorShape	Sets the inferred shape of an output tensor.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.GetInputCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of inputs to the operator.

```
uint32_t GetInputCount()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.GetInputEdgeDescription method

9/14/2020 • 2 minutes to read • Edit Online

Gets the description of the specified input edge of the operator.

```
void GetInputEdgeDescription(
    uint32_t inputIndex,
    _Out_ MLOperatorEdgeDescription* edgeDescription)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.GetInputTensorDimensionCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of dimensions of a tensor output of the operator.

```
void GetInputTensorDimensionCount(
    uint32_t inputIndex,
    _Out_ uint32_t* dimensionCount)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.GetInputTensorShape method

9/14/2020 • 2 minutes to read • Edit Online

Gets the sizes of dimensions of an input tensor of the operator. Returns an error if the input at the specified index is not a tensor.

```
void GetInputTensorShape(
    uint32_t inputIndex,
    uint32_t dimensionCount,
    _Out_writes_(dimensionCount) uint32_t* dimensions)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.GetOutputCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of outputs to the operator.

```
uint32_t GetOutputCount()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.IsInputValid method

9/14/2020 • 2 minutes to read • Edit Online

Returns true if an input to the operator is valid. This always returns true except for optional inputs and invalid indices.

```
bool IsInputValid(
    uint32_t inputIndex)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.IsOutputValid method

9/14/2020 • 2 minutes to read • Edit Online

Returns true if an output to the operator is valid. This always returns true except for optional outputs and invalid indices.

```
bool IsOutputValid(
    uint32_t outputIndex)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferenceContext.SetOutputTensorShape method

9/14/2020 • 2 minutes to read • Edit Online

Sets the inferred shape of an output tensor. Returns an error if the output at the specified index is not a tensor.

```
void SetOutputTensorShape(
    uint32_t outputIndex,
    uint32_t dimensionCount,
    const uint32_t* dimensions)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferrer interface

8/29/2019 • 2 minutes to read • Edit Online

Implemented by shape inferrers to infer shapes of an operator's output tensor edges. Shape inferrers may be provided when registering custom operator kernels to improve performance and to enable the kernel to query the shape of its output tensors when it is created and computed. Shape inferrers may also be provided when registering custom operator schema to improve model validation.

Methods

NAME	DESCRIPTION
InferOutputShapes	Called to infer shapes of an operator's output edges.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorShapeInferrer.InferOutputShapes method

9/14/2020 • 2 minutes to read • Edit Online

Called to infer shapes of an operator's output edges.

```
void InferOutputShapes(
    IMLOperatorShapeInferenceContext* context)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor interface

8/29/2019 • 2 minutes to read • Edit Online

Representation of a tensor used during computation of custom operator kernels.

Methods

NAME	DESCRIPTION
GetData	Returns a pointer to byte-addressable memory for the tensor.
GetDataInterface	Gets an interface pointer for the tensor.
GetDimensionCount	Gets the number of dimensions in the tensor.
GetShape	Gets the size of dimensions in the tensor.
GetTensorDataType	Gets the data type of the tensor.
IsCpuData	Indicates whether the memory used by the tensor is CPU-addressable.
IsDataInterface	Whether the contents of the tensor are represented by an interface type, or byte-addressable memory.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor.GetData method

9/14/2020 • 2 minutes to read • Edit Online

Returns a pointer to byte-addressable memory for the tensor. This may be used when IMLOperatorTensor::IsDataInterface returns false, because the kernel was registered using MLOperatorExecutionType::Cpu. The data size is derived from the tensor's shape. It is fully packed in memory.

```
void* GetData()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor.GetDataInterface method

9/14/2020 • 2 minutes to read • Edit Online

Gets an interface pointer for the tensor. This may be used when IMLOperatorTensor::IsDataInterface returns true, because the kernel was registered using MLOperatorExecutionType::D3D12. The *dataInterface* object supports the ID3D12Resource interface, and is a GPU buffer.

```
void GetDataInterface(
    _COM_Outptr_result_maybenull_ IUnknown** dataInterface)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor.GetDimensionCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of dimensions in the tensor. This may be zero.

```
uint32_t GetDimensionCount()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor.GetShape method

9/14/2020 • 2 minutes to read • Edit Online

Gets the size of dimensions in the tensor.

```
void GetShape(
    uint32_t dimensionCount,
    _Out_writes_(dimensionCount) uint32_t* dimensions)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor.GetTensorDataType method

9/14/2020 • 2 minutes to read • Edit Online

Gets the data type of the tensor.

```
MLOperatorTensorDataType GetTensorDataType()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor.IsCpuData method

9/14/2020 • 2 minutes to read • Edit Online

Indicates whether the memory used by the tensor is CPU-addressable. This is true when kernels are registered using MLOperatorExecutionType::Cpu.

```
bool IsCpuData()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensor.IsDataInterface method

8/29/2019 • 2 minutes to read • Edit Online

Whether the contents of the tensor are represented by an interface type, or byte-addressable memory. This returns true when kernels are registered using MLOperatorExecutionType::D3D12.

```
bool IsDataInterface()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensorShapeDescription interface

9/14/2020 • 2 minutes to read • Edit Online

Represents the set of input and output tensor shapes of an operator. This interface is called by the factory objects registered to create kernels. It is available to these factory objects unless corresponding kernels are registered using the MLOperatorKernelOptions::AllowDynamicInputShapes flag.

Methods

NAME	DESCRIPTION
GetInputTensorDimensionCount	Gets the number of dimensions of a tensor input of the operator.
GetInputTensorShape	Gets the sizes of dimensions of an input tensor of the operator.
GetOutputTensorDimensionCount	Gets the number of dimensions of a tensor output of the operator.
GetOutputTensorShape	Gets the sizes of dimensions of a tensor output of the operator.
HasOutputShapeDescription	Returns true if output shapes may be queried using **GetOutputTensorDimensionCount** and **GetOutputTensorShape**.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensorShapeDescription.GetInputTensorDimensionCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of dimensions of a tensor input of the operator. Returns an error if the input at the specified index is not a tensor.

```
void GetInputTensorDimensionCount(
    uint32_t inputIndex,
    _Out_ uint32_t* dimensionCount)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensorShapeDescription.GetInputTensorShape method

9/14/2020 • 2 minutes to read • Edit Online

Gets the sizes of dimensions of an input tensor of the operator. Returns an error if the input at the specified index is not a tensor.

```
void GetInputTensorShape(
    uint32_t inputIndex,
    uint32_t dimensionCount,
    _Out_writes_(dimensionCount) uint32_t* dimensions)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensorShapeDescription.GetOutputTensorDimensionCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of dimensions of a tensor output of the operator. Returns an error if the output at the specified index is not a tensor.

```
GetOutputTensorDimensionCount(
    uint32_t outputIndex,
    _Out_ uint32_t* dimensionCount)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensorShapeDescription.GetOutputTensorShape method

9/14/2020 • 2 minutes to read • Edit Online

Gets the sizes of dimensions of a tensor output of the operator. Returns an error if the output at the specified index is not a tensor.

```
GetOutputTensorShape(
    uint32_t outputIndex,
    uint32_t dimensionCount,
    _Out_writes_(dimensionCount) uint32_t* dimensions)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTensorShapeDescription.HasOutputShapeDescription method

8/29/2019 • 2 minutes to read • Edit Online

Returns true if output shapes may be queried using IMLOperatorTensorShapeDescription::GetOutputTensorDimensionCount and IMLOperatorTensorShapeDescription::GetOutputTensorShape. This is true if the kernel was registered with a shape inferrer.

```
bool HasOutputShapeDescription()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferenceContext interface

8/29/2019 • 2 minutes to read • Edit Online

Provides information about an operator's usage while type inferrers are being invoked.

Methods

NAME	DESCRIPTION
GetInputCount	Gets the number of inputs to the operator.
GetInputEdgeDescription	Gets the description of the specified input edge of the operator.
GetOutputCount	Gets the number of outputs to the operator.
IsInputValid	Returns true if an input to the operator is valid.
IsOutputValid	Returns true if an output to the operator is valid.
SetOutputEdgeDescription	Sets the inferred type of an output edge.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferenceContext.GetInputCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of inputs to the operator.

```
uint32_t GetInputCount()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferenceContext.GetInputEdgeDescription method

9/14/2020 • 2 minutes to read • Edit Online

Gets the description of the specified input edge of the operator.

```
void GetInputEdgeDescription(
    uint32_t inputIndex,
    _Out_ MLOperatorEdgeDescription* edgeDescription)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferenceContext.GetOutputCount method

9/14/2020 • 2 minutes to read • Edit Online

Gets the number of outputs to the operator.

```
uint32_t GetOutputCount()
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferenceContext.IsInputValid method

9/14/2020 • 2 minutes to read • Edit Online

Returns true if an input to the operator is valid. This always returns true except for optional inputs.

```
bool IsInputValid(
    uint32_t inputIndex)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferenceContext.IsOutputValid method

9/14/2020 • 2 minutes to read • Edit Online

Returns true if an output to the operator is valid. This always returns true except for optional outputs.

```
bool IsOutputValid(
    uint32_t outputIndex)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferenceContext.SetOutputEdgeDescription method

9/14/2020 • 2 minutes to read • Edit Online

Sets the inferred type of an output edge.

```
void SetOutputEdgeDescription(
    uint32_t outputIndex,
    const MLOperatorEdgeDescription* edgeDescription)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferrer interface

8/29/2019 • 2 minutes to read • Edit Online

Implemented by type inferrers to infer the types of an operator's output edges. Type inferrers must be provided when registering schema of custom operators if the MLOperatorSchemaDescription structure cannot express how output types are determined—for example, when an attribute of the operator determines the data type of one of that operator's outputs.

Methods

NAME	DESCRIPTION
InferOutputTypes	Called to infer types of an operator's output edges.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

IMLOperatorTypeInferrer.InferOutputTypes method

9/14/2020 • 2 minutes to read • Edit Online

Called to infer types of an operator's output edges.

```
void InferOutputTypes(
    IMLOperatorTypeInferenceContext* context)
```

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorAttribute struct

8/29/2019 • 2 minutes to read • Edit Online

Specifies the name and properties of an attribute of a custom operator. This is used when registering custom operator kernels and custom operator schema.

Fields

NAME	TYPE	DESCRIPTION
name	**char***	NULL-terminated UTF-8 string representing the name of the attribute in the associated operator type.
required	**bool**	Whether the attribute is required in any model using the associated operator type.
type	MLOperatorAttributeType	The type of the attribute in the associated operator type.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorAttributeNameValue struct

9/14/2020 • 2 minutes to read • Edit Online

Specifies the name and value(s) of an attribute of a custom operator. This is used when registering custom operator kernels and custom operator schema.

Fields

NAME	TYPE	DESCRIPTION
floats	**const float***	32-bit floating point value(s). Used when the type field is **MLOperatorAttributeType::Float** or **MLOperatorAttributeType::FloatArray**.
ints	**const int64_t***	64-bit integer value(s). Used when the type field is **MLOperatorAttributeType::Int** or **MLOperatorAttributeType::IntArray**.
name	**const char***	NULL-terminated UTF-8 string representing the name of the attribute in the associated operator type.
reserved	**const void***	
strings	**const char* const***	NULL-terminated UTF-8 string value(s). Used when the type field is **MLOperatorAttributeType::String** or **MLOperatorAttributeType::StringArray**.
type	MLOperatorAttributeType	The type of the attribute in the associated operator type.
valueCount	**uint32_t**	The number of elements in the attribute value. This must be 1, except for attributes which are of array types.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorEdgeDescription struct

9/14/2020 • 2 minutes to read • Edit Online

Specifies the properties of an input or output edge of an operator.

Fields

NAME	TYPE	DESCRIPTION
edgeType	MLOperatorEdgeType	The type of the edge.
reserved	**uint64_t**	
tensorDataType	MLOperatorTensorDataType	The data type of a tensor. Used when **edgeType** is set to **Tensor**.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorEdgeTypeConstraint struct

8/29/2019 • 2 minutes to read • Edit Online

Specifies constraints upon the types of edges supported in custom operator kernels and schema. The provided type label string corresponds to type labels in the ONNX specification for the same operator. For custom schema, it corresponds to type labels specified within MLOperatorSchemaEdgeDescription when registering the operator's schema.

Fields

NAME	TYPE	DESCRIPTION
allowedTypeCount	**uint32_t**	
allowedTypes	MLOperatorEdgeDescription*	The set of allowed types for the constraint.
typeLabel	**char***	The label of the type for which the constraint is being defined. This is constructed as in ONNX operator schema. For example, "T".

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorKernelDescription struct

9/14/2020 • 2 minutes to read • Edit Online

Description of a custom operator kernel used to register that schema.

Fields

NAME	TYPE	DESCRIPTION
defaultAttributeCount	**uint32_t**	The number of provided default attribute values.
defaultAttributes	**const** MLOperatorAttributeNameValue*	The default values of attributes. These will be applied when the attributes are missing in a model containing the operator type.
domain	**const char***	NULL-terminated UTF-8 string representing the name of the operator's domain.
executionOptions	**uint32_t**	Reserved for additional options. Must be 0.
executionType	MLOperatorExecutionType	Specifies whether a kernel uses the CPU or GPU for computation.
minimumOperatorSetVersion	**int32_t**	The minimum version of the operator sets for which this kernel is valid. The maximum version is inferred based on registrations of operator set schema for subsequent versions of the same domain.
name	**const char***	NULL-terminated UTF-8 string representing the name of the operator.
options	MLOperatorKernelOptions	Options for the kernel which apply to all execution provider types.
typeConstraintCount	**uint32_t**	The number of type constraints provided.
typeConstraints	**const** MLOperatorEdgeTypeConstraint*	An array of type constraints. Each constraint restricts input and outputs associated with a type label string to one or more edge types.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorSchemaDescription struct

9/14/2020 • 2 minutes to read • Edit Online

Description of a custom operator schema used to register that schema.

Fields

NAME	TYPE	DESCRIPTION
attributeCount	uint32_t	The number of provided attributes.
attributes	const MLOperatorAttribute*	The set of attributes supported by the operator type.
defaultAttributeCount	uint32_t	The number of provided default attribute values.
defaultAttributes	const MLOperatorAttributeNameValue*	The default values of attributes. These will be applied when the attributes are missing in a model containing the operator type.
inputCount	uint32_t	The number of inputs of the operator.
inputs	const MLOperatorSchemaEdgeDescription*	An array containing the descriptions of the operator's input edges.
name	const char*	NULL-terminated UTF-8 string representing the name of the operator.
operatorSetVersionAtLastChange	int32_t	The operator set version at which this operator was introduced or last changed.
outputCount	uint32_t	The number of outputs of the operator.
outputs	const MLOperatorSchemaEdgeDescription*	An array containing the descriptions of the operator's output edges.
typeConstraintCount	uint32_t	The number of type constraints provided.
typeConstraints	const MLOperatorEdgeTypeConstraint*	An array of type constraints. Each constraint restricts input and outputs associated with a type label string to one or more edge types.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorSchemaEdgeDescription struct

9/14/2020 • 2 minutes to read • Edit Online

Specifies information about an input or output edge of an operator. This is used while defining custom operator schema.

Fields

NAME	TYPE	DESCRIPTION
edgeDescription	MLOperatorEdgeDescription	A structure describing type support. This is used when **typeFormat** is **MLOperatorSchemaEdgeTypeFormat::EdgeDescription**.
options	MLOperatorParameterOptions	Options of the parameter, including whether it is optional or variadic.
reserved	**void***	
typeFormat	MLOperatorSchemaEdgeTypeFormat	The manner in which the type constraints and type mapping are defined.
typeLabel	**char***	A type label string constructed as in ONNX operator schema. For example, "T". This is used when **typeFormat** is **MLOperatorSchemaEdgeTypeFormat::Label**.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

MLOperatorSetId struct

9/14/2020 • 2 minutes to read • Edit Online

Specifies the identity of an operator set.

Fields

NAME	TYPE	DESCRIPTION
domain	**const char***	The domain of the operator, for example, "ai.onnx.ml", or an empty string for the ONNX domain.
version	**int32_t**	The version of the operator domain.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	MLOperatorAuthor.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

WinML native APIs

9/14/2020 • 2 minutes to read • Edit Online

The Windows Machine Learning native APIs are located in **windows.ai.machinelearning.native.h**.

APIs

The following is a list of the WinML native APIs with their syntax and descriptions.

Interfaces

NAME	DESCRIPTION
ILearningModelDeviceFactoryNative	Provides access to factory methods that enable the creation of ILearningModelDevice objects using ID3D12CommandQueue.
ITensorNative	Provides access to an ITensor as an array of bytes or ID3D12Resource objects.
ITensorStaticsNative	Provides access to factory methods that enable the creation of ITensor objects using ID3D12Resource.

Structures

NAME	DESCRIPTION
ILearningModelOperatorProviderNative	Provides access to IMLOperatorRegistry objects containing custom operator registrations.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ILearningModelDeviceFactoryNative interface

8/29/2019 • 2 minutes to read • Edit Online

Provides access to factory methods that enable the creation of ILearningModelDevice objects using ID3D12CommandQueue.

Methods

NAME	DESCRIPTION
CreateFromD3D12CommandQueue	Creates a LearningModelDevice that will run inference on the user-specified ID3D12CommandQueue.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ILearningModelDeviceFactoryNative.CreateFromD3D12CommandQueue method

8/29/2019 • 2 minutes to read • Edit Online

Creates a LearningModelDevice that will run inference on the user-specified ID3D12CommandQueue.

```
HRESULT CreateFromD3D12CommandQueue(
    ID3D12CommandQueue * value,
    [out] IUnknown ** result);
```

Parameters

NAME	TYPE	DESCRIPTION
value	ID3D12CommandQueue*	The **ID3D12CommandQueue** which the **LearningModelDevice** will be run against.
result	IUnknown**	The **LearningModelDevice** to be created.

Returns

HRESULT The result of the operation.

Examples

```
 // 1. create the d3d device.
com_ptr<ID3D12Device> pD3D12Device = nullptr;
CHECK_HRESULT(D3D12CreateDevice(
    nullptr,
    D3D_FEATURE_LEVEL::D3D_FEATURE_LEVEL_11_0,
    __uuidof(ID3D12Device),
    reinterpret_cast<void**>(&pD3D12Device)));

// 2. create the command queue.
com_ptr<ID3D12CommandQueue> dxQueue = nullptr;
D3D12_COMMAND_QUEUE_DESC commandQueueDesc = {};
commandQueueDesc.Type = D3D12_COMMAND_LIST_TYPE_DIRECT;
CHECK_HRESULT(pD3D12Device->CreateCommandQueue(
    &commandQueueDesc,
    __uuidof(ID3D12CommandQueue),
    reinterpret_cast<void**>(&dxQueue)));
com_ptr<ILearningModelDeviceFactoryNative> devicefactory =
    get_activation_factory<LearningModelDevice, ILearningModelDeviceFactoryNative>();
com_ptr<::IUnknown> spUnk;
CHECK_HRESULT(devicefactory->CreateFromD3D12CommandQueue(dxQueue.get(), spUnk.put()));
```

See also

- Custom Tensorization Sample

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ITensorNative interface

8/29/2019 • 2 minutes to read • Edit Online

Provides access to an ITensor as an array of bytes or ID3D12Resource objects.

Methods

NAME	DESCRIPTION
GetBuffer	Gets the tensor's buffer as an array of bytes.
GetD3D12Resource	Gets the tensor buffer as an ID3D12Resource.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ITensorNative.GetBuffer method

9/14/2020 • 2 minutes to read • Edit Online

Gets the tensor's buffer as an array of bytes.

```
HRESULT GetBuffer(
    [out, size_is(, *capacity)] BYTE **value,
    [out] UINT32 *capacity);
```

Parameters

NAME	TYPE	DESCRIPTION
value	BYTE**	The tensor's buffer.
capacity	UINT32*	The capacity of the buffer.

Returns

HRESULT The result of the operation.

Examples

```
TensorFloat SoftwareBitmapToSoftwareTensor(SoftwareBitmap softwareBitmap)
{
    // 1. Get access to the buffer of softwareBitmap
    BYTE* pData = nullptr;
    UINT32 size = 0;
    BitmapBuffer spBitmapBuffer(softwareBitmap.LockBuffer(BitmapBufferAccessMode::Read));
    winrt::Windows::Foundation::IMemoryBufferReference reference = spBitmapBuffer.CreateReference();
    auto spByteAccess = reference.as<::Windows::Foundation::IMemoryBufferByteAccess>();
    CHECK_HRESULT(spByteAccess->GetBuffer(&pData, &size));

    // ...
}
```

See also

- Custom Tensorization Sample

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ITensorNative.GetD3D12Resource method

9/14/2020 • 2 minutes to read • Edit Online

Gets the tensor buffer as an ID3D12Resource.

```
HRESULT GetD3D12Resource(
    [out] ID3D12Resource ** result);
```

Parameters

NAME	TYPE	DESCRIPTION
result	ID3D12Resource**	The tensor buffer as an **ID3D12Resource**.

Returns

HRESULT The result of the operation.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ITensorStaticsNative interface

8/29/2019 • 2 minutes to read • Edit Online

Provides access to factory methods that enable the creation of ITensor objects using ID3D12Resource.

Methods

NAME	DESCRIPTION
CreateFromD3D12Resource	Creates a tensor object (TensorFloat, TensorInt32Bit) from a user-specified ID3D12Resource.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ITensorStaticsNative.CreateFromD3D12Resource method

8/29/2019 • 2 minutes to read • Edit Online

Creates a tensor object (TensorFloat, TensorInt32Bit) from a user-specified ID3D12Resource.

```
HRESULT CreateFromD3D12Resource(
    ID3D12Resource *value,
    [size_is(shapeCount)] __int64 *shape,
    int shapeCount,
    [out] IUnknown ** result);
```

Parameters

NAME	TYPE	DESCRIPTION
value	ID3D12Resource*	The **ID3D12Resource** from which to create the tensor.
shape	**__int64***	The shape of the tensor.
shapeCount	**int**	The number of dimensions of the tensor.
result	IUnknown**	The resulting tensor.

Returns

HRESULT The result of the operation.

Examples

```
TensorFloat SoftwareBitmapToDX12Tensor(SoftwareBitmap softwareBitmap)
{
    // ...

    // GPU tensorize
    com_ptr<ITensorStaticsNative> tensorfactory = get_activation_factory<TensorFloat, ITensorStaticsNative>();
    com_ptr<::IUnknown> spUnkTensor;
    TensorFloat input1imagetensor(nullptr);
    int64_t shapes[4] = { 1,3, softwareBitmap.PixelWidth(), softwareBitmap.PixelHeight() };
    CHECK_HRESULT(tensorfactory->CreateFromD3D12Resource(pGPUResource.get(), shapes, 4, spUnkTensor.put()));
    spUnkTensor.try_as(input1imagetensor);

    // ...
}
```

See also

- Custom Tensorization Sample

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ILearningModelOperatorProviderNative struct

8/29/2019 • 2 minutes to read • Edit Online

Provides access to IMLOperatorRegistry objects containing custom operator registrations.

Methods

NAME	DESCRIPTION
GetRegistry	Gets an IMLOperatorRegistry object containing custom operator definitions.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

ILearningModelOperatorProviderNative.GetRegistry method

8/29/2019 • 2 minutes to read • Edit Online

Gets an IMLOperatorRegistry object containing custom operator definitions.

```
void GetRegistry(
    IMLOperatorRegistry** ppOperatorRegistry);
```

Parameters

NAME	TYPE	DESCRIPTION
ppOperatorRegistry	IMLOperatorRegistry**	The **IMLOperatorRegistry** object containing custom operator definitions.

Returns

void This method does not return a value.

Requirements

Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	windows.ai.machinelearning.native.h

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Tools and samples

8/29/2019 • 2 minutes to read • Edit Online

The Windows-Machine-Learning repository on GitHub contains sample applications that demonstrate how to use Windows Machine Learning, as well as tools that help verify models and troubleshoot issues during development.

Tools

The following tools are available on GitHub.

NAME	DESCRIPTION
WinML Code Generator (mlgen)	A Visual Studio extension to help you get started using WinML APIs on UWP apps by generating template code when you add a trained ONNX file into the UWP project. Available for Visual Studio 2017 and 2019. See the documentation for more information.
WinML Dashboard (Preview)	A GUI-based tool for viewing, editing, converting, and validating machine learning models for Windows ML's inference engine.
WinMLRunner	A command-line tool that can run .onnx or .pb models where the input and output variables are tensors or images. Useful because it lets you run the WinML APIs against a model and see if it hits any issues before trying to integrate it into an application.
WinMLTools	A Python package that provides model conversion from different ML toolkits into ONNX, as well as a quantization tool to reduce the memory footprint of a model.

Samples

The following sample applications are available on GitHub.

NAME	DESCRIPTION
AdapterSelection (Win32 C++)	A desktop application that demonstrates how to choose a specific device adapter for running your model.
Custom Operator Sample (Win32 C++)	A desktop application that defines multiple custom CPU operators. One of these is a debug operator that you can integrate into your own workflow.
Custom Tensorization (Win32 C++)	Shows how to tensorize an input image by using the WinML APIs on both the CPU and GPU.
Custom Vision (UWP C#)	Shows how to train an ONNX model in the cloud using Custom Vision, and integrate it into an application with WinML.

NAME	DESCRIPTION
Emoji8 (UWP C#)	Shows how you can use WinML to power a fun emotion-detecting application.
FNS Style Transfer (UWP C#)	Uses the FNS-Candy style transfer model to re-style images or video streams.
MNIST (UWP C#/C++)	Corresponds to Tutorial: Create a Windows Machine Learning UWP application (C#). Start from a basis and work through the tutorial, or run the completed project.
PlaneIdentifier (UWP C#, WPF C#)	Uses a pre-trained machine learning model, generated using the Custom Vision service on Azure, to detect if the given image contains a specific object: a plane.
SqueezeNet Object Detection (Win32 C++, UWP C#/JavaScript)	Uses SqueezeNet, a pre-trained machine learning model, to detect the predominant object in an image selected by the user from a file.
SqueezeNet Object Detection (Azure IoT Edge on Windows, C#)	This is a sample module showing how to run Windows ML inferencing in an Azure IoT Edge module running on Windows. Images are supplied by a connected camera, inferenced against the SqueezeNet model, and sent to IoT Hub.
winml_tracker (ROS C++)	A ROS (Robot Operating System) node which uses WinML to track people (or other objects) in camera frames.

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

FAQ (Frequently Asked Questions)

7/2/2019 • 3 minutes to read • Edit Online

This page contains answers to the most popular questions from the community.

How do I know if the ONNX model I have will run with Windows ML?

The easiest way to check if your model will run with Windows ML is by using the WinML Model Runner tool. Alternatively, you can check ONNX versions and Windows builds for more information on all supported ONNX versions for a given Windows release.

How do I convert a model of a different format to ONNX?

You can use WinMLTools to convert models of several different formats, such as Apple CoreML and scikit-learn, to ONNX.

I am getting errors when trying to export and/or convert my model to ONNX that say my model has "unsupported operators." What should I do?

Some operators in the native training framework might not be currently supported by an ONNX version. First, we recommend you check supported ONNX versions for your target Windows build, and try to convert your model to the max supported version. Later versions of ONNX include support for a larger set of operators compared to previous versions.

If you continue to run into issues, we recommend working with your data science team to try and avoid the unsupported operators. One of the approaches we recommend is to change the model's architecture in the source framework and attempt to convert/export the model to the target ONNX version. Note that you don't need to retrain the model yet—you can attempt to convert the architecture and, if successful, then you can move on to full retraining of your model.

Why can't I load a model?

There are several reasons why you might have trouble loading a model, but one of the most common ones when developing on UWP is due to file access restrictions. By default, UWP applications can only access certain parts of the file system, and require user permission or extra capabilities in order to access other locations. See File access permissions for more information.

Which version of WinMLTools should I use?

We always recommend you download and install the latest version of the **winmltools** package. This will ensure you can create ONNX models that target the latest versions of Windows.

Can I use onnxmltools instead of winmltools?

Yes, you can, but you will need to make sure you install the correct version of onnxmltools in order to target ONNX v1.2.2, which is the minimum ONNX version supported by Windows ML. If you are unsure of which version to install, we recommend installing the latest version of **winmltools** instead. This will ensure you will be able to target the ONNX version supported by Windows.

Which version of Visual Studio should I use in order to get automatic code generation (mlgen)?

The minimum recommended version of Visual Studio with support for mlgen is 15.8.7. In Windows 10, version 1903 and later, **mlgen** is no longer included in the SDK, so you'll need to download and install the extension. There is one for Visual Studio 2017 and one for Visual Studio 2019.

I get an error message when trying to run mlgen and no code is generated. What could possibly be happening?

The two most common errors when trying to execute mlgen are:

- **Required attribute 'consumed_inputs' is missing**: If you run into this error message, then most likely you are trying to run an ONNX v1.2 model with a version of the Windows 10 SDK older than 17763; we recommend that you check your SDK version and update it to version 17763 or later.
- **Type Error: Type (map(string,tensor(float))) of output arg (loss) of node (ZipMap) does not match the expected type...**: If you run into this error, then most likely your ONNX model is an older version than the one accepted by WinML starting with build 17763. We recommend that you update your converter package to the latest available version and reconvert your model to the 1.2 version of ONNX.

What does WinML run on by default?

If you don't specify a device to run on with LearningModelDeviceKind, or if you use **LearningModelDeviceKind.Default**, the system will decide which device will evaluate the model. This is usually the CPU. To make WinML run on the GPU, specify one of the following values when creating the LearningModelDevice:

- **LearningModelDeviceKind.DirectX**
- **LearningModelDeviceKind.DirectXHighPerformance**
- **LearningModelDeviceKind.DirectXMinPower**

NOTE

Use the following resources for help with Windows ML:

- To ask or answer technical questions about Windows ML, please use the **windows-machine-learning** tag on Stack Overflow.
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Windows Vision Skills (Preview)

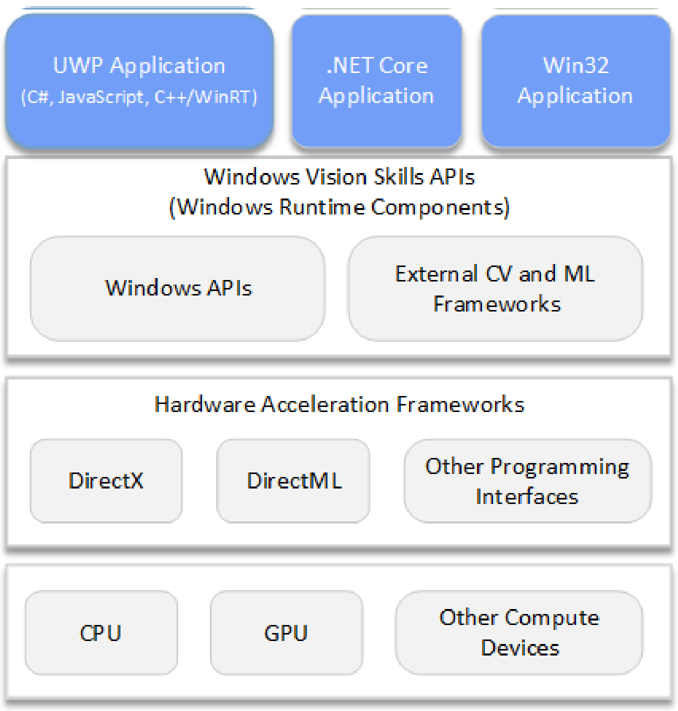
9/14/2020 • 2 minutes to read • Edit Online

NOTE

Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

Implementing and integrating efficient machine learning and computer vision solutions is a hard task for developers. The industry is moving at a fast pace and the amount of custom-tailored solutions coming out makes it strenuous for application developers to keep up. Existing APIs and lower-level frameworks add a steep learning curve before developers can leverage them effectively.

The **Windows Vision Skills** framework is meant to make it easier to utilize computer vision. It standardizes the way computer vision modules are put to use within a Windows application, running on the local device. It abstracts complexities into a single programming paradigm with standardized primitives, helping developers focus on building awesome computer vision applications.


UWP Application
(C#, JavaScript, C++/WinRT)
.NET Core Application
Win32 Application
Windows Vision Skills APIs
(Windows Runtime Components)
Windows APIs
External CV and ML Frameworks
Hardware Acceleration Frameworks
DirectX
DirectML
Other Programming Interfaces
CPU
GPU
Other Compute Devices

The implementation that contains the complex details is encapsulated by an extensible WinRT API that inherits the base classes and interfaces in the Microsoft.AI.Skills.SkillInterfacePreview namespace. This API can be ingested by all types of Windows apps (UWP, .NET Core, and Win32). This framework is open for all developers to build on top of.

- Get the NuGet package

What is a *skill*?

In the context of Windows Vision Skills, a skill is a streamlined, modular piece of code that processes input and produces output. A skill can encapsulate simple functionalities like edge detection with a single-purpose micro-skill, or rich sets of functionalities forming a scenario skill to address a complex problem like skeletal detection.

Benefits

- **Simple integration**: Skills make it easy to add features to your application with out-of-the-box APIs that don't require any machine learning or computer vision expertise, or prior Windows knowledge of low-level APIs.
- **Abstracting hardware acceleration**: The Windows Vision Skills framework queries the hardware assets and provides OS provisions that allow developers to make efficient compute decisions at runtime.
- **Interoperability**: The framework works with OS interfaces and assets such as image primitives from cameras, photos, and video, and it can be used in conjunction with the Windows Machine Learning APIs.
- **NuGet packages**: Windows Vision Skills is strongly versioned to ease iteration without breaking existing applications. They are easy to ingest, easy to update, and they preserve intellectual property through licensing.
- **Extensibility**: The framework can be easily extended to work with existing machine learning frameworks and libraries such as **OpenCV**.
- **Modularity**: Skills can be pieced together in succession within an application just like a recipe to address a complex scenario. Skills can also be bundled together in a single package by the developer.

While this preview focuses on vision-oriented scenarios and primitives, the API is meant to accommodate a wide range of input and output variables that enable audio processing, text processing, and more.

See also

- Windows Vision Skills NuGet packages
- Samples on GitHub
- API reference

NOTE

Use the following resources for help with Windows Vision Skills:

- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Tutorial: Create your own Windows Vision Skill (C#)

9/14/2020 • 13 minutes to read • Edit Online

NOTE

Some information relates to pre-released products, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

If you already have a custom vision solution, this tutorial shows how to wrap the solution in a Windows Vision Skill by extending the Microsoft.AI.Skills.SkillInterfacePreview base API.

Let's build a face sentiment analyzer skill that leverages the following:

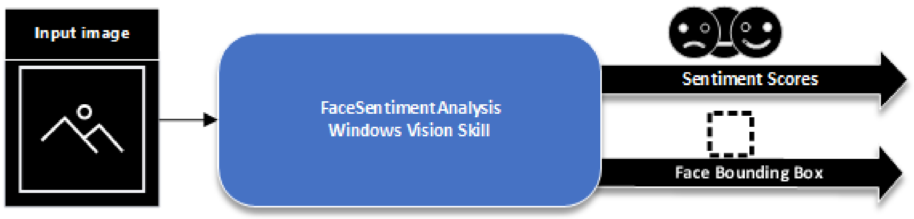
- The Windows.Media.FaceAnalysis and Windows.AI.MachineLearning APIs
- A machine learning model in ONNX format that infers sentiment from face images

This skill takes:

- An input image

And it outputs:

- A tensor of single precision score values in the range [0,1] for each sentiment it evaluates
- A tensor of float values in the range [0,1] defining a face bounding box relative coordinates: left (x,y), top (x,y), right (x,y), and bottom (x,y)



The full source code for the C# and C++/WinRT versions of this example is available on the sample GitHub repository:

- C# example skill
- C++/WinRT example skill

This tutorial will walk you through:

1. Implementing the main interfaces required for a Windows Vision Skill
2. Creating a .nuspec to produce a NuGet package
3. Obfuscating and deobfuscating files to conceal their content

Prerequisites

- Visual Studio 2019 (or Visual Studio 2017, version 15.7.4 or later)
- Windows 10, version 1809 or later
- The Windows 10 SDK, version 1809 or later
- The Microsoft.AI.Skills.SkillInterfacePreview NuGet package

1. Create and implement the main skill classes

First, we need to implement the main skill classes (see Important API concepts for more information):

- ISkillDescriptor
- ISkillBinding
- ISkill

Open up your custom vision solution in Visual Studio.

a. ISkillDescriptor

Create and implement a skill descriptor class inherited from ISkillDescriptor that provides information on the skill, provides a list of supported execution devices (CPU, GPU, and so on), and acts as a factory object for the skill.

1. Import the Microsoft.AI.Skills.SkillInterfacePreview namespace and derive your class from the ISkillDescriptor interface.

```
...
using Microsoft.AI.Skills.SkillInterfacePreview;
...

public sealed class FaceSentimentAnalyzerDescriptor : ISkillDescriptor
{
    ...
}
```

2. Create two member variables that will hold the descriptions of the inputs and outputs required by the skill. Then, in the descriptor's constructor, fill them accordingly with input and output feature descriptors. Additionally, create the *SkillInformation* object that provides all the necessary description properties of your skill.

```
...
// Member variables to hold input and output descriptions
private List<ISkillFeatureDescriptor> m_inputSkillDesc;
private List<ISkillFeatureDescriptor> m_outputSkillDesc;

// Properties required by the interface
public IReadOnlyList<ISkillFeatureDescriptor> InputFeatureDescriptors => m_inputSkillDesc;
public IReadOnlyDictionary<string, string> Metadata => null;
public IReadOnlyList<ISkillFeatureDescriptor> OutputFeatureDescriptors => m_outputSkillDesc;

// Constructor
public FaceSentimentAnalyzerDescriptor()
{
    Information = SkillInformation.Create(
            "FaceSentimentAnalyzer", // Name
            "Finds a face in the image and infers its predominant sentiment from a set of 8 possible
labels", // Description
            new Guid(0xf8d275ce, 0xc244, 0x4e71, 0x8a, 0x39, 0x57, 0x33, 0x5d, 0x29, 0x13, 0x88), // Id
            new Windows.ApplicationModel.PackageVersion() { Major = 0, Minor = 0, Build = 0, Revision =
8 }, // Version
            "Contoso Developer", // Author
            "Contoso Publishing" // Publisher
        );

    // Describe input feature
    m_inputSkillDesc = new List<ISkillFeatureDescriptor>();
    m_inputSkillDesc.Add(
        new SkillFeatureImageDescriptor(
            "InputImage", // skill input feature name
            "the input image onto which the sentiment analysis runs",
            true, // isRequired (since this is an input, it is required to be bound before the
evaluation occurs)
            -1, // width
            -1, // height
            -1, // maxDimension
            BitmapPixelFormat.Nv12,
            BitmapAlphaMode.Ignore)
    );

    // Describe first output feature
    m_outputSkillDesc = new List<ISkillFeatureDescriptor>();
    m_outputSkillDesc.Add(
        new SkillFeatureTensorDescriptor(
            "FaceRectangle", // skill output feature name
            "a face bounding box in relative coordinates (left, top, right, bottom)",
            false, // isRequired (since this is an output, it automatically get populated after the
evaluation occurs)
            new List<long>() { 4 }, // tensor shape
            SkillElementKind.Float)
        );

    // Describe second output feature
    m_outputSkillDesc.Add(
        new SkillFeatureTensorDescriptor(
            FaceSentimentScores, // skill output feature name
            "the prediction score for each class",
            false, // isRequired (since this is an output, it automatically get populated after the
evaluation occurs)
            new List<long>() { 1, 8 }, // tensor shape
            SkillElementKind.Float)
        );
}
```

3. Implement the required method that looks for available supported execution devices to execute the skill on and returns them to the consumer. In our case we return the CPU and all DirectX devices supporting D3D

version 12 or above.

```
public IAsyncOperation<IReadOnlyList<ISkillExecutionDevice>> GetSupportedExecutionDevicesAsync()
{
    return AsyncInfo.Run(async (token) =>
    {
        return await Task.Run(() =>
        {
            var result = new List<ISkillExecutionDevice>();

            // Add CPU as supported device
            result.Add(SkillExecutionDeviceCPU.Create());

            // Retrieve a list of DirectX devices available on the system and filter them by keeping
only the ones that support DX12+ feature level
            var devices = SkillExecutionDeviceDirectX.GetAvailableDirectXExecutionDevices();
            var compatibleDevices = devices.Where((device) => (device as
SkillExecutionDeviceDirectX).MaxSupportedFeatureLevel >= D3DFeatureLevelKind.D3D_FEATURE_LEVEL_12_0);
            result.AddRange(compatibleDevices);

            return result as IReadOnlyList<ISkillExecutionDevice>;
        });
    });
```

4. Implement the required methods for instantiating your skill.

- One of them selects the best available device:

```
public IAsyncOperation<ISkill> CreateSkillAsync()
{
    return AsyncInfo.Run(async (token) =>
    {
        // Retrieve the available execution devices
        var supportedDevices = await GetSupportedExecutionDevicesAsync();
        ISkillExecutionDevice deviceToUse = supportedDevices.First();

        // Either use the first device returned (CPU) or the highest performing GPU
        int powerIndex = int.MaxValue;
        foreach (var device in supportedDevices)
        {
            if (device.ExecutionDeviceKind == SkillExecutionDeviceKind.Gpu)
            {
                var directXDevice = device as SkillExecutionDeviceDirectX;
                if (directXDevice.HighPerformanceIndex < powerIndex)
                {
                    deviceToUse = device;
                    powerIndex = directXDevice.HighPerformanceIndex;
                }
            }
        }
        return await CreateSkillAsync(deviceToUse);
    });
}
```

- The other uses the specified execution device:

```
public IAsyncOperation<ISkill> CreateSkillAsync(ISkillExecutionDevice executionDevice)
{
    return AsyncInfo.Run(async (token) =>
    {
        // Create a skill instance with the executionDevice supplied
        var skillInstance = await FaceSentimentAnalyzerSkill.CreateAsync(this, executionDevice);

        return skillInstance as ISkill;
    });
}
```

b. ISkillBinding

Create and implement a skill binding class inherited from ISkillBinding interface that contains input and output variables consumed and produced by the skill.

1. Import the Microsoft.AI.Skills.SkillInterfacePreview namespace and derive your class from the ISkillBinding interface and its required collection type.

```
...
using Microsoft.AI.Skills.SkillInterfacePreview;
...

public sealed class FaceSentimentAnalyzerBinding : IReadOnlyDictionary<string, ISkillFeature>,
ISkillBinding
{
    ...
```

2. First Create two member variables:

- One is a helper class VisionSkillBindingHelper provided in the base interface to hold an input image feature named "InputImage".

```
private VisionSkillBindingHelper m_bindingHelper = null;
```

- The other holds the LearningModelBinding used to pass to our LearningModelSession specified as argument to our constructor later on in the skill class.

```
// WinML related member variables
internal LearningModelBinding m_winmlBinding = null;
```

Declare the required properties:

```
// ISkillBinding
public ISkillExecutionDevice Device => m_bindingHelper.Device;

// IReadOnlyDictionary
public bool ContainsKey(string key) => m_bindingHelper.ContainsKey(key);
public bool TryGetValue(string key, out ISkillFeature value) => m_bindingHelper.TryGetValue(key, out
value);
public ISkillFeature this[string key] => m_bindingHelper[key];
public IEnumerable<string> Keys => m_bindingHelper.Keys;
public IEnumerable<ISkillFeature> Values => m_bindingHelper.Values;
public int Count => m_bindingHelper.Count;
public IEnumerator<KeyValuePair<string, ISkillFeature>> GetEnumerator() =>
m_bindingHelper.AsEnumerable().GetEnumerator();
IEnumerator IEnumerable.GetEnumerator() => m_bindingHelper.AsEnumerable().GetEnumerator();
```

And implement the constructor. Note that the constructor is *internal*; in our paradigm, ISkillBinding instances are created by the skill and therefore should not expose a standalone constructor:

```
 // Constructor
internal FaceSentimentAnalyzerBinding(
    ISkillDescriptor descriptor,
    ISkillExecutionDevice device,
    LearningModelSession session)
{
    m_bindingHelper = new VisionSkillBindingHelper(descriptor, device);

    // Create WinML binding
    m_winmlBinding = new LearningModelBinding(session);
}
```

3. Create an enum that facilitates reading the sentiment types output by the skill

```
/// Defines the set of possible emotion label scored by this skill
public enum SentimentType
{
    neutral = 0,
    happiness,
    surprise,
    sadness,
    anger,
    disgust,
    fear,
    contempt
};
```

4. Implement optional additional methods that ease get and set operations onto the binding

```
/// Returns whether or not a face is found given the bound outputs
public bool IsFaceFound
{
    get
    {
        ISkillFeature feature = null;
        if (m_bindingHelper.TryGetValue(FaceSentimentAnalyzerConst.SKILL_OUTPUTNAME_FACERECTANGLE, out
feature))
        {
            var faceRect = (feature.FeatureValue as SkillFeatureTensorFloatValue).GetAsVectorView();
            return !(faceRect[0] == 0.0f &&
                faceRect[1] == 0.0f &&
                faceRect[2] == 0.0f &&
                faceRect[3] == 0.0f);
        }
        else
        {
            return false;
        }
    }
}

/// Returns the sentiment with the highest score
public SentimentType PredominantSentiment
{
    get
    {
        SentimentType predominantSentiment = SentimentType.neutral;
        ISkillFeature feature = null;
        if (m_bindingHelper.TryGetValue(FaceSentimentAnalyzerConst.SKILL_OUTPUTNAME_FACESENTIMENTSCORES,
out feature))
        {
            var faceSentimentScores = (feature.FeatureValue as
SkillFeatureTensorFloatValue).GetAsVectorView();

            float maxScore = float.MinValue;
            for (int i = 0; i < faceSentimentScores.Count; i++)
            {
                if (faceSentimentScores[i] > maxScore)
                {
                    predominantSentiment = (SentimentType)i;
                    maxScore = faceSentimentScores[i];
                }
            }
        }

        return predominantSentiment;
    }
}

/// Returns the face rectangle
public IReadOnlyList<float> FaceRectangle
{
    get
    {
        ISkillFeature feature = null;
        if (m_bindingHelper.TryGetValue(FaceSentimentAnalyzerConst.SKILL_OUTPUTNAME_FACERECTANGLE, out
feature))
        {
            return (feature.FeatureValue as SkillFeatureTensorFloatValue).GetAsVectorView();
        }
        else
        {
            return null;
        }
    }
}
```

c. ISkill

Create and implement a skill class inherited from ISkill interface that executes the skill logic and produces output given a set of input. It also acts as a factory object for the ISkillBinding derivative.

1. Import the Microsoft.AI.Skills.SkillInterfacePreview namespace and derive your class from the ISkill interface.

```
...
using Microsoft.AI.Skills.SkillInterfacePreview;
...

public sealed class FaceSentimentAnalyzerSkill : ISkill
{
    ...
```

2. First Create two member variables:

- One to hold a FaceDetector to find a face on the input image.

```
private FaceDetector m_faceDetector = null;
```

- Another to hold the LearningModelSession used to evaluate the sentiment analysis model:

```
private LearningModelSession m_winmlSession = null;
```

Declare the required properties:

```
public ISkillDescriptor SkillDescriptor { get; private set; }
public ISkillExecutionDevice Device { get; private set; }
```

And implement the constructor and static factory method. Note that the constructor is *private* and the factory method is *internal*; in our paradigm, ISkill instances are created by the skill descriptor and therefore should not expose a standalone constructor:

```
 // Constructor
private FaceSentimentAnalyzerSkill(
        ISkillDescriptor description,
        ISkillExecutionDevice device)
{
    SkillDescriptor = description;
    Device = device;
}

// ISkill Factory method
internal static IAsyncOperation<FaceSentimentAnalyzerSkill> CreateAsync(
    ISkillDescriptor descriptor,
    ISkillExecutionDevice device)
{
    return AsyncInfo.Run(async (token) =>
    {
        // Create instance
        var skillInstance = new FaceSentimentAnalyzerSkill(descriptor, device);

        // Instantiate the FaceDetector
        skillInstance.m_faceDetector = await FaceDetector.CreateAsync();

        // Load ONNX model and instantiate LearningModel
        var modelFile = await StorageFile.GetFileFromApplicationUriAsync(new Uri($"ms-
appx:///Contoso.FaceSentimentAnalyzer/emotion_ferplus.onnx"));
        var winmlModel = LearningModel.LoadFromFilePath(modelFile.Path);

        // Create LearningModelSession
        skillInstance.m_winmlSession = new LearningModelSession(winmlModel, GetWinMLDevice(device));

        return skillInstance;
    });
}
```

3. Then implement the ISkillBinding factory method:

```
// ISkillBinding Factory method
public IAsyncOperation<ISkillBinding> CreateSkillBindingAsync()
{
    return AsyncInfo.Run((token) =>
    {
        var completedTask = new TaskCompletionSource<ISkillBinding>();
        completedTask.SetResult(new FaceSentimentAnalyzerBinding(SkillDescriptor, Device,
m_winmlSession));
        return completedTask.Task;
    });
}
```

4. All that remains to be implemented now is the core logic of the skill via the EvaluateAsync() method declared in the base interface. We first do some sanity check and retrieve the output features to populate.

```
// Skill core logic
public IAsyncAction EvaluateAsync(ISkillBinding binding)
{
    FaceSentimentAnalyzerBinding bindingObj = binding as FaceSentimentAnalyzerBinding;
    if (bindingObj == null)
    {
        throw new ArgumentException("Invalid ISkillBinding parameter: This skill handles evaluation of 
FaceSentimentAnalyzerBinding instances only");
    }

    return AsyncInfo.Run(async (token) =>
    {
        // Retrieve input frame from the binding object
        VideoFrame inputFrame = (binding[FaceSentimentAnalyzerConst.SKILL_INPUTNAME_IMAGE].FeatureValue 
as SkillFeatureImageValue).VideoFrame;
        SoftwareBitmap softwareBitmapInput = inputFrame.SoftwareBitmap;

        // Retrieve a SoftwareBitmap to run face detection
        if (softwareBitmapInput == null)
        {
            if (inputFrame.Direct3DSurface == null)
            {
                throw (new ArgumentNullException("An invalid input frame has been bound"));
            }
            softwareBitmapInput = await 
SoftwareBitmap.CreateCopyFromSurfaceAsync(inputFrame.Direct3DSurface);
        }

        // Retrieve face rectangle output feature from the binding object
        var faceRectangleFeature = binding[FaceSentimentAnalyzerConst.SKILL_OUTPUTNAME_FACERECTANGLE];

        // Retrieve face sentiment scores output feature from the binding object
        var faceSentimentScores = 
binding[FaceSentimentAnalyzerConst.SKILL_OUTPUTNAME_FACESENTIMENTSCORES];
        ...
```

Then this particular skill proceeds in 2 steps:

- **Step 1**: Run FaceDetector against the image and retrieve the face bounding box.

```
        ...
        // Run face detection and retrieve face detection result
        var faceDetectionResult = await m_faceDetector.DetectFacesAsync(softwareBitmapInput);
        ...
```

- **Step 2**: If a face was detected, adjust the bounding box, normalize its coordinate for ease of use and proceed with sentiment analysis of that portion of the image using Windows.AI.MachineLearning. Once inference is done, then update the score of each possible sentiment returned as result.

```
        ...
        // If a face is found, update face rectangle feature
        if (faceDetectionResult.Count > 0)
        {
            // Retrieve the face bound and enlarge it by a factor of 1.5x while also ensuring clamping
to frame dimensions
            BitmapBounds faceBound = faceDetectionResult[0].FaceBox;
            var additionalOffset = faceBound.Width / 2;
            faceBound.X = Math.Max(0, faceBound.X - additionalOffset);
            faceBound.Y = Math.Max(0, faceBound.Y - additionalOffset);
            faceBound.Width = (uint)Math.Min(faceBound.Width + 2 * additionalOffset,
softwareBitmapInput.PixelWidth - faceBound.X);
            faceBound.Height = (uint)Math.Min(faceBound.Height + 2 * additionalOffset,
softwareBitmapInput.PixelHeight - faceBound.Y);

            // Set the face rectangle SkillFeatureValue in the skill binding object
            // note that values are in normalized coordinates between [0, 1] for ease of use
            await faceRectangleFeature.SetFeatureValueAsync(
                new List<float>()
                {
                        (float)faceBound.X / softwareBitmapInput.PixelWidth, // left
                        (float)faceBound.Y / softwareBitmapInput.PixelHeight, // top
                        (float)(faceBound.X + faceBound.Width) / softwareBitmapInput.PixelWidth, //
right
                        (float)(faceBound.Y + faceBound.Height) / softwareBitmapInput.PixelHeight //
bottom
                });

            // Bind the WinML input frame with the adequate face bounds specified as metadata
            bindingObj.m_winmlBinding.Bind(
                "Input3", // WinML input feature name defined in ONNX protobuf
                inputFrame, // VideoFrame
                new PropertySet() // VideoFrame bounds
                {
                    { "BitmapBounds",
                        PropertyValue.CreateUInt32Array(new uint[]{ faceBound.X, faceBound.Y,
faceBound.Width, faceBound.Height })
                    }
                });

            // Run WinML evaluation
            var winMLEvaluationResult = await m_winmlSession.EvaluateAsync(bindingObj.m_winmlBinding,
"");

            // Retrieve result using the WinML output feature name defined in ONNX protobuf
            var winMLModelResult = (winMLEvaluationResult.Outputs["Plus692_Output_0"] as
TensorFloat).GetAsVectorView();

            // Set the SkillFeatureValue in the skill binding object related to the face sentiment
scores for each possible SentimentType
            // note that we SoftMax the output of WinML to give a score normalized between [0, 1] for
ease of use
            var predictionScores = SoftMax(winMLModelResult);
            await faceSentimentScores.SetFeatureValueAsync(predictionScores);
        }
        else // if no face found, reset output SkillFeatureValues with 0s
        {
            await
faceRectangleFeature.SetFeatureValueAsync(FaceSentimentAnalyzerConst.ZeroFaceRectangleCoordinates);
            await
faceSentimentScores.SetFeatureValueAsync(FaceSentimentAnalyzerConst.ZeroFaceSentimentScores);
        }
    });
}
```

2. Package your skill to NuGet

All is left is to compile your skill and create a NuGet package out of your skill so that an application can ingest it.

(*Learn more about NuGet packages here*)

To create a NuGet package, you need to write a *.nuspec* file like the one below see original file in Git repo. This file is composed of two main sections:

- **metadata**: This portion contains name, description, author and owner, license and dependencies. Note that in our case, we depend on the Microsoft.AI.Skills.SkillInterfacePreview NuGet package. This NuGet package also links to a license and triggers a request for its approval before ingestion.
- **files**: This portion points to your compiled bits and assets. Note that the target location points to the framework version uap10.0.17763. This ensures that apps ingesting your package that target an earlier version than 10.0.17763 (the minimum OS version this skill requires) will receive an error message.

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
    <metadata>
        <!-- Required elements-->
        <id>Contoso.FaceSentimentAnalyzer_CS</id>
        <title>Contoso.FaceSentimentAnalyzer_CS</title>
        <version>0.0.0.5</version>
        <description>Face Sentiment Analyzer skill sample that extends the
Microsoft.AI.Skills.SkillInterfacePreview APIs</description>
        <authors>Contoso</authors>
        <owners>Contoso</owners>
        <copyright>Copyright (c) Microsoft Corporation.  All rights reserved.</copyright>
        <requireLicenseAcceptance>true</requireLicenseAcceptance>

<licenseUrl>https://github.com/Microsoft/WindowsVisionSkillsPreview/blob/master/license/doc/LICENSE_package.md
</licenseUrl>
        <projectUrl>https://github.com/Microsoft/WindowsVisionSkillsPreview</projectUrl>
        <iconUrl>https://github.com/Microsoft/WindowsVisionSkillsPreview/blob/master/doc/Logo.png?
raw=true</iconUrl>
        <releaseNotes>v0.0.0.5 release
https://github.com/Microsoft/WindowsVisionSkillsPreview/releases</releaseNotes>
        <dependencies>
            <dependency id="Microsoft.AI.Skills.SkillInterfacePreview" version="0.5.2.12" />
        </dependencies>
        <tags>ComputerVision AI VisionSkill</tags>
    </metadata>

    <files>
        <!-- WinMD, Intellisense and resource files-->
        <file src="..\common\emotion_ferplus.onnx" target="lib\uap10.0.17763\Contoso.FaceSentimentAnalyzer" />
        <file src="..\cs\FaceSentimentAnalyzer\bin\Release\Contoso.FaceSentimentAnalyzer.winmd"
target="lib\uap10.0.17763" />
        <file src="..\cs\FaceSentimentAnalyzer\bin\Release\Contoso.FaceSentimentAnalyzer.pri"
target="lib\uap10.0.17763" />
        <file src="..\common\Contoso.FaceSentimentAnalyzer.xml" target="lib\uap10.0.17763" />
    </files>
</package>
```

Then you need to pack your *.nuspec* using nuget.exe (download from official site) to produce a *.nupkg* NuGet package file. Open a command line and navigate to the location of nuget.exe, then call:

```
> .\nuget.exe pack <path to your .nuspec>
```

To test your package locally, you can then put this *.nupkg* file in a folder that you set as a NuGet feed in Visual Studio (See how-to here).

Hooray, you've created your first Windows Vision Skill! You can upload the packaged skill to NuGet.org.

3. One more thing.. obfuscating and deobfuscating asset files to conceal your intellectual property

To deter your consumer from tampering with or accessing your skill assets (model files, images, etc.), you can obfuscate files as a pre-build step and deobfuscate files at runtime. The example in our sample GitHub contains implementation of helper classes that leverage Windows.Security.Cryptography to obfuscate files at compile time and deobfuscate them at runtime. Note that this part is shown only in the C++/WinRT version of the example skill to keep the C# version simpler.

- Obfuscation is a pre-build event that you can set your project to execute all the time or execute once and use the output as an asset directly. In this example, we use a dedicated compiled tool (Obfuscator.exe). You have to make sure you compile this tool first before you invoke it as a pre-build event of your skill compile time. Note that since it executes on your development machine at compile time, you can compile it once using any target and platform supported (i.e. in this case *Debug/Win32*).

  You can set this pre-build event in Visual Studio by:

- C++ project: **right click your skill project** -> uncollapse **Build Event** -> select **Pre-Build Event** -> enter the **Command Line**

- C# project: **right click your skill project** -> select **Build Event** -> enter the **Pre-Build event command Line**

  This command: 1: Copies the asset file locally 2: Encrypts the file to a *.crypt* file (can be any extension name you want) using the logic defined in that requires a GUID key 3: Deletes the local file

NOTE

We suggest you modify the encryption logic proposed in the sample to make it unique to your skill.

```
copy $(ProjectDir)..\..\Common\emotion_ferplus.onnx $(ProjectDir) &amp;&amp;
^$(ProjectDir)..\Obfuscator\Win32\Debug\Obfuscator.exe $(ProjectDir)emotion_ferplus.onnx $(ProjectDir)
emotion_ferplus.crypt 678BD455-4190-45D3-B5DA-41543283C092 &amp;&amp; ^del $(ProjectDir)emotion_ferplus.onnx
```

- Deobfuscation is exposed via a simple helper Windows Runtime Component ingested by the skill. It's decryption logic follows the encryption one defined in previous step.

```
// FaceSentimentAnalyzerSkill.cpp
...
#include "winrt/DeobfuscationHelper.h"
...

// ISkill Factory method
Windows::Foundation::IAsyncOperation<winrt::Contoso::FaceSentimentAnalyzer::FaceSentimentAnalyzerSkill>
FaceSentimentAnalyzerSkill::CreateAsync(
    ISkillDescriptor descriptor,
    ISkillExecutionDevice device)
{
    ...

    // Load WinML model
    auto modelFile =
Windows::Storage::StorageFile::GetFileFromApplicationUriAsync(Windows::Foundation::Uri(L"ms-
appx:///Contoso.FaceSentimentAnalyzer/" + WINML_MODEL_FILENAME)).get();

    // Deobfuscate model file and retrieve LearningModel instance
    LearningModel learningModel =
winrt::DeobfuscationHelper::Deobfuscator::DeobfuscateModelAsync(modelFile, descriptor.Id()).get();

    ...
```

NOTE

Use the following resources for help with Windows Vision Skills:

- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Tutorial: Create a Windows Vision Skill UWP application

9/14/2020 • 2 minutes to read • Edit Online

NOTE

Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

In the previous tutorial, we learned how to create and package a Windows Vision Skill. Now, let's learn how to integrate that into a Universal Windows Platform (UWP) application. You can download the complete sample, available on GitHub to see what it looks like when you finish.

Also with relevance to this tutorial, you can find here more information about loading *VideoFrames* from the following sources:

- an existing *SoftwareBitmap*
- an image file
- camera via FrameReader
- camera via Microsoft.Toolkit: CameraPreview, CameraHelper

Prerequisites

- Completing previous tutorial on Creating your own Windows Vision Skill

API flow

We will revisit the API flow described in the Important API concepts page but now with a concrete set of classes in C#. The full sample code is available on GitHub.

We'll cover the lines of code relevant to the Windows Vision Skill API:

- Instantiate the ISkillDescriptor derivative

```
...

// member variable to hold the skill descriptor
private FaceSentimentAnalyzerDescriptor m_skillDescriptor = null;

...

// Instatiate skill descriptor to display details about the skill and populate UI
m_skillDescriptor = new FaceSentimentAnalyzerDescriptor();

...
```

- Query the available execution devices using the ISKillDescriptor instance

```
...

// member variable to hold the available execution devices
private IReadOnlyList<ISkillExecutionDevice> m_availableExecutionDevices = null;

...

m_availableExecutionDevices = await m_skillDescriptor.GetSupportedExecutionDevicesAsync();

...
```

- Instantiate the skill using the ISKillDescriptor instance and the desired ISkillExecutionDevice

```
...

// member variable to hold the skill
private FaceSentimentAnalyzerSkill m_skill = null;

...

// Initialize skill with the selected supported device
m_skill = await
m_skillDescriptor.CreateSkillAsync(m_availableExecutionDevices[UISkillExecutionDevices.SelectedIndex])
as FaceSentimentAnalyzerSkill;

...
```

- Instantiate a skill binding object using the ISKill instance

```
...

// member variable to hold the skill binding
private FaceSentimentAnalyzerBinding m_binding = null;

...

// Instantiate a binding object that will hold the skill's input and output resource
m_binding = await m_skill.CreateSkillBindingAsync() as FaceSentimentAnalyzerBinding;

...
```

- Retrieve your input primitive (*VideoFrame*) and bind it to your binding object by accessing the corresponding ISkillFeature indexed via its name. Note that this skill declares a convenience set method *SetInputImage*

```
...

// Retrieve a VideoFrame from an image file
VideoFrame frame = await LoadVideoFrameFromFilePickedAsync();

...

// Update input image feature
await m_binding.SetInputImageAsync(frame);

...
```

this convenience method could be bypassed and has the same effect as setting the value directly like so:

```
// Update input image feature
await m_binding["InputImage"].SetFeatureValueAsync(frame);
```

- Run your skill over your binding object

```
// Evaluate the binding
await m_skill.EvaluateAsync(m_binding);
```

- Retrieve your output primitives from your binding object. Note that this skill declares a convenience getter property named *IsFaceFound*.

```
// Retrieve results
if (m_binding.IsFaceFound)
{
    IReadOnlyList<float> scores = (m_binding["FaceSentimentScores"].FeatureValue as
SkillFeatureTensorFloatValue).GetAsVectorView();
}
```

This convenience method, as detailed in the previous tutorial and below, has the same effect as obtaining the output feature directly from the binding and validating its values are not zeros.

```
public bool FaceSentimentAnalyzerBinding.IsFaceFound
{
    get
    {
        ISkillFeature feature = null;
        if (m_bindingHelper.TryGetValue("FaceRectangle", out feature))
        {
            var faceRect = (feature.FeatureValue as SkillFeatureTensorFloatValue).GetAsVectorView();
            return !(faceRect[0] == 0.0f &&
                faceRect[1] == 0.0f &&
                faceRect[2] == 0.0f &&
                faceRect[3] == 0.0f);
        }
        else
        {
            return false;
        }
    }
```

NOTE

Use the following resources for help with Windows Vision Skills:

- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Tutorial: Create a vision skill desktop application (C++)

8/29/2019 • 4 minutes to read • Edit Online

NOTE

Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

NOTE

If you are creating a Windows Vision Skill that needs to be used in a non-UWP app (i.e. a Win32 or .NET Core desktop application), you need to ensure the Skill is aware of its runtime environment.

In this tutorial, you'll learn how to:

- Modify the Skill package to be aware of its runtime environment. Specifically, the Skill needs to be aware of whether or not it is running inside a UWP app container.
- Provide manifest and header files necessary for the Skill to function in a non-UWP app.

Prerequisites

- Visual Studio 2019 (or Visual Studio 2017, version 15.7.4 or later)
- Windows 10, version 1809 or later
- Windows 10 SDK, version 1809 or later

1. Make sure that the skill is UWP-container-aware at runtime:

- Detecting the UWP app container runtime environment from inside the skill:

Example code:

```
// helper function to determine if the skill is being called from a UWP app container or not.
bool IsUWPContainer()
{
    HANDLE hProcessToken = INVALID_HANDLE_VALUE;
    HANDLE hProcess;
    hProcess = GetCurrentProcess();

    if (!OpenProcessToken(hProcess, TOKEN_QUERY, &hProcessToken))
    {
        throw winrt::hresult(HRESULT_FROM_WIN32(GetLastError()));
    }

    BOOL bIsAppContainer = false;
    DWORD dwLength;

    if (!GetTokenInformation(hProcessToken, TokenIsAppContainer, &bIsAppContainer, sizeof(bIsAppContainer),
&dwLength))
    {
        // if we were denied token information we are definitely not in an app container.
        bIsAppContainer = false;
    }

    return bIsAppContainer;
}
```

- If the skill is invoked from a UWP app container or UWP packaged container, the file access is limited to app package paths. Hence, any model file or dependencies need to be packaged and loaded from the container paths.

However, if the skill is invoked from a non-container environment like a Win32 C++ or .Net Core 3.0 Desktop app, then the UWP app container environment is not available. Instead, most of the disk access will be available as per the permissions of the desktop app consuming the skill. Therefore, we recommend you keep model files and other resources at the same location as the skill's library (*.dll*) location.

Example code:

```
winrt::Windows::Storage::StorageFile modelFile = nullptr;

// if running from within a UWP app container, access resources using a URI relative to its path
if (IsUWPContainer())
{
    auto modelFile =
Windows::Storage::StorageFile::GetFileFromApplicationUriAsync(Windows::Foundation::Uri(L"ms-
appx:///Contoso.FaceSentimentAnalyzer/" + WINML_MODEL_FILENAME)).get();
}
// If running from a regular app process such as a Desktop app, access resources using the full system path
else
{
    WCHAR DllPath[MAX_PATH] = { 0 };
    GetModuleFileName(NULL, DllPath, _countof(DllPath));
    // Get path of current DLL
    auto file = Windows::Storage::StorageFile::GetFileFromPathAsync(DllPath).get();
    // Use the path of the parent directory to access other resources bundled with the DLL
    auto folder = file.GetParentAsync().get();
    modelFile = folder.GetFileAsync(WINML_MODEL_FILENAME).get();
```

2. Provide (package in Nuget) header files (.h) and *.manifest* files for ease of consumption by Win32 or .Net Core 3.0 app developers.

   2.1. Generate header files (*.h*) for C++ apps. In Visual Studio, select your project. Then:

   - WinRT C++ component: Select Project -> properties -> MIDL -> output -> Header File

- WinRT C# component: Since there is no option to generate header files in C# project, you need first to convert the generated metadata files (*.winmd*) to interface definition files (*.idl*), then convert these *.idl* to header files (*.h*). This can be done using the Visual Studio developer command prompt:
    - Generate *.idl* from *.winmd* using winmdidl.exe

    ```
    > winmdidl <filename.winmd> /utf8 /metadata_dir:<path-to-sdk-unionmetadata> /metadata_dir: <path-to-additional-winmds> /outdir:<output-path>
    ```

    - Generate *.h* from *.idl* using midlrt.exe

    ```
    > midlrt <filename.idl> /metadata_dir  <path-to-sdk-unionmetadata> /ns_prefix
    ```

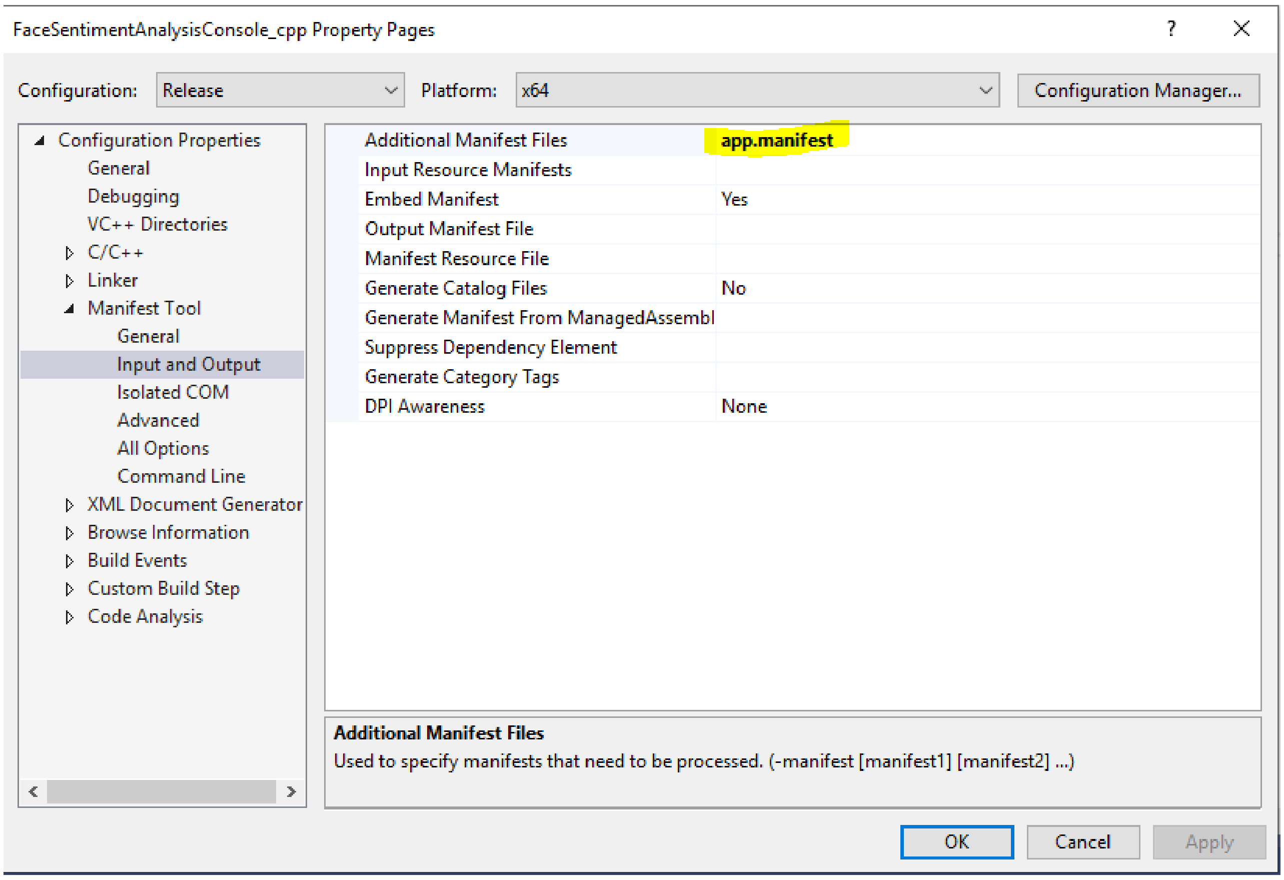
2.2. Create a Manifest file to enable Side-by-Side registration of Skills in non-packaged apps. This file lists the runtime classes defined in your WinRT component, so that they can be registered and loaded at runtime. We provide handy scripts that can create a manifest by parsing your interface definition files (*idl*). Refer to the skill samples for an end-to-end demonstration.

2.2.1. Side by side manifest format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <assembly xmlns="urn:schemas-microsoft-com:asm.v3" manifestVersion="1.0">
        <assemblyIdentity
        type="win32"
        name="manifest Identityname, preferably same as dll name without extension and same as filename of this manifest"
        version="1.0.0.0"/>

        <file name="dll name of the skill component, including extension">

            <activatableClass
            name="runtimeclassname1"
            threadingModel="both"
            xmlns="urn:schemas-microsoft-com:winrt.v1" />

            <activatableClass
            name="runtimeclassname2"
            threadingModel="both"
            xmlns="urn:schemas-microsoft-com:winrt.v1" />

        </file>
    </assembly>
```

2.3. In your app, add a manifest file which mentions the skill manifest you just generated.

2.3.1. App project settings to generate an app side manifest file, and side manifest file format:


FaceSentimentAnalysisConsole_cpp Property Pages

Configuration: Release Platform: x64 Configuration Manager...

Configuration Properties
- General
- Debugging
- VC++ Directories
- C/C++
- Linker
- Manifest Tool
  - General
  - Input and Output
  - Isolated COM
  - Advanced
  - All Options
  - Command Line
- XML Document Generator
- Browse Information
- Build Events
- Custom Build Step
- Code Analysis

Additional Manifest Files	app.manifest
Input Resource Manifests	
Embed Manifest	Yes
Output Manifest File	
Manifest Resource File	
Generate Catalog Files	No
Generate Manifest From ManagedAssembl	
Suppress Dependency Element	
Generate Category Tags	
DPI Awareness	None

Additional Manifest Files
Used to specify manifests that need to be processed. (-manifest [manifest1] [manifest2] ...)

```xml
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
    <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
    <dependency>
        <dependentAssembly>
            <assemblyIdentity
                type="win32"
                name="Microsoft.AI.Skills.SkillInterfacePreview"
                version="1.0.0.0"/>
        </dependentAssembly>
    </dependency>

    <dependency>
        <dependentAssembly>
            <assemblyIdentity
                type="win32"
                name="<name-of-manifest-file-without-extension>"
                version="1.0.0.0"/>
        </dependentAssembly>
    </dependency>

</assembly>
```

Next steps

Hooray, your Skill is now ready to be used in a Desktop Application. Play around with skill samples on GitHub and extend them however you like.

NOTE

Use the following resources for help with Windows Vision Skills:

- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Important API concepts

9/14/2020 • 3 minutes to read • Edit Online

NOTE

Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

The Microsoft.AI.Skills.SkillInterfacePreview namespace provides a set of base interfaces to be extended by all skills as well as classes and helper methods for skill implementers on Windows.

This API is meant to streamline the way skills work and how developers interact with them. The goal is to abstract away the API specificities of each skill's logic and leverage instead a consistent developer intuition over a broad set of solutions and applications. It is also meant to leverage Windows primitives which eases interop with OS APIs. This constitutes a template to derive from that standardizes the developer flow and API interactions across all skills.

Therefore all skills are expected to implement at the very least the following interfaces:

NAME	DESCRIPTION
ISkillDescriptor	Provides information on the skill and exposes its requirements (input, output, version, etc). Acts as a factory for the ISkill.
ISkillBinding	Serves as an indexed container of ISkillFeature. It acts as a conduit for input and output variables to be passed back and forth to the ISkill. It handles pre-processing and post-processing of the input/output data and simplifies their access.
ISkill	Exposes the core logic of processing its input and forming its output via an ISkillBinding. Acts as a factory for the ISkillBinding.

Skills are meant to optimally leverage the hardware capabilities (CPU, GPU, etc.) of each system they run on. These hardware acceleration devices are represented by deriving the ISkillExecutionDevice interface associated with a SkillExecutionDeviceKind. Therefore, ISkillDescriptor derivatives have to find, filter, and expose the set of ISkillExecutionDevice objects currently available on the host system that can execute the ISkill logic.

This will allow the developer consuming the skill package to best chose how to tap into supported hardware resources at runtime on a user's system. There are some ISkillExecutionDevice derivative classes already defined and available for convenience (SkillExecutionDeviceCPU and SkillExecutionDeviceDirectX).

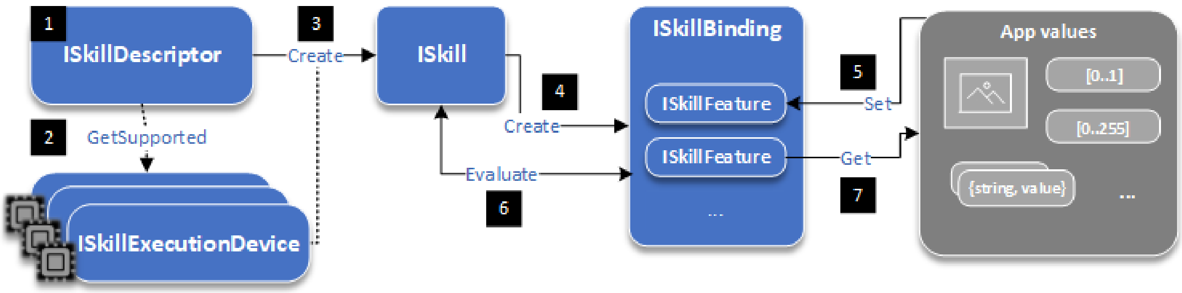
In order to ensure that input and output variables are passed in the correct format to the skill, the ISkillFeature interface is defined. It is meant to encapsulate a value in a predefined format. This value is represented by a derivative of ISkillFeatureValue, which has multiple common derivatives already defined and available for convenience (ISkillFeatureTensorValue, SkillFeatureImageValue, and SkillFeatureMapValue).

Each ISkillFeatureValue derivative has a corresponding ISkillFeatureDescriptor derivative defined that describes the expected format supported by the skill (ISkillFeatureTensorDescriptor, ISkillFeatureImageDescriptor, and ISkillFeatureMapDescriptor). These ISkillFeatureDescriptor derivatives also act as factory objects for the ISkillFeatureValue objects they describe.

At instantiation time, if the primitive used to assign to ISkillFeatureValue does not match the description provided by its ISkillFeatureDescriptor counterpart, an automatic conversion specific to each type can occur seamlessly (for example, transcoding when binding an image in a different format than the one required, or cropping if the aspect ratio differs).

API flow

Since all skills derive the same set of base interfaces from Microsoft.AI.Skills.SkillInterfacePreview, they all follow the same flow which usually boils down to the following operations:



1. Instantiate the ISkillDescriptor derivative.

2. Query the available execution devices using the instance from step 1 (using ISkillDescriptor.GetSupportedExecutionDevicesAsync) or skip and directly attempt step 3.

3. Instantiate the skill using the instance from step 1 and the desired execution device from step 2 (using ISkillDescriptor.CreateSkillAsync(ISkillExecutionDevice)) or the default one decided by the skill developer (using ISkillDescriptor.CreateAsync()).

4. Instantiate a skill binding object using the skill instance from step 3 (using ISkill.CreateSkillBindingAsync).

5. Retrieve your primitives (VideoFrame, IVectorView, IMapView, etc.) from your application and bind them to your binding object from step 4 (by accessing the corresponding ISkillFeature indexed via its name and calling ISkillFeature.SetFeatureValueAsync(Object)).

6. Run your skill over your binding object (by calling ISkill.EvaluateAsync).

7. Retrieve your output primitives from your binding object instantiated in step 4 (by accessing the corresponding ISkillFeature indexed via its name and calling the getter ISkillFeature.FeatureValue).

8. Rinse and repeat from step 5 using the same binding object and new primitives.

To see this in action, refer to the tutorial on ingesting a Windows Vision Skill from an app.

NOTE

Use the following resources for help with Windows Vision Skills:

- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Samples

9/19/2019 • 2 minutes to read • Edit Online

NOTE

Some information relates to pre-released products, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

The Windows Vision Skills GitHub contains sample applications that demonstrate how to use pre-built Windows Vision Skills NuGet packages, as well as tools that help you create your own Windows Vision Skills.

NAME	DESCRIPTION
Sentiment Analyzer (UWP C#/C++, .NET Core, Win32 Desktop C++)	An end-to-end sample that shows how to write a Windows Vision Skill by extending the Microsoft.AI.Skills.SkillInterfacePreview base API.
Object Detector (UWP C#, .NET Core 3.0 C#, Win32 Desktop C++)	A set of sample apps that detect and classify 80 objects using the Object Detector NuGet package.
Skeletal Detector (UWP C#, .NET Core 3.0 C#, Win32 Desktop C++)	A set of sample apps that can detect and identify the poses of the bodies of individuals in a video or image.
Concept Tagger (UWP C#, .NET Core 3.0 C#, Win32 Desktop C++)	A set of sample apps that can identify and classify concepts in an image. This is useful in scenarios like indexing pictures.
Image Scanning (UWP C#, .NET Core 3.0 C#, Win32 Desktop C++)	A set of samples for a bundle of skills aimed at achieving common productivity scenarios focused on scanning the content of an image. These skills provide the *Windows Camera App* and *OfficeLens* scanning experiences and can now empower other Windows applications.

NOTE

Use the following resources for help with Windows Vision Skills:

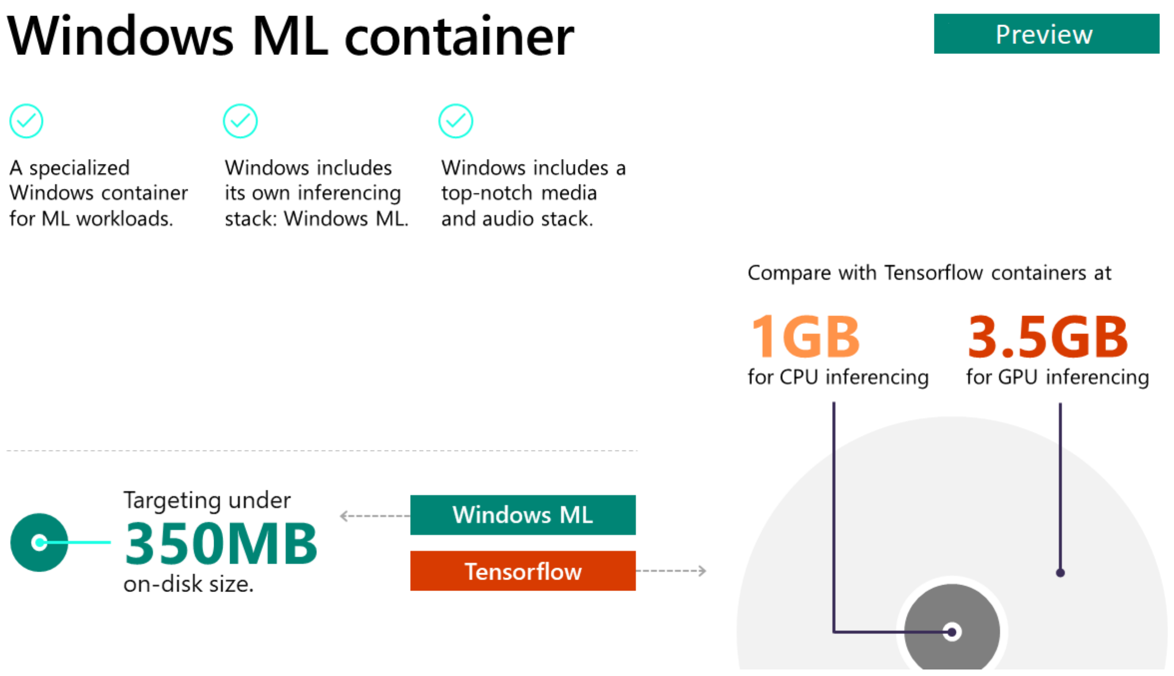
- To report a bug, please file an issue on our GitHub.
- To request a feature, please head over to Windows Developer Feedback.

Windows ML container Insider Preview

10/15/2019 • 2 minutes to read • Edit Online

Windows ML is a high-performance API set, which provides ML inferencing on Windows by consuming standard ONNX models. Often, Machine Learning requires a lot of computing power to use. One of Windows ML's main benefits is that it can use any DirectX12-compliant GPU or Microsoft Compute Driver Model-compliant ASIC to perform hardware accelerated calculations, making it more accessible.

The Windows ML container is a new Windows container that is specifically designed for containerizing workloads that utilize Windows ML APIs. In addition, it also provides direct access to peripheral devices such as sensors or USB cameras on buses such as USB, I2C, SPI, and GPIO. By optimizing the container specifically for those workloads, we're able to reduce the on-disk size of the container to around 350MB. This is significantly less than any other container that has hardware-accelerated ML inferencing capabilities.


Windows ML container
Preview
A specialized Windows container for ML workloads.
Windows includes its own inferencing stack: Windows ML.
Windows includes a top-notch media and audio stack.
Compare with Tensorflow containers at
1GB for CPU inferencing
3.5GB for GPU inferencing
Targeting under 350MB on-disk size.
Windows ML
Tensorflow

Windows ML container gives businesses a fast and agile platform to build enterprise-grade IoT solutions. It combines the advanced features of Windows with the security of the Windows 10 IoT platform and the manageability of the Azure IoT Edge service.

Getting started

Interested in using the Windows ML container? Check out our Getting Started guide for more information.

Supported APIs

Windows ML container supports workloads written in various languages and frameworks, such as C++, C#, .NET Core, Node.JS, and Python. However, due to performance and size optimizations, the OS API surface supported by Windows ML container is smaller than that of the full Windows container. Many User Interface APIs and high level UWP APIs are not available upon workloads running on the container. Supported APIs include those in the OneCore API surface, WinRT contracts for supporting device access, and Windows ML APIs.

For a detailed list of support APIs and contracts, see the list of supported APIs.

Feedback

We want to hear from you!

As always, your feedback is very important to us! Please share your comments and experience with us at winmlcfb@microsoft.com

Getting started

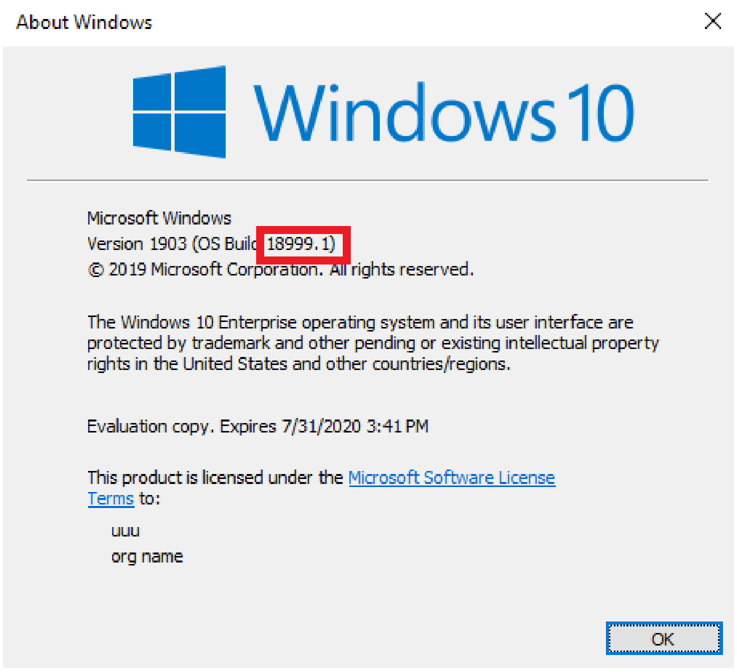
10/16/2019 • 7 minutes to read • Edit Online

The Windows ML container is intended for use by developers who are already familiar with the basics of IoT and Windows ML development. To learn more about using Windows ML, see the Windows ML docs.

Prerequisites and environment setup

To install the Insider version of Windows, you will need to join the Windows Insider Program.

Once you have installed the Insider release of Windows host, you can run `winver` in a command prompt to find the host version.



About Windows

Windows 10

Microsoft Windows
Version 1903 (OS Build 18999.1)
© 2019 Microsoft Corporation. All rights reserved.

The Windows 10 Enterprise operating system and its user interface are protected by trademark and other pending or existing intellectual property rights in the United States and other countries/regions.

Evaluation copy. Expires 7/31/2020 3:41 PM

This product is licensed under the Microsoft Software License Terms to:

uuu
org name

In the example above, the host version is `18999.1`.

Host OS and container version matching requirements

Windows ML container runs on top of the Windows Enterprise host. Your version of the Windows ML container must be an exact match to the version of the **fast ring Windows 10 Insider Preview Build (20H1)**.

Windows ML container on Docker hub

Once you have identified the version of the host, find the matching tag on Docker Hub page for Window ML container Insider. Locate a corresponding version tag and Docker pull URL.

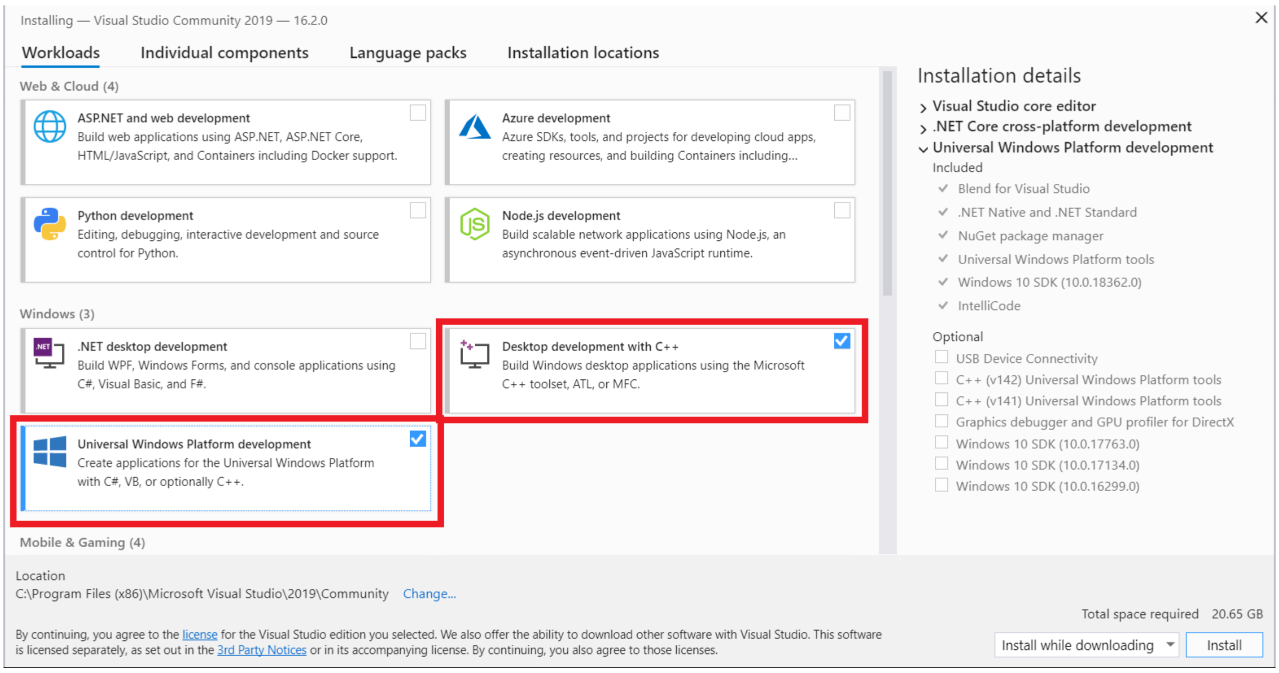
```
10.0.18999.1 (20H1)
docker pull mcr.microsoft.com/windows/ml/insider:10.0.18999.1
```

Visual Studio 2019

We recommend you use Visual Studio 2019 when developing for the Windows ML container. The latest Community edition is available free at the Visual Studio site. If you have not worked with Visual Studio before, follow the instructions and guidance on the site for more information.

When configuring your Visual Studio installation, please ensure you have installed the following packages:

- Universal Windows Platform development
- .NET Core cross-platform development.NET Core 2.2 development tools



Installing — Visual Studio Community 2019 — 16.2.0

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

☐ **ASP.NET and web development** — Build web applications using ASP.NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker support.

☐ **Azure development** — Azure SDKs, tools, and projects for developing cloud apps, creating resources, and building Containers including...

☐ **Python development** — Editing, debugging, interactive development and source control for Python.

☐ **Node.js development** — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Windows (3)

☐ **.NET desktop development** — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.

✓ **Desktop development with C++** — Build Windows desktop applications using the Microsoft C++ toolset, ATL, or MFC.

✓ **Universal Windows Platform development** — Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Mobile & Gaming (4)

Installation details

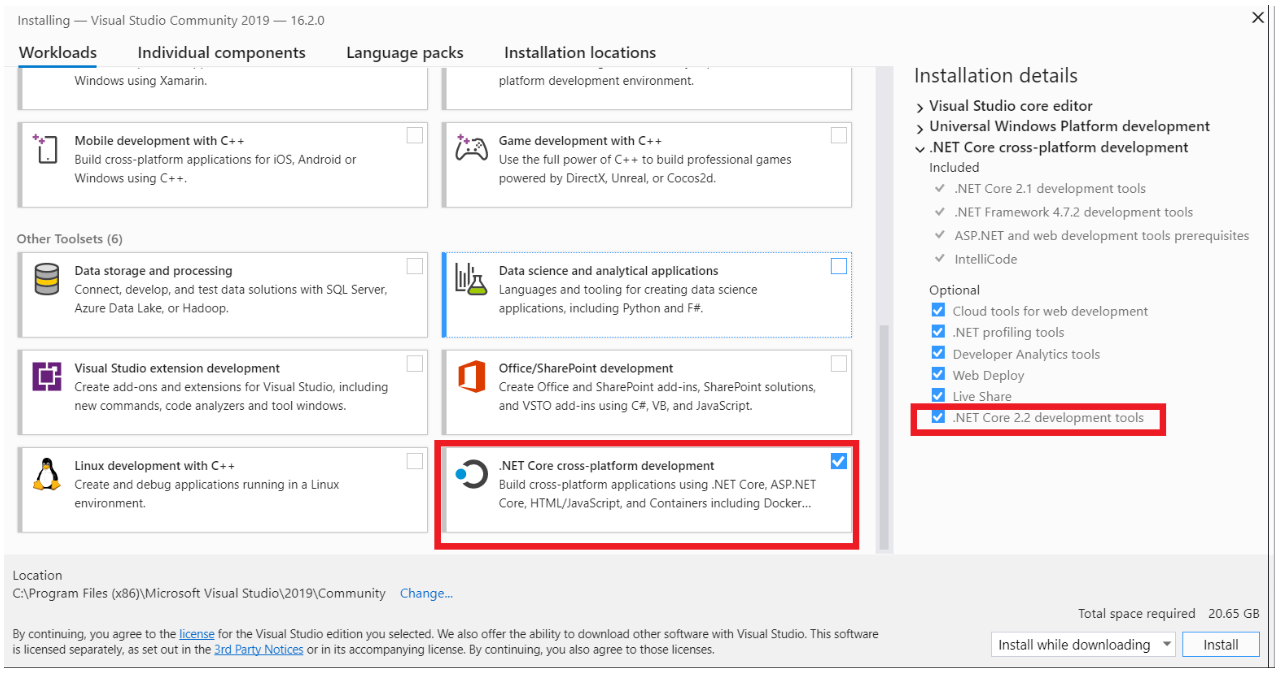
- Visual Studio core editor
- .NET Core cross-platform development
- Universal Windows Platform development
  - Included
    - ✓ Blend for Visual Studio
    - ✓ .NET Native and .NET Standard
    - ✓ NuGet package manager
    - ✓ Universal Windows Platform tools
    - ✓ Windows 10 SDK (10.0.18362.0)
    - ✓ IntelliCode
  - Optional
    - ☐ USB Device Connectivity
    - ☐ C++ (v142) Universal Windows Platform tools
    - ☐ C++ (v141) Universal Windows Platform tools
    - ☐ Graphics debugger and GPU profiler for DirectX
    - ☐ Windows 10 SDK (10.0.17763.0)
    - ☐ Windows 10 SDK (10.0.17134.0)
    - ☐ Windows 10 SDK (10.0.16299.0)

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community Change...

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 20.65 GB

Install while downloading | Install



Installing — Visual Studio Community 2019 — 16.2.0

Workloads | Individual components | Language packs | Installation locations

Windows using Xamarin.

platform development environment.

☐ **Mobile development with C++** — Build cross-platform applications for iOS, Android or Windows using C++.

☐ **Game development with C++** — Use the full power of C++ to build professional games powered by DirectX, Unreal, or Cocos2d.

Other Toolsets (6)

☐ **Data storage and processing** — Connect, develop, and test data solutions with SQL Server, Azure Data Lake, or Hadoop.

☐ **Data science and analytical applications** — Languages and tooling for creating data science applications, including Python and F#.

☐ **Visual Studio extension development** — Create add-ons and extensions for Visual Studio, including new commands, code analyzers and tool windows.

☐ **Office/SharePoint development** — Create Office and SharePoint add-ins, SharePoint solutions, and VSTO add-ins using C#, VB, and JavaScript.

☐ **Linux development with C++** — Create and debug applications running in a Linux environment.

✓ **.NET Core cross-platform development** — Build cross-platform applications using .NET Core, ASP.NET Core, HTML/JavaScript, and Containers including Docker...

Installation details

- Visual Studio core editor
- Universal Windows Platform development
- .NET Core cross-platform development
  - Included
    - ✓ .NET Core 2.1 development tools
    - ✓ .NET Framework 4.7.2 development tools
    - ✓ ASP.NET and web development tools prerequisites
    - ✓ IntelliCode
  - Optional
    - ✓ Cloud tools for web development
    - ✓ .NET profiling tools
    - ✓ Developer Analytics tools
    - ✓ Web Deploy
    - ✓ Live Share
    - ✓ .NET Core 2.2 development tools

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community Change...

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 20.65 GB

Install while downloading | Install

Windows SDK Insider Preview

Ensure that the version of your Windows ML container and your Windows SDK Insider Preview are matched as closely as possible. An exact version match is not required, but the larger the mismatch, the greater the chance of errors or other issues.

The latest Insider SDK is available here.

Configure Visual Studio 2019 NuGet package source

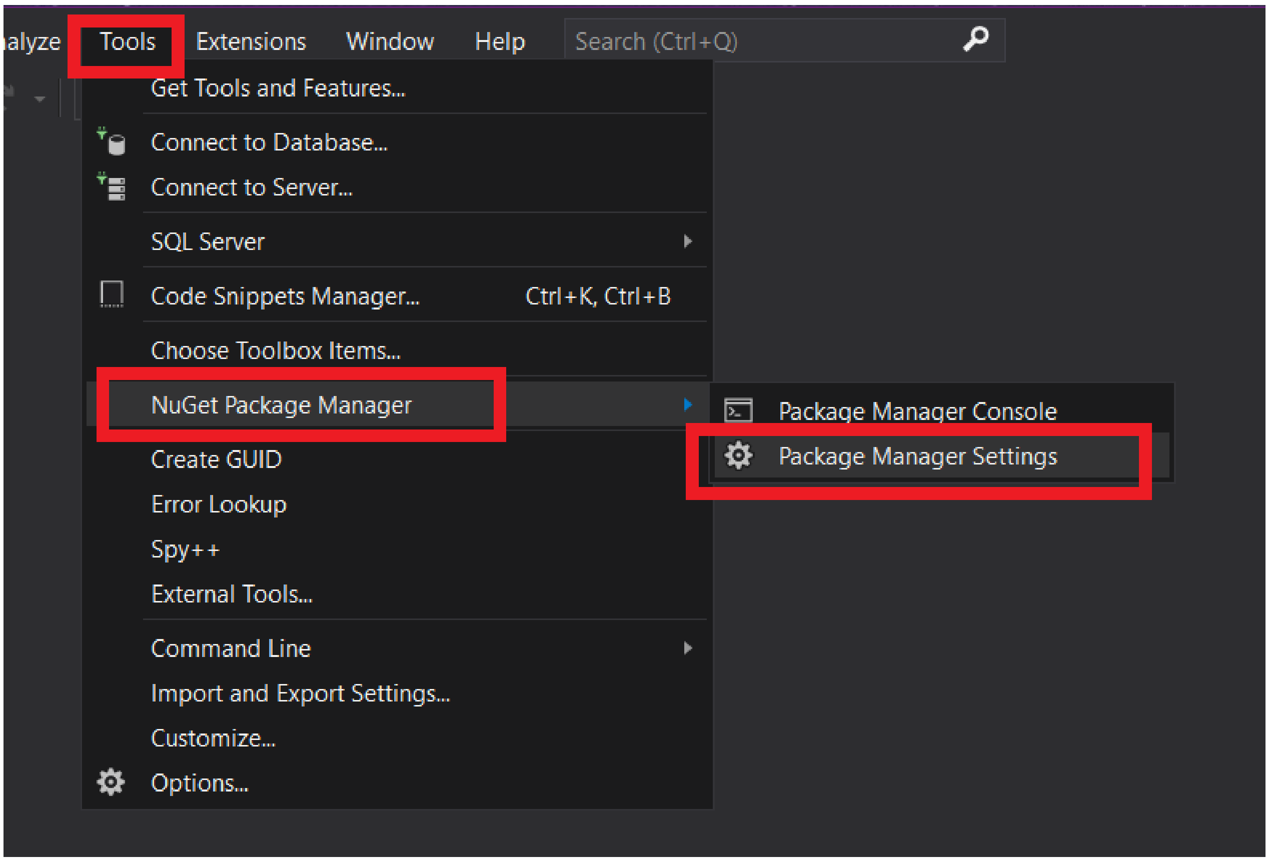
While Windows ML container is in Preview, the Windows Headless WinRT contract is provided via NuGet package from a separate package source.

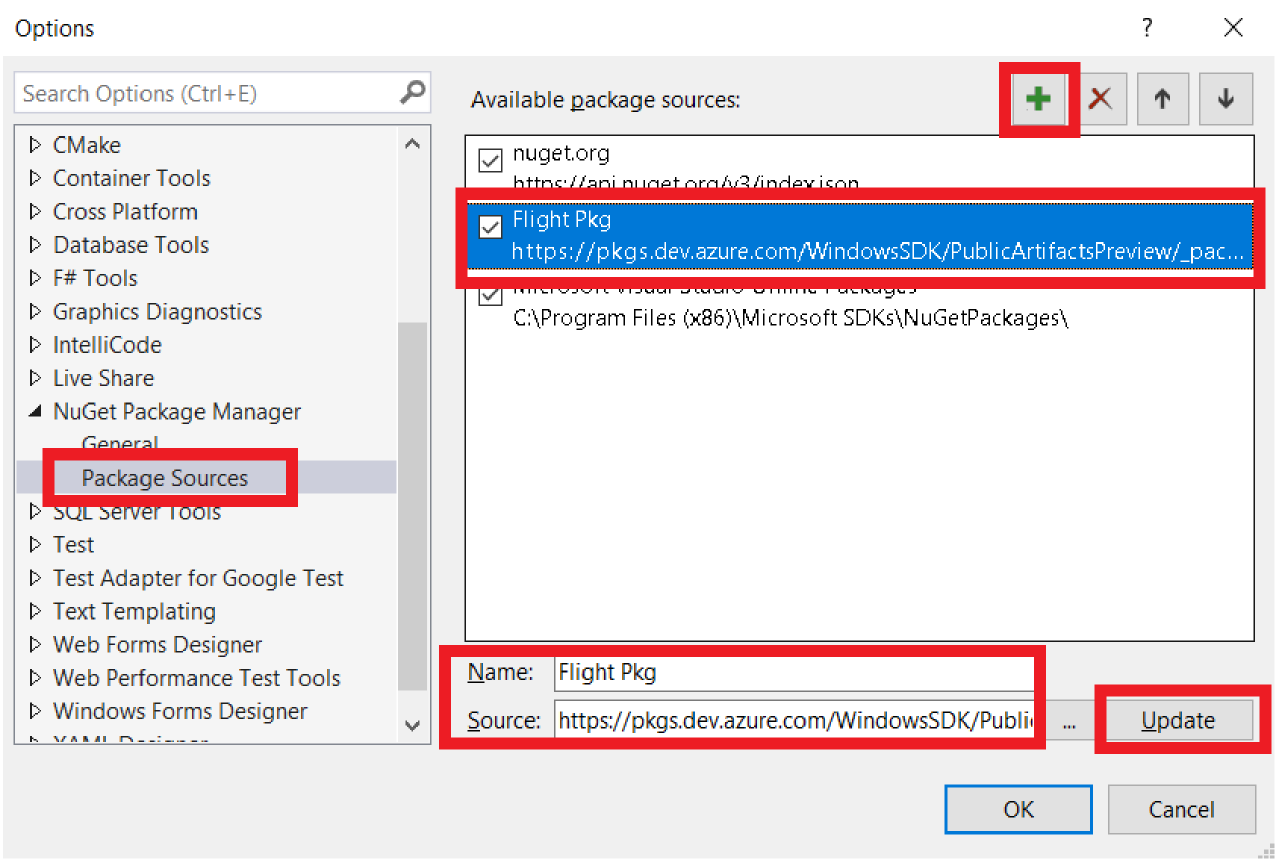
Make sure the following sources are in the package source:

- https://api.nuget.org/v3/index.json
- https://pkgs.dev.azure.com/WindowsSDK/PublicArtifactsPreview/_packaging/WindowsSDKFlight/nuget/v3/index.json

To add the new source, follow these steps:

- Open Visual Studio 2019
- Select **Tools -> NuGet Package Manager -> Package Manager Settings -> Package Sources** and add the package SDK flight source URL.





Available package sources:

- ✓ nuget.org
  https://api.nuget.org/v3/index.json
- ✓ Flight Pkg
  https://pkgs.dev.azure.com/WindowsSDK/PublicArtifactsPreview/_pac...
- ✓ Microsoft Visual Studio Offline Packages
  C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

Name: Flight Pkg

Source: https://pkgs.dev.azure.com/WindowsSDK/Publi

GPU support

For GPU acceleration to be supported by the container, the host must be able to use a GPU and graphics driver that meets these requirements:

GPU requirements

VENDOR	ARCHITECTURE	TYPICAL CUSTOMER-FACING GPU NAMES
AMD	GCN 4 or later	Radeon RX 400 series or later, or Radeon Pro WX series
Intel	Kaby Lake or later	Intel HD Graphics 600 series or later
NVIDIA	Kepler or later	GeForce 600 series or later, or Quadro K-series or later

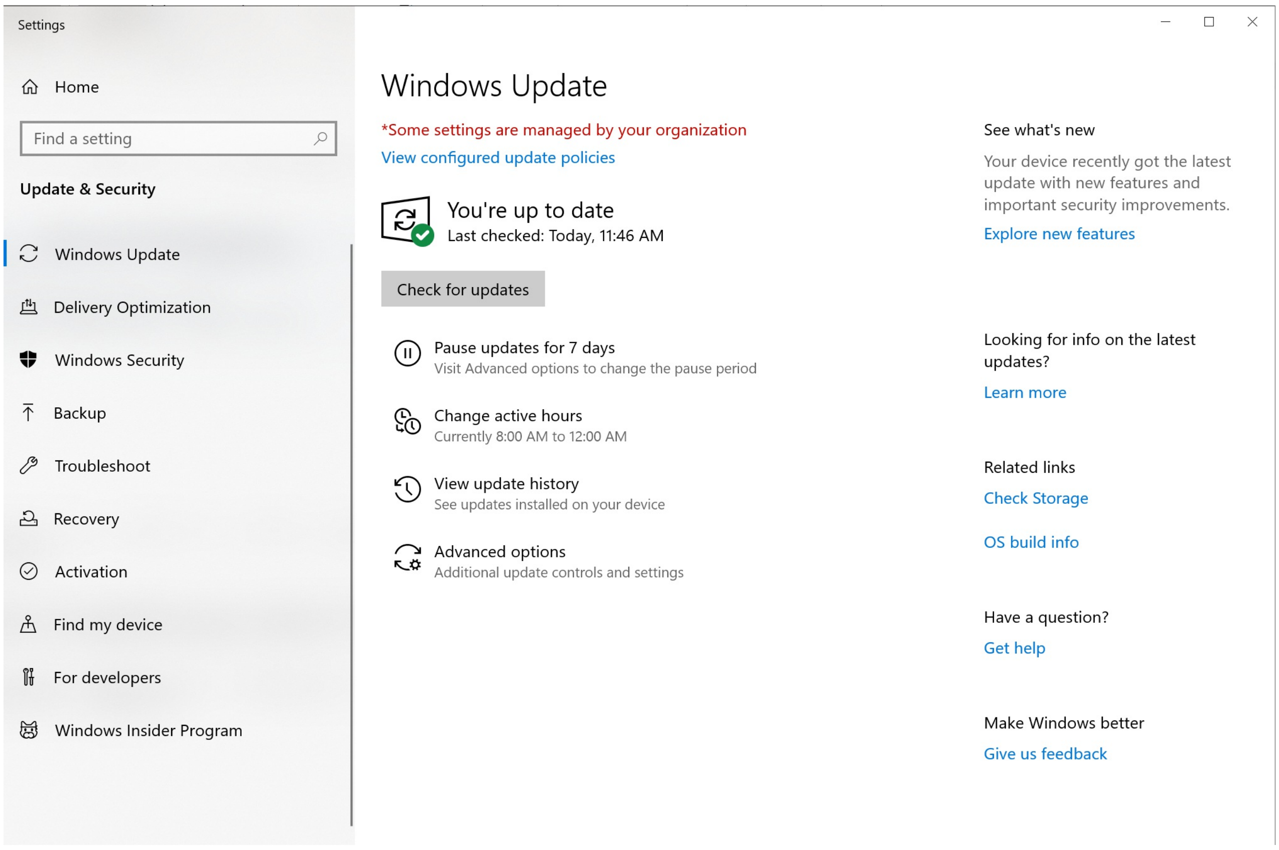
Graphics driver requirements

VENDOR	MIN DRIVER VERSION
AMD	26.20.12002.65
Intel	26.20.100.6812
NVIDIA	26.21.14.3086

If the GPU or driver does not meet the requirements above, or if the container host OS is running in a VM, then only CPU-based inferencing is supported.

Graphics driver installation

Download the most recent graphics drivers for your system by navigating to **Settings > Update & Security > Windows Update > Check for updates**.



Windows Update

*Some settings are managed by your organization

View configured update policies

You're up to date

Last checked: Today, 11:46 AM

Check for updates

Pause updates for 7 days

Visit Advanced options to change the pause period

Change active hours

Currently 8:00 AM to 12:00 AM

View update history

See updates installed on your device

Advanced options

Additional update controls and settings

See what's new

Your device recently got the latest update with new features and important security improvements.

Explore new features

Looking for info on the latest updates?

Learn more

Related links

Check Storage

OS build info

Have a question?

Get help

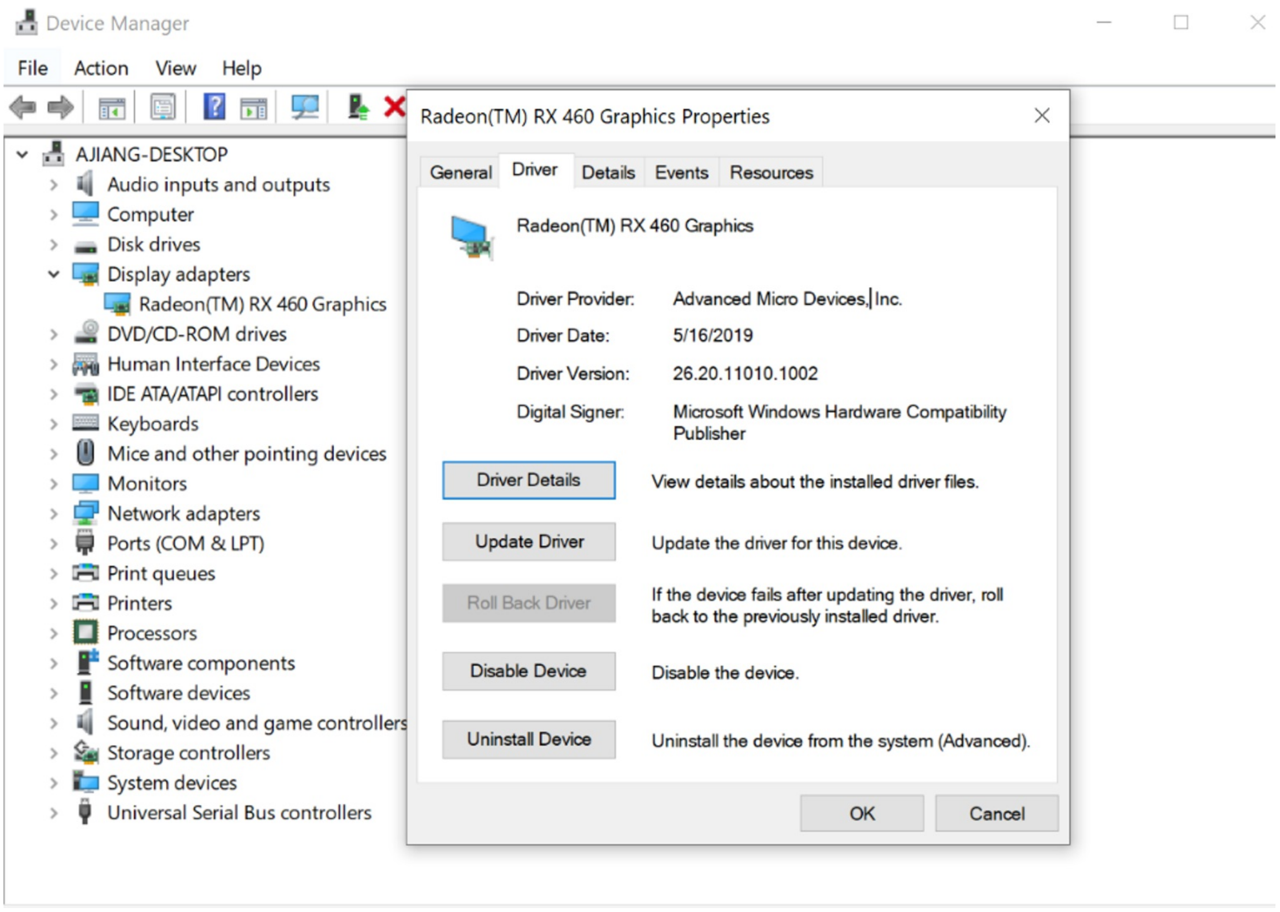
Make Windows better

Give us feedback

Troubleshooting graphics driver

Check that your driver version meets the minimum requirements by navigating to **Device Manager > Display**

Adapters, right-clicking on your graphics adapter and selecting **Properties**, and looking under the **Driver** tab:



Radeon(TM) RX 460 Graphics

Driver Provider: Advanced Micro Devices, Inc.

Driver Date: 5/16/2019

Driver Version: 26.20.11010.1002

Digital Signer: Microsoft Windows Hardware Compatibility Publisher

Driver Details — View details about the installed driver files.

Update Driver — Update the driver for this device.

Roll Back Driver — If the device fails after updating the driver, roll back to the previously installed driver.

Disable Device — Disable the device.

Uninstall Device — Uninstall the device from the system (Advanced).

If you are unable to download a graphics driver through Windows Update that meets the minimum version requirements, please email winmlc-questions@microsoft.com with your system's DxDiag information attached. For instructions on running and saving DxDiag information, see this support page.

Set up and test a basic environment

1. Ensure your host OS and Windows ML container image share the same version number.

2. Enable the **Containers** functionality on the host OS.

   Run the following command in an *elevated* command window. You may be prompted to restart the system.

```
dism /online /Enable-Feature /FeatureName:Containers
```

3. Download the nightly build versions of docker.exe and dockerd.exe.

```
curl.exe -o %windir%\system32\dockerd.exe https://master.dockerproject.org/windows/x86_64/dockerd.exe
```

```
curl.exe -o %windir%\system32\docker.exe https://master.dockerproject.org/windows/x86_64/docker.exe
```

Register and start the Docker service.

```
dockerd.exe --register-service

net start docker
```

You should see the following output message.

```
The Docker Engine service is starting.
The Docker Engine service was started successfully.
```

4. You can confirm that Docker is running correct correctly with the following command.

```
docker version
```

This should produce the following output message.

```
Client:
    Version:           master-dockerproject-2019-08-03
    API version:       1.40
    Go version:        go1.12.7
    Git commit:        e505a7c2
    Built:             Sun Aug  4 00:02:51 2019
    OS/Arch:           windows/amd64
    Experimental:      false

Server:
    Engine:
    Version:          master-dockerproject-2019-08-03
    API version:      1.41 (minimum version 1.24)
    Go version:       go1.12.7
    Git commit:       7449ca3
    Built:            Sun Aug  4 00:12:27 2019
    OS/Arch:          windows/amd64
    Experimental:     false
```

5. Once Docker is started, import the container image file into Docker using the following command.

```
docker pull mcr.microsoft.com/windows/ml/insider:10.0.18999.1
docker tag mcr.microsoft.com/windows/ml/insider:10.0.18999.1 windowsml:latest
```

6. After the image is installed, you can run `docker images` to enumerate all available container images. By default, this image will not have a repository name or tag. You may specify one, or simply use the image ID from the hash provided to reference the image in future steps.

The output may look similar to this.

```
C:\Windows\system32>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              a9d5d08d079f        10 seconds ago      319MB

C:\Windows\system32>docker tag a9d5d08d079f windowsml:latest

C:\Windows\system32>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
windowsml           latest              a9d5d08d079f        25 seconds ago      319MB
```

7. Download WinMLRunner v1.2.1.1 from https://github.com/microsoft/Windows-Machine-Learning/releases/tag/1.2.1.1 with the following command.

```
curl -o WinMLRunner.zip -L https://github.com/microsoft/Windows-Machine-
Learning/releases/download/1.2.1.1/WinMLRunner.v1.2.1.1.zip
```

Then, unzip the .zip into the current folder.

8. Download SqueezeNet.onnx sample from https://github.com/microsoft/Windows-Machine-Learning/tree/1.2.1.1/SharedContent/models with the following command.

```
curl -o SqueezeNet.onnx -L https://github.com/microsoft/Windows-Machine-
Learning/raw/1.2.1.1/SharedContent/models/SqueezeNet.onnx
```

9. Create a Dockerfile to copy the necessary files into the imported Windows ML container image.

You can copy the commands below to directly create a Dockerfile:

```
echo FROM windowsml:latest               >  Dockerfile
echo WORKDIR C:/App                      >> Dockerfile
echo COPY ./x64/WinMLRunner.exe C:/App/  >> Dockerfile
echo COPY ./SqueezeNet.onnx C:/App/      >> Dockerfile
```

```
C:\tgz>type Dockerfile
FROM windowsml:latest
WORKDIR C:/App
COPY ./x64/WinMLRunner.exe C:/App/
COPY ./SqueezeNet.onnx C:/App/
```

10. Build a new container based on the Dockerfile, with the `docker build command`.

```
C:\tgz>docker build -t winmlrunner:latest .
```

Results will be similar to the following.

```
Sending build context to Docker daemon  5.525MB
Step 1/4 : FROM windowsml:latest
 ---> a9d5d08d079f
Step 2/4 : WORKDIR C:/App
 ---> Running in 37e9d759365f
Removing intermediate container 37e9d759365f
 ---> 8ab270ff4deb
Step 3/4 : COPY ./x64/WinMLRunner.exe C:/App/
 ---> 16e2b23a5de0
Step 4/4 : COPY ./SqueezeNet.onnx C:/App/
 ---> 05269bb7c5f8
Successfully built 05269bb7c5f8
Successfully tagged winmlrunner:latest

C:\tgz>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
winmlrunner         latest              ed74203b2655        About a minute ago   325MB
windowsml           latest              a9d5d08d079f        23 minutes ago       319MB
```

11. Launch a GPU-enabled process-isolated container based on the Windows ML container and WinMLRunner image.

```
docker run -it --isolation process --device class/5B45201D-F2F2-4F3B-85BB-30FF1F953599 winmlrunner:latest cmd
/k
```

There are a few important arguments that you must specify for running Windows ML containers using Docker.

- `-it`
  Used to allow interactive shell components to forward `stdin` and use `tty`. If you omit this argument, the command line interface will not function properly.
- `--isolation process` Specifies the type of container as a process-isolated container.
- `--device class/5B45201D-F2F2-4F3B-85BB-30FF1F953599` Specifies the class GUID for the device that should be exposed into the container. `class/5B45201D-F2F2-4F3B-85BB-30FF1F953599` refers to `GUID_DEVINTERFACE_DISPLAY_ADAPTER`, which enables DirectX GPUs.
- `winmlrunner:latest` Specifies the image to run, that was created in previous steps. This can either be the repository name and tag you provided during a `docker tag` command, or the image ID/hash of the image. If you used **latest** as the tag name, it can be omitted.
- `cmd /k`
  This is the command that docker will execute inside the container.

12. From the container command line, run WinMLRunner using CPU.

```
WinMLRunner.exe -model C:/App/SqueezeNet.onnx -cpu
```

Output should appear similar to the following.

```
DXGI module not found.
Loading model (path = C:/App/SqueezeNet.onnx)...
=================================================================
Name: squeezenet_old
Author: onnx-caffe2
Version: 9223372036854775807
Domain:
Description:
Path: C:/App/SqueezeNet.onnx
Support FP16: false

Input Feature Info:
Name: data_0
Feature Kind: Float

Output Feature Info:
Name: softmaxout_1
Feature Kind: Float

=================================================================



Creating Session with CPU device
Binding (device = CPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation =
WinML)...[SUCCESS]
Evaluating (device = CPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation =
WinML)...[SUCCESS]
```

Congratulations, your environment is now set up correctly to use Windows ML container.

13. You can also run WinMLRunner using the GPU. Specify one of AMD Radeon, Nvidia or Intel through the `-GPUAdapterName` command line argument.

```
WinMLRunner.exe -model C:/App/SqueezeNet.onnx -GPUAdapterName [radeon/nvidia/intel]
```

Build Apps the for Windows ML container

Samples for Windows ML container

To get started, make sure your Visual Studio 2019 is set up and configured according to the above instructions. Then, try the following samples:

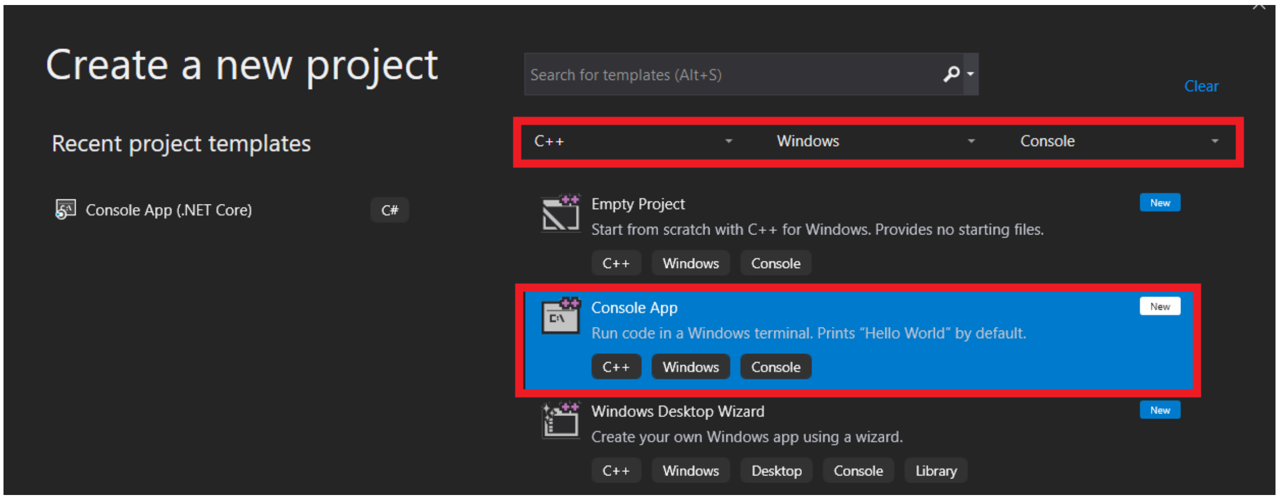
- CustomVision. This sample uses the model trained by Azure Custom Vision Service. The trained model is exported as an ONNX file, and included as part of the sample app that runs inside the container.
- SqueezeNetObjectDetection. This app (cpp and c# only) uses the SqueezeNet model to detect the predominant object in an image.

Create a Visual Studio 2019 C# project from scratch

Windows ML container only supports a subset of Windows APIs due to its small size. When you create a Visual Studio project, you can specify this smaller API surface, to detect unsupported APIs before runtime.

To use the Headless WinRT API contract:

1. In Visual Studio 2019, create a new **C# Console App (.NET Core)** project.



Create a new project

Recent project templates

Console App (.NET Core) C#

Search for templates (Alt+S)

Clear

C++ | Windows | Console

Empty Project — New
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console

Console App — New
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console

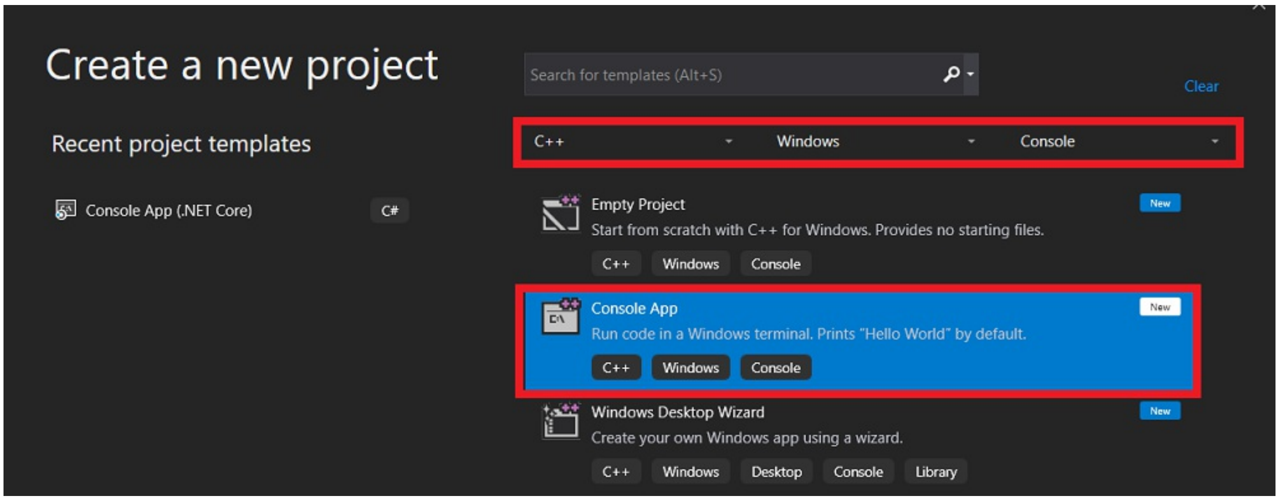
Windows Desktop Wizard — New
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library

1. Select **Tools->Nuget->Package Manager Console**
2. In the Package Manager Console, run:

```
Install-Package Microsoft.Windows.SDK.Headless.Contracts -Prerelease
```

Create a Visual Studio 2019 C++ project from scratch

1. In Visual Studio 2019, create a new **C++ Console App** project.



Create a new project

Recent project templates

Console App (.NET Core) C#

Search for templates (Alt+S)

Clear

C++ | Windows | Console

Empty Project — New
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console

Console App — New
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console

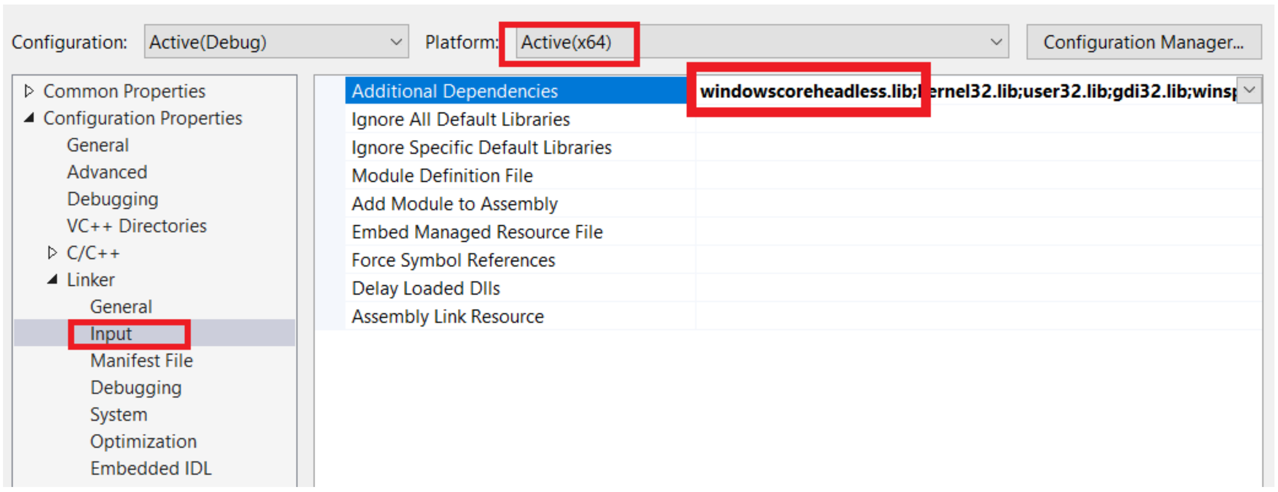
Windows Desktop Wizard — New
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library

1. Select **Tools->Nuget->Package Manager Console**

2. In the Package Manager Console, run:

```
Install-Package Microsoft.Windows.SDK.Headless.Contracts -Prerelease
Install-Package Microsoft.Windows.CppWinRT -Version 2.0.190730.2
```

4. Update project to use `windowscoreheadless.lib`:
   a. Right click on your project.
   b. Choose properties
   c. In the dialog choose Linker->Input
   d. Update Additional Dependencies to include `windowscoreheadless.lib`. For example:
      a. `windowscoreheadless.lib;%(...AdditionalDependencies...)`


Configuration: Active(Debug)
Platform: Active(x64)

Additional Dependencies: windowscoreheadless.lib;kernel32.lib;user32.lib;gdi32.lib;winsp

Use the Windows ML container Insider Preview with Azure IoT Edge Runtime

10/15/2019 • 3 minutes to read • Edit Online

Due to host OS and container version matching requirements, in order to use Azure IoT Edge Runtime with Windows ML container Insider Preview, there are a few steps that need to be taken to prepare Azure IoT Edge runtime environment.

IMPORTANT

The steps below assume some experience and familiarity with using Docker and Azure IoT Edge.

Environment Setup

On a Windows Insider Preview host machine with a matching version to the installed Windows ML container, do the following:

- Install the .NET Core SDK.
- Install Git.
- Install Docker from Docker Master, following the instructions in getting started.
- Install Azure CLI.
- Create a private container registry on Azure Container Registry for storing Azure IoT Edge modules. (Note that It may be easier to login if Admin account is enabled.)
- Create an Azure IoT Hub to manage devices and deployments on the cloud.

Build private Azure IoT Edge runtime module

Prepare base OS image

- Launch CMD or powershell as admin,
- Login to Azure Container Registry from docker.
- Pull Windows ML container base image (*change the tag to match your Insider host version*) with the following commands.
  - `docker pull mcr.microsoft.com/windows/ml/insider:10.0.18999.1`
  - `docker tag mcr.microsoft.com/windows/ml/insider:10.0.18999.1 windowsml:latest`
- Clone Azure IoT Edge into `C:\iotedge`.
  - `cd /d c:\`
  - `git clone https://github.com/Azure/iotedge.git`

Build Edge hub container

Execute the following commands to build your Edge hub container.

- `cd c:\iotedge\edge-hub\src\Microsoft.Azure.Devices.Edge.Hub.Service\`
- `dotnet publish -r win-x64`
- `cd bin\Debug\netcoreapp2.1\win-x64\`
- Create a Dockerfile with content like below following and save it at

`c:\iotedge\edge-hub\src\Microsoft.Azure.Devices.Edge.Hub.Service\bin\Debug\netcoreapp2.1\win-x64`:

```
FROM windowsml:latest
WORKDIR /app
COPY publish/ ./
# Expose MQTT, AMQP and HTTPS ports-
EXPOSE 8883/tcp
EXPOSE 5671/tcp
EXPOSE 443/tcp
CMD ["Microsoft.Azure.Devices.Edge.Hub.Service.exe"]
```

Continue with the following commands.

- `docker build . -t your-own-registry.azurecr.io/hub:latest`
- `docker push your-own-registry.azurecr.io/hub:latest`

Build Agent container

Execute the following commands to build your Agent container.

- `cd c:\iotedge\edge-agent\src\Microsoft.Azure.Devices.Edge.Agent.Service\`
- `dotnet publish -r win-x64`
- `cd bin\Debug\netcoreapp2.1\win-x64`
- Create a Dockerfile with content like the following and save it at: `c:\iotedge\edge-agent\src\Microsoft.Azure.Devices.Edge.Agent.Service\bin\Debug\netcoreapp2.1\win-x64`

```
FROM windowsml:latest
# Configure web servers to bind to port 80 when present
ENV ASPNETCORE_URLS=http://+:80
WORKDIR /app
COPY publish/ ./
CMD ["Microsoft.Azure.Devices.Edge.Agent.Service.exe"]
```

Continue with the following commands.

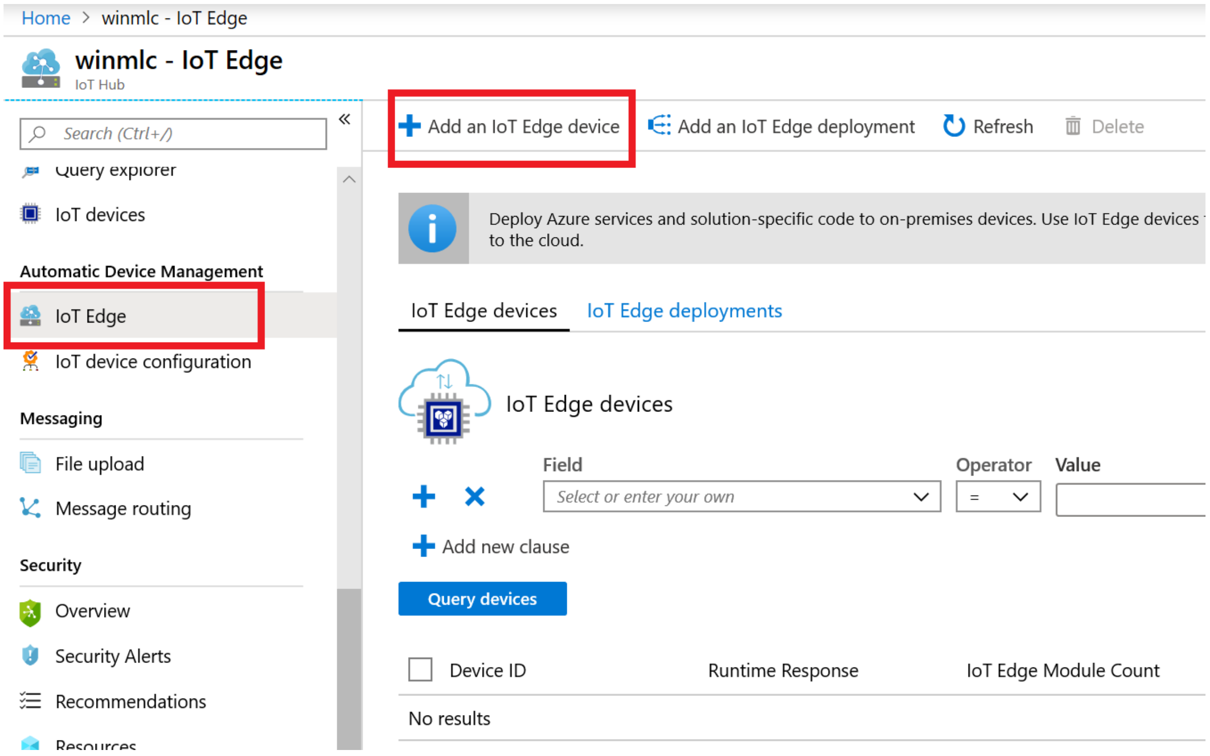
- `docker build . -t your-own-registry.azurecr.io/agent:latest`
- `docker push your-own-registry.azurecr.io/agent:latest`

Build other Azure IoT Edge modules

If there are other modules that will be deployed to the Edge device running Azure IoT Edge and Windows ML Insider Preview container on a Windows Insider host, you will need to rebuild those modules based on the same Windows Insider container image that is matched to the version of the Windows Insider host.

Configure IoT Edge in the Azure IoT Hub portal

- Create an IoT Edge device on your Azure IoT Hub.

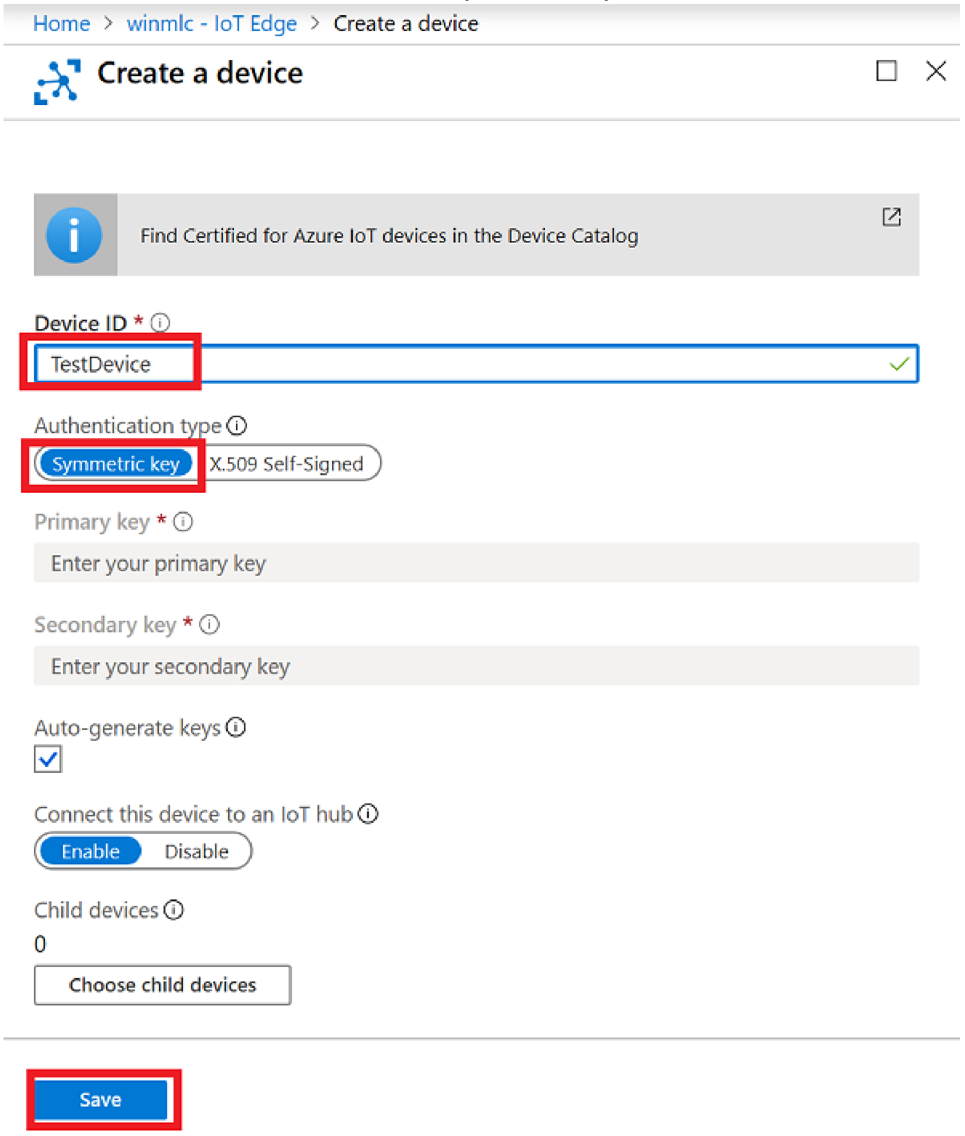

Home > winmlc - IoT Edge

winmlc - IoT Edge
IoT Hub

Deploy Azure services and solution-specific code to on-premises devices. Use IoT Edge devices
to the cloud.

Device ID	Runtime Response	IoT Edge Module Count
No results		

- Name the device **TestDevice** and select Symmetric Key.


Home > winmlc - IoT Edge > Create a device

Create a device

Find Certified for Azure IoT devices in the Device Catalog

Device ID *: TestDevice

Authentication type: [✓ Symmetric key] X.509 Self-Signed

Primary key *: Enter your primary key

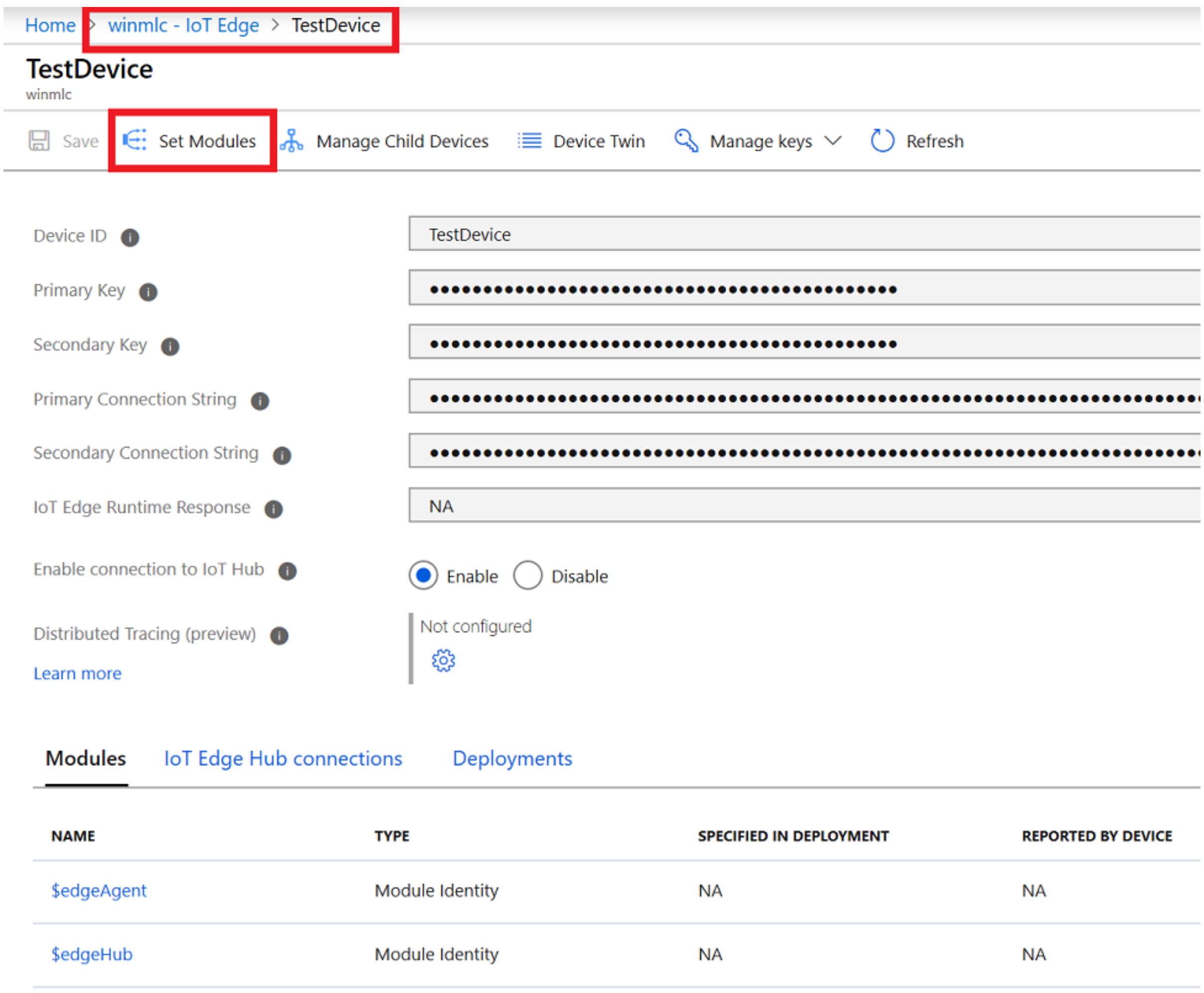
Secondary key *: Enter your secondary key

Auto-generate keys: ✓

Connect this device to an IoT hub: [✓ Enable] Disable

Child devices: 0

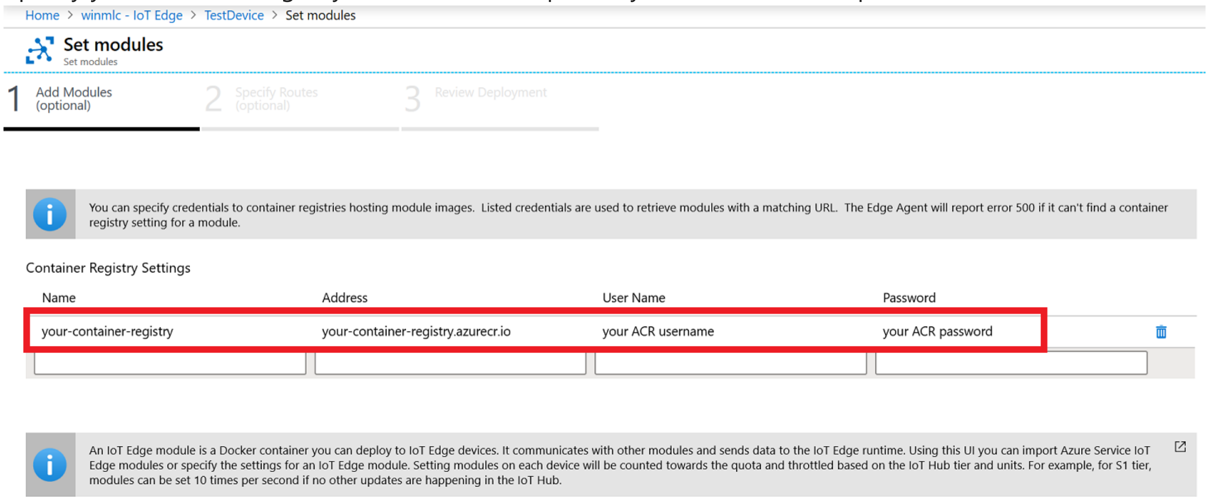
- Select **Set Modules**.



Device ID	TestDevice
Primary Key	••
Secondary Key	••
Primary Connection String	••
Secondary Connection String	••
IoT Edge Runtime Response	NA
Enable connection to IoT Hub	[✓ Enable] Disable
Distributed Tracing (preview) Learn more	Not configured

NAME	TYPE	SPECIFIED IN DEPLOYMENT	REPORTED BY DEVICE
$edgeAgent	Module Identity	NA	NA
$edgeHub	Module Identity	NA	NA

- Specify your container registry details where the privately built modules was pushed to.


Home > winmlc - IoT Edge > TestDevice > Set modules
Set modules
Set modules
1 Add Modules (optional)
2 Specify Routes (optional)
3 Review Deployment

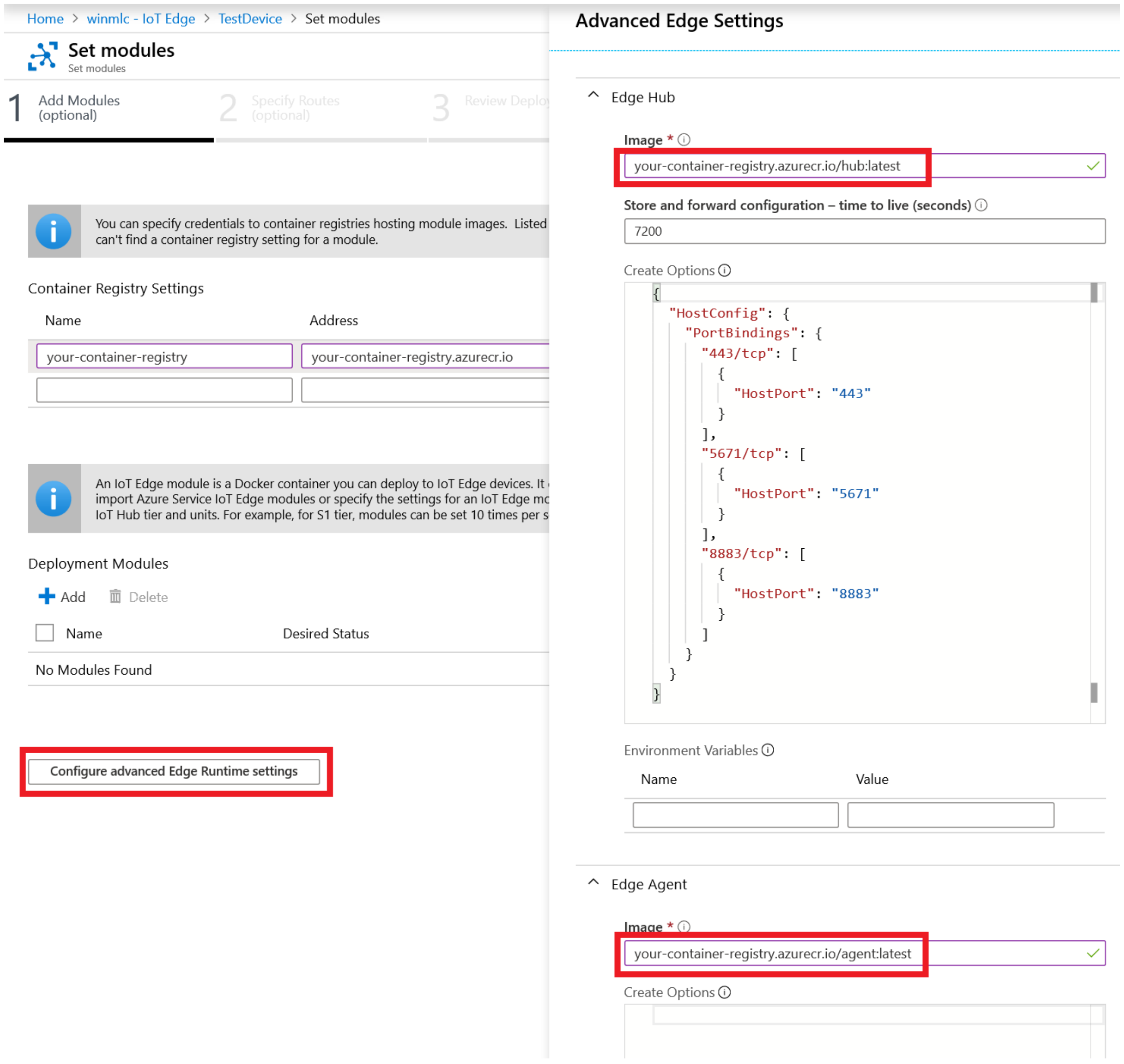
You can specify credentials to container registries hosting module images. Listed credentials are used to retrieve modules with a matching URL. The Edge Agent will report error 500 if it can't find a container registry setting for a module.

Container Registry Settings

Name	Address	User Name	Password
your-container-registry	your-container-registry.azurecr.io	your ACR username	your ACR password

An IoT Edge module is a Docker container you can deploy to IoT Edge devices. It communicates with other modules and sends data to the IoT Edge runtime. Using this UI you can import Azure Service IoT Edge modules or specify the settings for an IoT Edge module. Setting modules on each device will be counted towards the quota and throttled based on the IoT Hub tier and units. For example, for S1 tier, modules can be set 10 times per second if no other updates are happening in the IoT Hub.

- Click on **Configure advanced Edge Runtime Settings** button on Set Modules page, then set Edge Hub and Edge Agent to where you published the private containers to.


Home > winmlc - IoT Edge > TestDevice > Set modules

Set modules
Set modules

1 Add Modules (optional) 2 Specify Routes (optional) 3 Review Deploy

You can specify credentials to container registries hosting module images. Listed can't find a container registry setting for a module.

Container Registry Settings

Name	Address
your-container-registry	your-container-registry.azurecr.io

An IoT Edge module is a Docker container you can deploy to IoT Edge devices. It import Azure Service IoT Edge modules or specify the settings for an IoT Edge mo IoT Hub tier and units. For example, for S1 tier, modules can be set 10 times per s

Deployment Modules

+ Add 🗑 Delete

Name	Desired Status
No Modules Found	

Configure advanced Edge Runtime settings

Advanced Edge Settings

Edge Hub

Image *: your-container-registry.azurecr.io/hub:latest

Store and forward configuration – time to live (seconds): 7200

Create Options

```
{
  "HostConfig": {
    "PortBindings": {
      "443/tcp": [
        {
          "HostPort": "443"
        }
      ],
      "5671/tcp": [
        {
          "HostPort": "5671"
        }
      ],
      "8883/tcp": [
        {
          "HostPort": "8883"
        }
      ]
    }
  }
}
```

Environment Variables

Name	Value

Edge Agent

Image *: your-container-registry.azurecr.io/agent:latest

Create Options

- Select the **Save** button.
- Don't forget to select **next -> next - submit**

From this point, you can optionally add some Deployment modules to your device.

Setting up Azure IoT Edge on Windows ML container host

IMPORTANT

When setting up Azure IoT Edge, make sure that the version of the container OS matches the version of the container host that was used to build the Edge Hub and Edge Agent modules.

- The device where IoT Edge is being installed on below should **not** have Docker running.
- You can stop and disable Docker service if you are experimenting on the container build machine from above.

Download and modify installation powershell script

- Save a copy of `IoTEdgeSecurityDaemon.ps1` locally, with this command running from an *elevated* commant prompt.

```
curl -o c:\IotEdgeSecurityDaemon.ps1 -L
https://raw.githubusercontent.com/Azure/iotedge/1.0.8/scripts/windows/setup/IotEdgeSecurityDaemon.ps1
```

- Modify `IoTEdgeSecurityDaemon.ps1` to disable build version constraints
- In the `Initialize-IoTedge` function, comment out the following line.

```
 #   if (-not (Setup-Environment -ContainerOs $ContainerOs -SkipArchCheck -SkipBatteryCheck)) {
 #       return
 #   }
```

- In the `Install-Packages` function, comment out the following line:

```
#    if (-not (Setup-Environment -ContainerOs $ContainerOs -SkipArchCheck:$SkipArchCheck -
SkipBatteryCheck:$SkipBatteryCheck)) {
#        return
#    }
```

Install IoT Edge

- Run the following command from an *elevated* PowerShell instance:

```
. {Invoke-WebRequest -useb c:\IotEdgeSecurityDaemon.ps1} | Invoke-Expression;  Deploy-IoTEdge
```

- If your device reboots, wait for it to restart. If necessary, launch an *elevated* PowerShell instance once again, and run the following command.

NOTE

Remember to replace the container registry, username, password, and connection string in the below example.

```
. {Invoke-WebRequest -useb c:\IotEdgeSecurityDaemon.ps1} | Invoke-Expression; Initialize-IoTEdge -AgentImage
your-container-registry.azurecr.io/agent:latest  -Username yourRegistryUserName -Password $(ConvertTo-
SecureString yourregistryPassword -AsPlainText -Force)  -Manual -DeviceConnectionString
connectionstringOfTestDevice
```

- Wait for a few minutes for the process to conclude, then run the following command.

```
PS C:\> iotedge list
```

You should be able to see the agent and hub running, as displayed below.

```
NAME             STATUS           DESCRIPTION      CONFIG
edgeHub          running          Up 9 seconds     winmlc.azurecr.io/hub:latest
edgeAgent        running          Up 22 seconds    winmlc.azurecr.io/agent:latest
```

API List

10/16/2019 • 8 minutes to read • Edit Online

APIs exported by OneCore.lib umbrella library are available for native applications in the Windows ML container.

Due to the small size of Windows ML container, only a subset of WinRT APIs are available in the container image. The list below is intended to exhaustively list the types included in the base OS image. If you need a type that's not included, please mail winmlcfb@microsoft.com.

Windows.AI

Windows.AI.MachineLearning

ILearningModelFeatureDescriptor
ILearningModelFeatureValue
ILearningModelOperatorProvider
ITensor
ImageFeatureDescriptor
ImageFeatureValue
LearningModel
LearningModelBinding
LearningModelDevice
LearningModelDeviceKind
LearningModelEvaluationResult
LearningModelFeatureKind
LearningModelSession
LearningModelSessionOptions
MachineLearningContract
MapFeatureDescriptor
SequenceFeatureDescriptor
TensorBoolean
TensorDouble
TensorFeatureDescriptor
TensorFloat
TensorFloat16Bit
TensorInt16Bit
TensorInt32Bit
TensorInt64Bit
TensorInt8Bit
TensorKind
TensorString
TensorUInt16Bit
TensorUInt32Bit
TensorUInt64Bit
TensorUInt8Bit

Windows.ApplicationModel

Windows.ApplicationModel.Background

DeviceWatcherTrigger

IBackgroundTrigger

Windows.Data

Windows.Data.Html

HtmlUtilities

Windows.Data.Json

IJsonValue
JsonArray
JsonError
JsonErrorStatus
JsonObject
JsonValue
JsonValueType

Windows.Data.Text

AlternateNormalizationFormat
AlternateWordForm
SelectableWordSegment
SelectableWordSegmentsTokenizingHandler
SelectableWordsSegmenter
SemanticTextQuery
TextConversionGenerator
TextPhoneme
TextPredictionGenerator
TextPredictionOptions
TextReverseConversionGenerator
TextSegment
UnicodeCharacters
UnicodeGeneralCategory
UnicodeNumericType
WordSegment
WordSegmentsTokenizingHandler
WordsSegmenter

Windows.Data.Xml.Dom

DtdEntity
DtdNotation
IXmlCharacterData
IXmlNode
IXmlNodeSelector
IXmlNodeSerializer
IXmlText br> NodeType
XmlAttribute
XmlCDataSection
XmlComment
XmlDocument
XmlDocumentFragment
XmlDocumentType
XmlDomImplementation
XmlElement
XmlEntityReference

XmlLoadSettings
XmlNamedNodeMap
XmlNodeList
XmlProcessingInstruction
XmlText

Windows.Data.Xml.Xsl

XsltProcessor

Windows.Devices

Windows.Devices

ILowLevelDevicesAggregateProvider
LowLevelDevicesAggregateProvider
LowLevelDevicesController

Windows.Devices.Adc

AdcChannel
AdcChannelMode
AdcController

Windows.Devices.Adc.Provider

ProviderAdcChannelMode

Windows.Devices.Background

DeviceServicingDetails
DeviceUseDetails

Windows.Devices.Bluetooth

BluetoothAdapter
BluetoothAddressType
BluetoothCacheMode
BluetoothClassOfDevice
BluetoothConnectionStatus
BluetoothDevice
BluetoothDeviceId
BluetoothError
BluetoothLEAppearance
BluetoothLEAppearanceCategories
BluetoothLEAppearanceSubcategories
BluetoothLEDevice
BluetoothMajorClass
BluetoothMinorClass
BluetoothServiceCapabilities
BluetoothSignalStrengthFilter
BluetoothUuidHelper

Windows.Devices.Bluetooth.Advertisement

BluetoothLEAdvertisement
BluetoothLEAdvertisementBytePattern
BluetoothLEAdvertisementDataSection
BluetoothLEAdvertisementDataTypes
BluetoothLEAdvertisementFilter

BluetoothLEAdvertisementFlags
BluetoothLEAdvertisementPublisher
BluetoothLEAdvertisementPublisherStatus
BluetoothLEAdvertisementPublisherStatusChangedEventArgs
BluetoothLEAdvertisementReceivedEventArgs
BluetoothLEAdvertisementType
BluetoothLEAdvertisementWatcher
BluetoothLEAdvertisementWatcherStatus
BluetoothLEAdvertisementWatcherStoppedEventArgs
BluetoothLEManufacturerData
BluetoothLEScanningMode

Windows.Devices.Bluetooth.Background

BluetoothEventTriggeringMode
BluetoothLEAdvertisementPublisherTriggerDetails
BluetoothLEAdvertisementWatcherTriggerDetails
GattCharacteristicNotificationTriggerDetails
GattServiceProviderConnection
GattServiceProviderTriggerDetails
RfcommConnectionTriggerDetails
RfcommInboundConnectionInformation
RfcommOutboundConnectionInformation

Windows.Devices.Bluetooth.GenericAttributeProfile

GattCharacteristic
GattCharacteristicProperties
GattCharacteristicUuids
GattCharacteristicsResult
GattClientCharacteristicConfigurationDescriptorValue
GattClientNotificationResult
GattCommunicationStatus
GattDescriptor
GattDescriptorUuids
GattDescriptorsResult
GattDeviceService
GattDeviceServicesResult
GattLocalCharacteristic
GattLocalCharacteristicParameters
GattLocalCharacteristicResult
GattLocalDescriptor
GattLocalDescriptorParameters
GattLocalDescriptorResult
GattLocalService
GattOpenStatus
GattPresentationFormat
GattPresentationFormatTypes
GattProtectionLevel
GattProtocolError
GattReadClientCharacteristicConfigurationDescriptorResult
GattReadRequest
GattReadRequestedEventArgs
GattReadResult

GattReliableWriteTransaction

GattRequestState

GattRequestStateChangedEventArgs

GattServiceProvider

GattServiceProviderAdvertisementStatus

GattServiceProviderAdvertisementStatusChangedEventArgs

GattServiceProviderAdvertisingParameters

GattServiceProviderResult

GattServiceUuids

GattSession

GattSessionStatus

GattSessionStatusChangedEventArgs

GattSharingMode

GattSubscribedClient

GattValueChangedEventArgs

GattWriteOption

GattWriteRequest

GattWriteRequestedEventArgs

GattWriteResult

Windows.Devices.Bluetooth.Rfcomm

RfcommDeviceService

RfcommDeviceServicesResult

RfcommServiceId

RfcommServiceProvider

Windows.Devices.Custom

CustomDevice

CustomDeviceContract

DeviceAccessMode

DeviceSharingMode

IOControlCode

KnownDeviceTypes

Windows.Devices.Enumeration

DeviceAccessChangedEventArgs

DeviceAccessInformation

DeviceAccessStatus

DeviceClass

DeviceConnectionChangeTriggerDetails

DeviceDisconnectButtonClickedEventArgs

DeviceInformation

DeviceInformationCollection

DeviceInformationCustomPairing

DeviceInformationKind

DeviceInformationPairing

DeviceInformationUpdate

DevicePairingKinds

DevicePairingProtectionLevel

DevicePairingRequestedEventArgs

DevicePairingResult

DevicePairingResultStatus

DevicePickerDisplayStatusOptions

DevicePickerFilter

DeviceSelectedEventArgs

DeviceThumbnail

DeviceUnpairingResult

DeviceUnpairingResultStatus

DeviceWatcher

DeviceWatcherEvent

DeviceWatcherEventKind

DeviceWatcherStatus

DeviceWatcherTriggerDetails

EnclosureLocation

IDevicePairingSettings

Panel

Windows.Devices.Enumeration.Pnp

PnpObject

PnpObjectCollection

PnpObjectType

PnpObjectUpdate

PnpObjectWatcher

Windows.Devices.Gpio

GpioChangeCount

GpioChangeCounter

GpioChangePolarity

GpioChangeReader

GpioChangeRecord

GpioController

GpioOpenStatus

GpioPin

GpioPinDriveMode

GpioPinEdge

GpioPinValue

GpioPinValueChangedEventArgs

GpioSharingMode

Windows.Devices.Gpio.Provider

GpioPinProviderValueChangedEventArgs

IGpioControllerProvider

IGpioPinProvider

IGpioProvider

ProviderGpioPinDriveMode

ProviderGpioPinEdge

ProviderGpioPinValue

ProviderGpioSharingMode

Windows.Devices.HumanInterfaceDevice

HidBooleanControl

HidBooleanControlDescription

HidCollection

HidCollectionType

HidDevice

HidFeatureReport

HidInputReport
HidInputReportReceivedEventArgs
HidNumericControl
HidNumericControlDescription
HidOutputReport
HidReportType

Windows.Devices.I2c

I2cBusSpeed
I2cConnectionSettings
I2cController
I2cDevice
I2cSharingMode
I2cTransferResult
I2cTransferStatus
II2cDeviceStatics

Windows.Devices.I2c.Provider

II2cControllerProvider
II2cDeviceProvider
II2cProvider
ProviderI2cBusSpeed
ProviderI2cConnectionSettings
ProviderI2cSharingMode
ProviderI2cTransferResult
ProviderI2cTransferStatus

Windows.Devices.Power

Battery
BatteryReport

Windows.Devices.Pwm

PwmController
PwmPin
PwmPulsePolarity

Windows.Devices.Pwm.Provider

IPwmControllerProvider
IPwmProvider

Windows.Devices.Radios

Radio
RadioAccessStatus
RadioKind
RadioState

Windows.Devices.Sensors

Accelerometer
AccelerometerDataThreshold
AccelerometerReading
AccelerometerReadingChangedEventArgs
AccelerometerReadingType
AccelerometerShakenEventArgs
ActivitySensor

ActivitySensorReading
ActivitySensorReadingChangeReport
ActivitySensorReadingChangedEventArgs
ActivitySensorReadingConfidence
ActivitySensorTriggerDetails
ActivityType
Altimeter
AltimeterReading
AltimeterReadingChangedEventArgs
Barometer
BarometerDataThreshold
BarometerReading
BarometerReadingChangedEventArgs
Compass
CompassDataThreshold
CompassReading
CompassReadingChangedEventArgs
Gyrometer
GyrometerDataThreshold
GyrometerReading
GyrometerReadingChangedEventArgs
HingeAngleReading
HingeAngleSensor
HingeAngleSensorReadingChangedEventArgs
ISensorDataThreshold
Inclinometer
InclinometerDataThreshold
InclinometerReading
InclinometerReadingChangedEventArgs
LightSensor
LightSensorDataThreshold
LightSensorReading
LightSensorReadingChangedEventArgs
Magnetometer
MagnetometerAccuracy
MagnetometerDataThreshold
MagnetometerReading
MagnetometerReadingChangedEventArgs
OrientationSensor
OrientationSensorReading
OrientationSensorReadingChangedEventArgs
Pedometer
PedometerDataThreshold
PedometerReading
PedometerReadingChangedEventArgs
PedometerStepKind
ProximitySensor
ProximitySensorDataThreshold
ProximitySensorDisplayOnOffController
ProximitySensorReading
ProximitySensorReadingChangedEventArgs
SensorDataThresholdTriggerDetails

SensorOptimizationGoal
SensorQuaternion
SensorReadingType
SensorRotationMatrix
SensorType
SimpleOrientation
SimpleOrientationSensor
SimpleOrientationSensorOrientationChangedEventArgs

Windows.Devices.Sensors.Custom

CustomSensor
CustomSensorReading
CustomSensorReadingChangedEventArgs

Windows.Devices.SerialCommunication

ErrorReceivedEventArgs
PinChangedEventArgs
SerialDevice
SerialError
SerialHandshake
SerialParity
SerialPinChange
SerialStopBitCount

Windows.Devices.Spi

ISpiDeviceStatics
SpiBusInfo
SpiConnectionSettings
SpiController
SpiDevice
SpiMode
SpiSharingMode

Windows.Devices.Spi.Provider

ISpiControllerProvider
ISpiDeviceProvider
ISpiProvider
ProviderSpiConnectionSettings
ProviderSpiMode
ProviderSpiSharingMode

Windows.Devices.Usb

UsbBulkInEndpointDescriptor
UsbBulkInPipe
UsbBulkOutEndpointDescriptor
UsbBulkOutPipe
UsbConfiguration
UsbConfigurationDescriptor
UsbControlRecipient
UsbControlRequestType
UsbControlTransferType
UsbDescriptor
UsbDevice

UsbDeviceClass
UsbDeviceClasses
UsbDeviceDescriptor
UsbEndpointDescriptor
UsbEndpointType
UsbInterface
UsbInterfaceDescriptor
UsbInterfaceSetting
UsbInterruptInEndpointDescriptor
UsbInterruptInEventArgs
UsbInterruptInPipe
UsbInterruptOutEndpointDescriptor
UsbInterruptOutPipe
UsbReadOptions
UsbSetupPacket
UsbTransferDirection
UsbWriteOptions

Windows.Devices.WiFi

WiFiAccessStatus
WiFiAdapter
WiFiAvailableNetwork
WiFiConnectionMethod
WiFiConnectionResult
WiFiConnectionStatus
WiFiNetworkKind
WiFiNetworkReport
WiFiPhyKind
WiFiReconnectionKind
WiFiWpsConfigurationResult
WiFiWpsConfigurationStatus
WiFiWpsKind

Windows.Foundation

Windows.Foundation

AsyncActionCompletedHandler
AsyncActionProgressHandler
AsyncActionWithProgressCompletedHandler
AsyncOperationCompletedHandler
AsyncOperationProgressHandler
AsyncOperationWithProgressCompletedHandler
AsyncStatus
DateTime
Deferral
DeferralCompletedHandler
EventHandler
EventRegistrationToken
FoundationContract
GuidHelper
HResult
IAsyncAction

IAsyncActionWithProgress

IAsyncInfo

IAsyncOperationWithProgress

IAsyncOperation

IClosable

IGetActivationFactory

IMemoryBuffer

IMemoryBufferReference

IPropertyValue

IReferenceArray

IReference

IStringable

IWwwFormUrlDecoderEntry

MemoryBuffer

Point

PropertyType

PropertyValue

Rect

Size

TimeSpan

TypedEventHandler

UniversalApiContract

Uri

WwwFormUrlDecoder

WwwFormUrlDecoderEntry

Windows.Foundation.Collections

CollectionChange

IIterable

IIterator

IKeyValuePair

IMapChangedEventArgs

IMapView

IMap

IObservableMap

IObservableVector

IPropertySet

IVectorChangedEventArgs

IVectorView

IVector

MapChangedEventHandler

PropertySet

StringMap

ValueSet

VectorChangedEventHandler

Windows.Foundation.Diagnostics

AsyncCausalityTracer

CausalityRelation

CausalitySource

CausalitySynchronousWork

CausalityTraceLevel

ErrorDetails
ErrorOptions
FileLoggingSession
IErrorReportingSettings
IFileLoggingSession
ILoggingChannel
ILoggingSession
ILoggingTarget
LogFileGeneratedEventArgs
LoggingActivity
LoggingChannel
LoggingChannelOptions
LoggingFieldFormat
LoggingFields
LoggingLevel
LoggingOpcode
LoggingOptions
LoggingSession
RuntimeBrokerErrorSettings
TracingStatusChangedEventArgs

Windows.Foundation.Metadata

ActivatableAttribute
AllowForWebAttribute
AllowMultipleAttribute
ApiContractAttribute
ApiInformation
AttributeNameAttribute
AttributeTargets
AttributeUsageAttribute
ComposableAttribute
CompositionType
ContractVersionAttribute
CreateFromStringAttribute
DefaultAttribute
DefaultOverloadAttribute
DeprecatedAttribute
DeprecationType
DualApiPartitionAttribute
ExclusiveToAttribute
ExperimentalAttribute
FastAbiAttribute
FeatureAttribute
FeatureStage
GCPressureAmount
GCPressureAttribute
GuidAttribute
HasVariantAttribute
InternalAttribute
LengthIsAttribute
MarshalingBehaviorAttribute
MarshalingType

MetadataMarshalAttribute

MuseAttribute

NoExceptionAttribute

OverloadAttribute

OverridableAttribute

Platform

PlatformAttribute

PreviousContractVersionAttribute

ProtectedAttribute

RangeAttribute

RemoteAsyncAttribute

StaticAttribute

ThreadingAttribute

ThreadingModel

VariantAttribute

VersionAttribute

WebHostHiddenAttribute

Windows.Foundation.Numerics

Matrix3x2

Matrix4x4

Plane

Quaternion

Rational

Vector2

Vector3

Vector4

Windows.Globalization

Windows.Globalization

ApplicationLanguages

Calendar

CalendarIdentifiers

ClockIdentifiers

CurrencyAmount

CurrencyIdentifiers

DayOfWeek

GeographicRegion

Language

LanguageLayoutDirection

NumeralSystemIdentifiers

Windows.Globalization.Collation

CharacterGrouping

CharacterGroupings

Windows.Globalization.DateTimeFormatting

DateTimeFormatter

DayFormat

DayOfWeekFormat

HourFormat

MinuteFormat

MonthFormat
SecondFormat
YearFormat

Windows.Globalization.NumberFormatting

CurrencyFormatter
CurrencyFormatterMode
DecimalFormatter
INumberFormatter
INumberFormatter2
INumberFormatterOptions
INumberParser
INumberRounder
INumberRounderOption
ISignedZeroOption
ISignificantDigitsOption
IncrementNumberRounder
NumeralSystemTranslator
PercentFormatter
PermilleFormatter
RoundingAlgorithm
SignificantDigitsNumberRounder

Windows.Globalization.PhoneNumberFormatting

PhoneNumberFormat
PhoneNumberFormatter
PhoneNumberInfo
PhoneNumberMatchResult
PhoneNumberParseResult
PredictedPhoneNumberKind

Windows.Graphics

Windows.Graphics

DisplayAdapterId

Windows.Graphics.DirectX

DirectXPixelFormat

Windows.Graphics.DirectX.Direct3D11

Direct3DMultisampleDescription
Direct3DSurfaceDescription
IDirect3DDevice
IDirect3DSurface

Windows.Graphics.Display

Windows.Graphics.Display.DisplayOrientations

Windows.Graphics.Imaging

NOTE

Some APIs in this namespace have restrictions when used in the Windows ML container. See Limitations for details.

BitmapAlphaMode
BitmapBounds
BitmapBuffer
BitmapBufferAccessMode
BitmapCodecInformation
BitmapDecoder
BitmapEncoder
BitmapFlip
BitmapFrame
BitmapInterpolationMode
BitmapPixelFormat
BitmapPlaneDescription
BitmapProperties
BitmapPropertiesView
BitmapPropertySet
BitmapRotation
BitmapSize
BitmapTransform
BitmapTypedValue
ColorManagementMode
ExifOrientationMode
IBitmapFrame
IBitmapFrameWithSoftwareBitmap
IBitmapPropertiesView
ImageStream
JpegSubsamplingMode
PixelDataProvider
PngFilterMode
SoftwareBitmap
TiffCompressionMode

Windows.Media

Windows.Media

AudioBuffer
AudioBufferAccessMode
AudioFrame
AudioProcessing
AutoRepeatModeChangeRequestedEventArgs
IMediaExtension
IMediaFrame
IMediaMarker
IMediaMarkers
ImageDisplayProperties
MediaMarkerTypes
MediaPlaybackAutoRepeatMode
MediaPlaybackStatus
MediaPlaybackType
MediaProcessingTriggerDetails
MediaTimeRange
MediaTimelineController
MediaTimelineControllerFailedEventArgs

MediaTimelineControllerState
MusicDisplayProperties
PlaybackPositionChangeRequestedEventArgs
PlaybackRateChangeRequestedEventArgs
ShuffleEnabledChangeRequestedEventArgs
SoundLevel
SystemMediaTransportControls
SystemMediaTransportControlsButton
SystemMediaTransportControlsButtonPressedEventArgs
SystemMediaTransportControlsDisplayUpdater
SystemMediaTransportControlsProperty
SystemMediaTransportControlsPropertyChangedEventArgs
SystemMediaTransportControlsTimelineProperties
VideoDisplayProperties
VideoEffects
VideoFrame

Windows.Networking

Windows.Networking

DomainNameType
EndpointPair
HostName
HostNameSortOptions
HostNameType

Windows.Networking.BackgroundTransfer

BackgroundDownloadProgress
BackgroundTransferBehavior
BackgroundTransferCompletionGroup
BackgroundTransferCompletionGroupTriggerDetails
BackgroundTransferContentPart
BackgroundTransferCostPolicy
BackgroundTransferError
BackgroundTransferFileRange
BackgroundTransferGroup
BackgroundTransferPriority
BackgroundTransferRangesDownloadedEventArgs
BackgroundTransferStatus
BackgroundUploadProgress
ContentPrefetcher
DownloadOperation
IBackgroundTransferBase
IBackgroundTransferContentPartFactory
IBackgroundTransferOperation
IBackgroundTransferOperationPriority
ResponseInformation
UnconstrainedTransferRequestResult
UploadOperation

Windows.Networking.Connectivity

AttributedNetworkUsage

CellularApnAuthenticationType
CellularApnContext
ConnectionCost
ConnectionProfile
ConnectionProfileDeleteStatus
ConnectionProfileFilter
ConnectionSession
ConnectivityInterval
ConnectivityManager
DataPlanStatus
DataPlanUsage
DataUsage
DataUsageGranularity
DomainConnectivityLevel
LanIdentifier
LanIdentifierData
NetworkAdapter
NetworkAuthenticationType
NetworkConnectivityLevel
NetworkCostType
NetworkEncryptionType
NetworkInformation
NetworkItem
NetworkSecuritySettings
NetworkStateChangeEventDetails
NetworkStatusChangedEventHandler
NetworkTypes
NetworkUsage
NetworkUsageStates
ProviderNetworkUsage
ProxyConfiguration
RoamingStates
RoutePolicy
TriStates
WlanConnectionProfileDetails
WwanConnectionProfileDetails
WwanContract
WwanDataClass
WwanNetworkIPKind
WwanNetworkRegistrationState

Windows.Networking.Sockets

BandwidthStatistics
ControlChannelTrigger
ControlChannelTriggerContract
ControlChannelTriggerResetReason
ControlChannelTriggerResourceType
ControlChannelTriggerStatus
DatagramSocket
DatagramSocketControl
DatagramSocketInformation
DatagramSocketMessageReceivedEventArgs

IControlChannelTriggerEventDetails
IControlChannelTriggerResetEventDetails
IWebSocket
IWebSocketControl
IWebSocketControl2
IWebSocketInformation
IWebSocketInformation2
MessageWebSocket
MessageWebSocketControl
MessageWebSocketInformation
MessageWebSocketMessageReceivedEventArgs
MessageWebSocketReceiveMode
RoundTripTimeStatistics
ServerMessageWebSocket
ServerMessageWebSocketControl
ServerMessageWebSocketInformation
ServerStreamWebSocket
ServerStreamWebSocketInformation
SocketActivityConnectedStandbyAction
SocketActivityContext
SocketActivityInformation
SocketActivityKind
SocketActivityTriggerDetails
SocketActivityTriggerReason
SocketError
SocketErrorStatus
SocketMessageType
SocketProtectionLevel
SocketQualityOfService
SocketSslErrorSeverity
StreamSocket
StreamSocketControl
StreamSocketInformation
StreamSocketListener
StreamSocketListenerConnectionReceivedEventArgs
StreamSocketListenerControl
StreamSocketListenerInformation
StreamWebSocket
StreamWebSocketControl
StreamWebSocketInformation
WebSocketClosedEventArgs
WebSocketError
WebSocketServerCustomValidationRequestedEventArgs

Windows.Security

Windows.Security.Credentials

PasswordCredential

Windows.Security.Cryptography

BinaryStringEncoding
CryptographicBuffer

Windows.Security.Cryptography.Certificates

Certificate
CertificateChain
CertificateChainPolicy
CertificateEnrollmentManager
CertificateExtension
CertificateKeyUsages
CertificateQuery
CertificateRequestProperties
CertificateStore
CertificateStores
ChainBuildingParameters
ChainValidationParameters
ChainValidationResult
CmsAttachedSignature
CmsDetachedSignature
CmsSignerInfo
CmsTimestampInfo
EnrollKeyUsages
ExportOption
InstallOptions
KeyAlgorithmNames
KeyAttestationHelper
KeyProtectionLevel
KeySize
KeyStorageProviderNames
PfxImportParameters
SignatureValidationResult
StandardCertificateStoreNames
SubjectAlternativeNameInfo
UserCertificateEnrollmentManager
UserCertificateStore

Windows.Security.Cryptography.Core

AsymmetricAlgorithmNames
AsymmetricKeyAlgorithmProvider
Capi1KdfTargetAlgorithm
CryptographicEngine
CryptographicHash
CryptographicKey
CryptographicPadding
CryptographicPrivateKeyBlobType
CryptographicPublicKeyBlobType
EccCurveNames
EncryptedAndAuthenticatedData
HashAlgorithmNames
HashAlgorithmProvider
KeyDerivationAlgorithmNames
KeyDerivationAlgorithmProvider
KeyDerivationParameters
MacAlgorithmNames
MacAlgorithmProvider

PersistedKeyProvider
SymmetricAlgorithmNames
SymmetricKeyAlgorithmProvider

Windows.Security.Cryptography.DataProtection

DataProtectionProvider

Windows.Security.EnterpriseData

BufferProtectUnprotectResult
DataProtectionInfo
DataProtectionManager
DataProtectionStatus
EnforcementLevel
EnterpriseDataContract
FileProtectionInfo
FileProtectionManager
FileProtectionStatus
FileRevocationManager
FileUnprotectOptions
ProtectedAccessResumedEventArgs
ProtectedAccessSuspendingEventArgs
ProtectedContainerExportResult
ProtectedContainerImportResult
ProtectedContentRevokedEventArgs
ProtectedFileCreateResult
ProtectedImportExportStatus
ProtectionPolicyAuditAction
ProtectionPolicyAuditInfo
ProtectionPolicyEvaluationResult
ProtectionPolicyManager
ProtectionPolicyRequestAccessBehavior
ThreadNetworkContext

Windows.Security.ExchangeActiveSyncProvisioning

EasClientDeviceInformation
EasClientSecurityPolicy
EasComplianceResults
EasContract
EasDisallowConvenienceLogonResult
EasEncryptionProviderType
EasMaxInactivityTimeLockResult
EasMaxPasswordFailedAttemptsResult
EasMinPasswordComplexCharactersResult
EasMinPasswordLengthResult
EasPasswordExpirationResult
EasPasswordHistoryResult
EasRequireEncryptionResult

Windows.Storage

Windows.Storage

AppDataPaths
ApplicationData

ApplicationDataCompositeValue
ApplicationDataContainer
ApplicationDataContainerSettings
ApplicationDataCreateDisposition
ApplicationDataLocality
ApplicationDataSetVersionHandler
CachedFileManager
CreationCollisionOption
DownloadsFolder
FileAccessMode
FileAttributes
FileIO
IStorageFile
IStorageFile2
IStorageFilePropertiesWithAvailability
IStorageFolder
IStorageFolder2
IStorageItem
IStorageItem2
IStorageItemProperties
IStorageItemProperties2
IStorageItemPropertiesWithProvider
IStreamedFileDataRequest
KnownFolderId
KnownFolders
KnownFoldersAccessStatus
KnownLibraryId
NameCollisionOption
PathIO
SetVersionDeferral
SetVersionRequest
StorageDeleteOption
StorageFile
StorageFolder
StorageItemTypes
StorageLibrary
StorageLibraryChange
StorageLibraryChangeReader
StorageLibraryChangeTracker
StorageLibraryChangeType
StorageOpenOptions
StorageProvider
StorageStreamTransaction
StreamedFileDataRequest
StreamedFileDataRequestedHandler
StreamedFileFailureMode
SystemAudioProperties
SystemDataPaths
SystemGPSProperties
SystemImageProperties
SystemMediaProperties
SystemMusicProperties

SystemPhotoProperties
SystemProperties
SystemVideoProperties
UserDataPaths

Windows.Storage.AccessCache

AccessCacheOptions
AccessListEntry
AccessListEntryView
IStorageItemAccessList
ItemRemovedEventArgs
RecentStorageItemVisibility
StorageApplicationPermissions
StorageItemAccessList
StorageItemMostRecentlyUsedList

Windows.Storage.BulkAccess

FileInformation
FileInformationFactory
FolderInformation
IStorageItemInformation

Windows.Storage.Compression

CompressAlgorithm
Compressor
Decompressor

Windows.Storage.FileProperties

BasicProperties
DocumentProperties
IStorageItemExtraProperties
ImageProperties
MusicProperties
PhotoOrientation
PropertyPrefetchOptions
StorageItemContentProperties
StorageItemThumbnail
ThumbnailMode
ThumbnailOptions
ThumbnailType
VideoOrientation
VideoProperties

Windows.Storage.Provider

CachedFileOptions
CachedFileTarget
CachedFileUpdater
CachedFileUpdaterUI
CloudFilesContract
FileUpdateRequest
FileUpdateRequestDeferral
FileUpdateRequestedEventArgs
FileUpdateStatus

IStorageProviderItemPropertySource
IStorageProviderPropertyCapabilities
IStorageProviderUriSource
ReadActivationMode
StorageProviderFileTypeInfo
StorageProviderGetContentInfoForPathResult
StorageProviderGetPathForContentUriResult
StorageProviderHardlinkPolicy
StorageProviderHydrationPolicy
StorageProviderHydrationPolicyModifier
StorageProviderInSyncPolicy
StorageProviderItemProperties
StorageProviderItemProperty
StorageProviderItemPropertyDefinition
StorageProviderPopulationPolicy
StorageProviderProtectionMode
StorageProviderSyncRootInfo
StorageProviderSyncRootManager
StorageProviderUriSourceStatus
UIStatus
WriteActivationMode

Windows.Storage.Search

CommonFileQuery
CommonFolderQuery
ContentIndexer
ContentIndexerQuery
DateStackOption
FolderDepth
IIndexableContent
IStorageFolderQueryOperations
IStorageQueryResultBase
IndexableContent
IndexedState
IndexerOption
QueryOptions
SortEntry
SortEntryVector
StorageFileQueryResult
StorageFolderQueryResult
StorageItemQueryResult
StorageLibraryChangeTrackerTriggerDetails
StorageLibraryContentChangedTriggerDetails
ValueAndLanguage

Windows.Storage.Streams

Buffer
ByteOrder
DataReader
DataReaderLoadOperation
DataWriter
DataWriterStoreOperation

FileInputStream
FileOpenDisposition
FileOutputStream
FileRandomAccessStream
IBuffer
IContentTypeProvider
IDataReader
IInputStream
IInputStreamReference
IOutputStream
IRandomAccessStream
IRandomAccessStreamReference
IRandomAccessStreamWithContentType
InMemoryRandomAccessStream
InputStreamOptions
InputStreamOverStream
OutputStreamOverStream
RandomAccessStream
RandomAccessStreamOverStream
RandomAccessStreamReference
UnicodeEncoding

Windows.System

Windows.System

AppDiagnosticInfoWatcherStatus
AppExecutionStateChangeResult
AppMemoryReport
AppMemoryUsageLevel
AppMemoryUsageLimitChangingEventArgs
AppResourceGroupBackgroundTaskReport
AppResourceGroupEnergyQuotaState
AppResourceGroupExecutionState
AppResourceGroupInfoWatcherStatus
AppResourceGroupMemoryReport
AppResourceGroupStateReport
AppUriHandlerHost
AppUriHandlerRegistration
AppUriHandlerRegistrationManager
AutoUpdateTimeZoneStatus
DateTimeSettings
DiagnosticAccessStatus
DispatcherQueue
DispatcherQueueController
DispatcherQueueHandler
DispatcherQueuePriority
DispatcherQueueShutdownStartingEventArgs
DispatcherQueueTimer
KnownUserProperties
LaunchFileStatus
LaunchQuerySupportStatus
LaunchQuerySupportType

LaunchUriResult

LaunchUriStatus

MemoryManager

PowerState

ProcessLauncher

ProcessLauncherOptions

ProcessLauncherResult

ProcessMemoryReport

ProcessorArchitecture

ProtocolForResultsOperation

RemoteLaunchUriStatus

RemoteLauncherOptions

ShutdownKind

ShutdownManager

SystemManagementContract

TimeZoneSettings

Windows.System.Diagnostics

DiagnosticActionResult

DiagnosticActionState

DiagnosticInvoker

ProcessCpuUsage

ProcessCpuUsageReport

ProcessDiskUsage

ProcessDiskUsageReport

ProcessMemoryUsage

ProcessMemoryUsageReport

SystemCpuUsage

SystemCpuUsageReport

SystemDiagnosticInfo

SystemMemoryUsage

SystemMemoryUsageReport

Windows.System.Diagnostics.Telemetry

PlatformTelemetryClient

PlatformTelemetryRegistrationResult

PlatformTelemetryRegistrationSettings

PlatformTelemetryRegistrationStatus

Windows.System.Diagnostics.TraceReporting

PlatformDiagnosticActionState

PlatformDiagnosticActions

PlatformDiagnosticEscalationType

PlatformDiagnosticEventBufferLatencies

PlatformDiagnosticTraceInfo

PlatformDiagnosticTracePriority

PlatformDiagnosticTraceRuntimeInfo

PlatformDiagnosticTraceSlotState

PlatformDiagnosticTraceSlotType

Windows.System.Display

DisplayRequest

Windows.System.Inventory

InstalledDesktopApp

Windows.System.Power

BackgroundEnergyManager
BatteryStatus
EnergySaverStatus
ForegroundEnergyManager
PowerManager
PowerSupplyStatus

Windows.System.Power.Diagnostics

BackgroundEnergyDiagnostics
ForegroundEnergyDiagnostics

Windows.System.Profile

AnalyticsInfo
AnalyticsVersionInfo
AppApplicability
EducationSettings
HardwareIdentification
HardwareToken
KnownRetailInfoProperties
PlatformDataCollectionLevel
PlatformDiagnosticsAndUsageDataSettings
ProfileHardwareTokenContract
ProfileRetailInfoContract
ProfileSharedModeContract
RetailInfo
SharedModeSettings
SystemIdentification
SystemIdentificationInfo
SystemIdentificationSource
SystemOutOfBoxExperienceState
SystemSetupInfo
UnsupportedAppRequirement
UnsupportedAppRequirementReasons
WindowsIntegrityPolicy

Windows.System.Profile.SystemManufacturers

OemSupportInfo
SmbiosInformation
SystemManufacturersContract
SystemSupportDeviceInfo
SystemSupportInfo

Windows.System.Threading

ThreadPool
ThreadPoolTimer
TimerDestroyedHandler
TimerElapsedHandler
WorkItemHandler
WorkItemOptions

WorkItemPriority

Windows.System.Threading.Core

PreallocatedWorkItem
SignalHandler
SignalNotifier

Windows.System.User

User
UserAuthenticationStatus
UserAuthenticationStatusChangeDeferral
UserAuthenticationStatusChangingEventArgs
UserChangedEventArgs
UserDeviceAssociation
UserDeviceAssociationChangedEventArgs
UserPicker
UserPictureSize
UserType
UserWatcher
UserWatcherStatus
UserWatcherUpdateKind
VirtualKey
VirtualKeyModifiers

Windows.UI

Windows.UI.Text.Core

CoreTextInputScope

Windows.Web

Windows.Web

IUriToStreamResolver
WebError
WebErrorStatus

Windows.Web.AtomPub

AtomPubClient
ResourceCollection
ServiceDocument
Workspace

Windows.Web.Http

HttpBufferContent
HttpClient
HttpCompletionOption
HttpCookie
HttpCookieCollection
HttpCookieManager
HttpFormUrlEncodedContent
HttpGetBufferResult
HttpGetInputStreamResult
HttpGetStringResult

HttpMethod
HttpMultipartContent
HttpMultipartFormDataContent
HttpProgress
HttpProgressStage
HttpRequestMessage
HttpRequestResult
HttpResponseMessage
HttpResponseMessageSource
HttpStatusCode
HttpStreamContent
HttpStringContent
HttpTransportInformation
HttpVersion
IHttpContent

Windows.Web.Http.Diagnostics

HttpDiagnosticProvider
HttpDiagnosticProviderRequestResponseCompletedEventArgs
HttpDiagnosticProviderRequestResponseTimestamps
HttpDiagnosticProviderRequestSentEventArgs
HttpDiagnosticProviderResponseReceivedEventArgs
HttpDiagnosticRequestInitiator
HttpDiagnosticSourceLocation
HttpDiagnosticsContract

Windows.Web.Http.Filters

HttpBaseProtocolFilter
HttpCacheControl
HttpCacheReadBehavior
HttpCacheWriteBehavior
HttpCookieUsageBehavior
HttpServerCustomValidationRequestedEventArgs
IHttpFilter

Windows.Web.Http.Headers

HttpCacheDirectiveHeaderValueCollection
HttpChallengeHeaderValue
HttpChallengeHeaderValueCollection
HttpConnectionOptionHeaderValue
HttpConnectionOptionHeaderValueCollection
HttpContentCodingHeaderValue
HttpContentCodingHeaderValueCollection
HttpContentCodingWithQualityHeaderValue
HttpContentCodingWithQualityHeaderValueCollection
HttpContentDispositionHeaderValue
HttpContentHeaderCollection
HttpContentRangeHeaderValue
HttpCookiePairHeaderValue
HttpCookiePairHeaderValueCollection
HttpCredentialsHeaderValue
HttpDateOrDeltaHeaderValue
HttpExpectationHeaderValue

HttpExpectationHeaderValueCollection
HttpLanguageHeaderValueCollection
HttpLanguageRangeWithQualityHeaderValue
HttpLanguageRangeWithQualityHeaderValueCollection
HttpMediaTypeHeaderValue
HttpMediaTypeWithQualityHeaderValue
HttpMediaTypeWithQualityHeaderValueCollection
HttpMethodHeaderValueCollection
HttpNameValueHeaderValue
HttpProductHeaderValue
HttpProductInfoHeaderValue
HttpProductInfoHeaderValueCollection
HttpRequestHeaderCollection
HttpResponseHeaderCollection
HttpTransferCodingHeaderValue
HttpTransferCodingHeaderValueCollection

Windows.Web.Syndication

ISyndicationClient
ISyndicationNode
ISyndicationText
RetrievalProgress
SyndicationAttribute
SyndicationCategory
SyndicationClient
SyndicationContent
SyndicationError
SyndicationErrorStatus
SyndicationFeed
SyndicationFormat
SyndicationGenerator
SyndicationItem
SyndicationLink
SyndicationNode
SyndicationPerson
SyndicationText
SyndicationTextType
TransferProgress

Limitations

Windows.Graphics.Imaging

Several APIs in the Windows.Graphics.Imaging namespace are subject to limitations when used in the Windows ML container. These restrictions are as follows.

- Windows.Graphics.Imaging supports encoding and decoding for JPG, PNG, GIF, TIFF, and BMP file formats only.
- If using the BMP format with BI_BITFIELDS compression, only the formats 16bppBGR555, 16bppBGR565, and 32bppBGRA are supported.
- There is no support for CMYK, High Dynamic Range, Half-float pixel formats, Fixed-point pixel formats, or images with more than four channels.
- There is no support for 32bppRGBE, 16bppYQuantizedDctCoefficients, 16bppCbQuantizedDctCoefficients, and 16bppCrQuantizedDctCoefficients pixel formats.

- There is no support for reading or writing XMP metadata or Photo Metadata Policies. In transcoding, the API will attempt to preserve XMP metadata if possible.
- There is no support for colorspace manipulation and conversation.
- There is *partial* support for the `ColorManageToSRgb` flag in GetPixelDataAsync and GetSoftwareBitmapAsync. If the flag is specified, the operation will succeed if the image has a commonly-recognized sRGB color profile. Otherwise, the HRESULT equivalent of `ERROR_ICM_NOT_ENABLED` will be returned.