

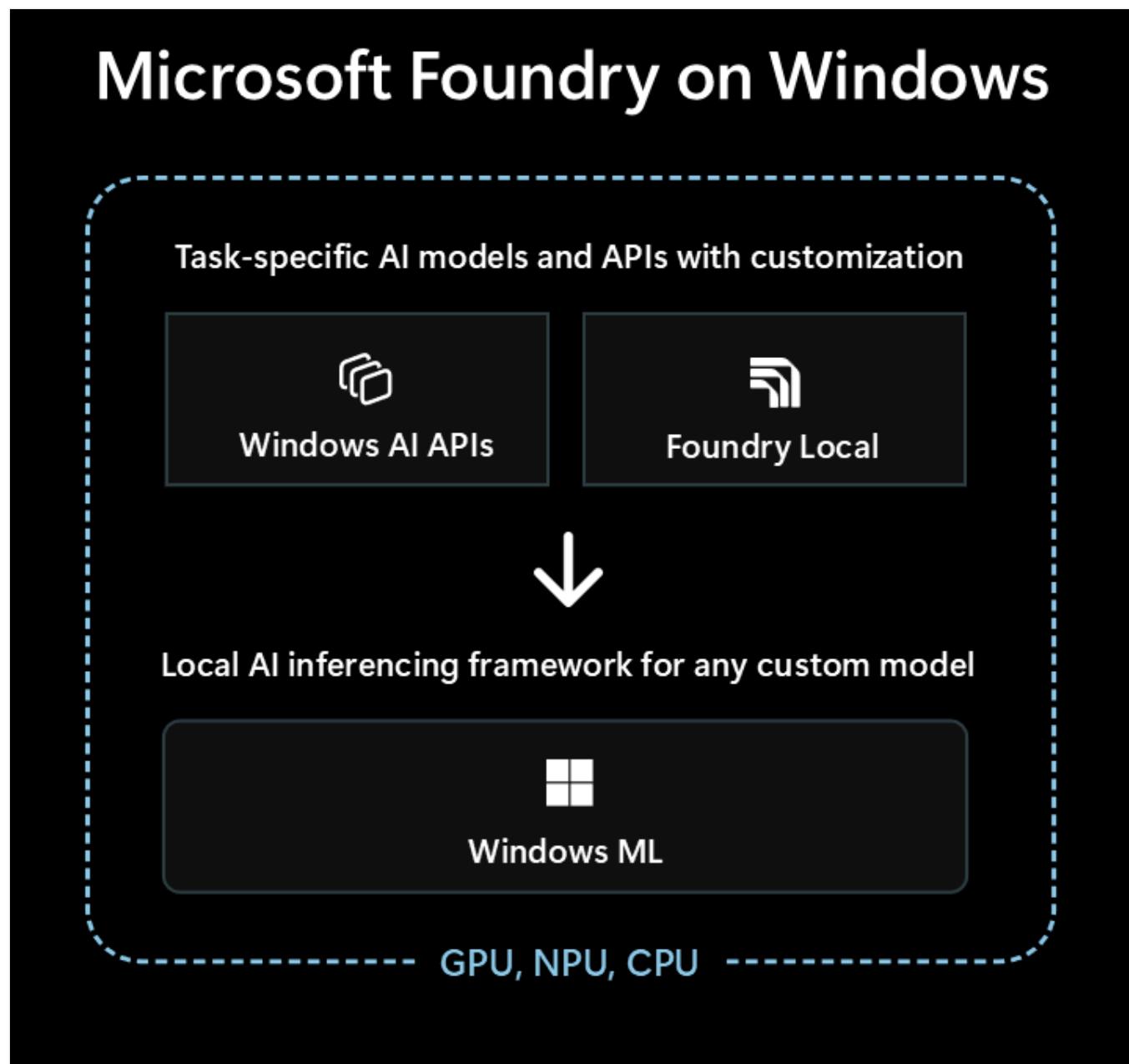
Use local AI with Microsoft Foundry on Windows

Microsoft Foundry on Windows is the premier solution for developers looking to integrate local AI capabilities into their Windows apps.

Microsoft Foundry on Windows provides developers with...

- Ready-to-use AI models and APIs via [Windows AI APIs](#) and [Foundry Local](#)
- AI inferencing framework to run any model locally via [Windows ML](#)

Regardless of whether you're new to AI, or an experienced Machine Learning (ML) expert, Microsoft Foundry on Windows has something for you.



Ready-to-use AI models and APIs

Your app can effortlessly use the following local AI models and APIs in less than an hour. Distribution and runtime of the model files is handled by Microsoft, and the models are shared across apps. Using these models and APIs only takes a handful of lines of code, zero ML-expertise needed.

[+] Expand table

Model type or API	What is it	Options and supported devices
Large Language Models (LLMs)	Generative text models	Phi Silica via AI APIs (supports fine-tuning) or 20+ OSS LLM models via Foundry Local See Local LLMs to learn more.
Image Description	Get a natural-language text description of an image	Image Description via AI APIs (Copilot+ PCs)
Image Foreground Extractor	Segment the foreground of an image	Image Foreground Extractor via AI APIs (Copilot+ PCs)
Image Generation	Generate images from text	Image Generation via AI APIs (Copilot+ PCs)
Image Object Erase	Erase objects from images	Image Object Erase via AI APIs (Copilot+ PCs)
Image Object Extractor	Segment specific objects in an image	Image Object Extractor via AI APIs (Copilot+ PCs)
Image Super Resolution	Increase the resolution of images	Image Super Resolution via AI APIs (Copilot+ PCs)
Semantic Search	Semantically search text and images	App Content Search via AI APIs (Copilot+ PCs)
Speech Recognition	Convert speech to text	Whisper via Foundry Local or Speech Recognition via Windows SDK See Speech Recognition to learn more.
Text Recognition (OCR)	Recognize text from images	OCR via AI APIs (Copilot+ PCs)
Video Super Resolution (VSR)	Increase the resolution of videos	Video Super Resolution via AI APIs (Copilot+ PCs)

Using other models with Windows ML

You can use a wide variety of models from Hugging Face or other sources, or even train your own models, and run those locally on Windows 10+ PCs using [Windows ML](#) (*model compatibility and performance will vary based on device hardware*).

See [find or train models for use with Windows ML](#) to learn more.

Which option to start with

Follow this decision tree to select the best approach for your application and scenario:

1. Check if the built-in [Windows AI APIs](#) cover your scenario and you're targeting Copilot+ PCs. This is the fastest path to market with minimal development effort.
2. If Windows AI APIs don't have what you need, or you need to support Windows 10+, consider [Foundry Local](#) for LLM or voice-to-text scenarios.
3. If you need custom models, want to leverage existing models from Hugging Face or other sources, or have specific model requirements that aren't covered by the above options, [Windows ML](#) gives you the flexibility to find or train your own models.

Your app can also use a combination of all three of these technologies.

Technologies available for local AI

The following technologies are available in Microsoft Foundry on Windows:

[\[\] Expand table](#)

	Windows AI APIs	Foundry Local	Windows ML
What is it	Ready-to-use AI models and APIs across a variety of task types, optimized for Copilot+ PCs	Ready-to-use LLMs and voice-to-text models	ONNX Runtime framework for running models you find or train
Supported devices	Copilot+ PCs	All Windows 10+ PCs and cross-platform <i>(Performance varies based on available hardware, not all models available)</i>	All Windows 10+ PCs, and cross-platform via open-source ONNX Runtime <i>(Performance varies based on available hardware)</i>
Model types and APIs available	LLM Image Description Image Foreground Extractor Image Generation	LLMs (multiple) voice-to-text	Find or train your own models

	Windows AI APIs	Foundry Local	Windows ML
	Image Object Erase Image Object Extractor Image Super Resolution Semantic Search Text Recognition (OCR) Video Super Resolution	Browse 20+ available models	
Model distribution	Hosted by Microsoft, acquired at runtime, and shared across apps	Hosted by Microsoft, acquired at runtime, and shared across apps	Distribution handled by your app (app libraries can share models across apps)
Learn more	Read the AI APIs docs	Read the Foundry Local docs	Read the Windows ML docs

Microsoft Foundry on Windows also includes developer tooling such as [AI Toolkit for Visual Studio Code](#) and [AI Dev Gallery](#) that will help you be successful building AI capabilities.

[AI Toolkit for Visual Studio Code](#) is a VS Code Extension that enables you to download and run AI models locally, including access to hardware-acceleration for better performance and scale through [DirectML](#). The AI Toolkit can also help you with:

- Testing models in an intuitive playground or in your application with a REST API.
- Fine-tuning your AI model, both locally or in the cloud (on a virtual machine) to create new skills, improve reliability of responses, set the tone and format of the response.
- Fine-tuning popular small-language models (SLMs), like [Phi-3](#) and [Mistral](#).
- Deploy your AI feature either to the cloud or with an application that runs on a device.
- Leverage hardware acceleration for better performance with AI features using DirectML. DirectML is a low-level API that enables your Windows device hardware to accelerate the performance of ML models using the device GPU or NPU. Pairing DirectML with the ONNX Runtime is typically the most straightforward way for developers to bring hardware-accelerated AI to their users at scale. Learn more: [DirectML Overview](#).
- Quantize and validate a model for use on NPU by using the model conversion capabilities

Ideas for leveraging local AI

A few ways that Windows apps can leverage local AI to enhance their functionality and user experience include:

- Apps can [use Generative AI LLM models](#) to understand complex topics to summarize, rewrite, report on, or expand.
- Apps can [use LLM models](#) to transform free-form content into a structured format that your app can understand.

- Apps can use [Semantic Search models](#) that allow users to search for content by meaning and quickly find related content.
- Apps can use natural language processing models to reason over complex natural language requirements, and plan and execute actions to accomplish the user's ask.
- Apps can use image manipulation models to intelligently modify images, erase or add subjects, upscale, or generate new content.
- Apps can use predictive diagnostic models to help identify and predict issues and help guide the user or do it for them.

Using Cloud AI Models

If using local AI features isn't the right path for you, [using Cloud AI models and resources](#) can be a solution.

Use Responsible AI practices

Whenever you are incorporating AI features in your Windows app, we **highly recommend** following the [Developing Responsible Generative AI Applications and Features on Windows](#) guidance.

Last updated on 01/24/2026

Ready-to-use local LLMs in Microsoft Foundry on Windows

Microsoft Foundry on Windows provides multiple ready-to-use local Large Language Models (LLMs) that you can integrate into your Windows applications.

Ready-to-use LLMs

Your app can effortlessly use the following local LLMs in less than an hour. Distribution of the LLM is handled by Microsoft, and the models are shared across apps. Using these LLMs only takes a handful of lines of code, zero ML-expertise needed.

 Expand table

What is it	Supported devices	Docs
Phi Silica The same on-device LLM that inbox Windows experiences use	Copilot+ PCs (NPU)	Learn more
20+ open-source LLMs Choose from over 20+ available OSS LLM models	Windows 10+ <i>(Performance varies, not all models available on all devices)</i>	Learn more

Fine-tune local LLMs

If the above ready-to-use LLMs don't work for your scenario, you can fine-tune LLMs for your scenario. This option requires some work to build a fine-tuning training dataset, but is less work than training your own model.

- Phi Silica: See [LoRA Fine-Tuning for Phi Silica](#) to get started.

Use LLMs from Hugging Face or other sources

You can use a wide variety of LLMs from Hugging Face or other sources, and run those locally on Windows 10+ PCs using [Windows ML](#) (*model compatibility and performance will vary based on device hardware*). This option can be more complex and may take more time compared to the ready-to-use local LLMs.

See [find or train models for use with Windows ML](#) to learn more.

Last updated on 01/24/2026

Get started with Phi Silica

Important

The Phi Silica APIs are part of a Limited Access Feature (see [LimitedAccessFeatures class](#)). For more information or to request an unlock token, please use the [LAF Access Token Request Form](#).

Phi Silica is a powerful NPU-tuned local language model that provides many capabilities found in Large Language Models (LLMs). The model employs a technique called speculative decoding to accelerate text generation using a smaller draft model that can propose multiple token sequences and be validated in parallel by the main model.

Note

Phi Silica features are not available in China.

Phi Silica is optimized for efficiency and performance on Windows Copilot+ PCs and can be integrated into your Windows apps through the Windows AI APIs in the Windows App SDK.

This level of optimization is not available in other versions of Phi.

For API details, see:

- [microsoft.windows.ai](#)
- [microsoft.windows.ai.imaging](#)
- [microsoft.windows.ai.text](#)

Integrate Phi Silica

With a local Phi Silica language model you can generate text responses to user prompts. First, ensure you have the pre-requisites and models available on your device as outlined in [Getting Started with Windows AI APIs](#).

Specify the required namespaces

To use Phi Silica, make sure you are using the required namespaces:

C#

```
using Microsoft.Windows.AI;
using Microsoft.Windows.AI.Text;
```

C++/WinRT

```
#include "winrt/Microsoft.Windows.AI.Text.h"
using namespace Microsoft::Windows::AI;
using namespace Microsoft::Windows::AI::Text;
```

Generate a response

This example shows how to generate a response to a Q&A prompt with custom content moderation (see [Content Moderation with the Windows AI APIs](#)).

1. Ensure the language model is available by calling the [GetReadyState](#) method and waiting for the [EnsureReadyAsync](#) method to return successfully.
2. Once the language model is available, create a [LanguageModel](#) object to reference it.
3. Submit a string prompt to the model using the [GenerateResponseAsync](#) method, which returns the complete result.

C#

```
if (LanguageModel.GetReadyState() == AIFeatureReadyState.NotReady)
{
    var op = await LanguageModel.EnsureReadyAsync();
}

using LanguageModel languageModel = await LanguageModel.CreateAsync();

string prompt = "Provide the molecular formula for glucose.";

LanguageModelOptions options = new LanguageModelOptions();
ContentFilterOptions filterOptions = new ContentFilterOptions();
filterOptions.PromptMaxAllowedSeverityLevel.Violent = SeverityLevel.Minimum;
options.ContentFilterOptions = filterOptions;

var result = await languageModel.GenerateResponseAsync(prompt, options);

Console.WriteLine(result.Text);
```

C++/WinRT

```
if (LanguageModel::GetReadyState() == AIFeatureReadyState::NotReady)
{
    auto op = LanguageModel::EnsureReadyAsync().get();
}
```

```
auto languageModel = LanguageModel::CreateAsync().get();

const winrt::hstring prompt = L"Provide the molecular formula for glucose.';

LanguageModelResponseResult result =
languageModel.GenerateResponseAsync(prompt).get();
std::cout << result.Text().c_str() << std::endl;
```

The response generated by this example is:

Output

C6H12O6

Text Intelligence Skills

Phi Silica includes built-in text transformation capabilities (known as Text Intelligence Skills) that can deliver structured, concise, and user-friendly responses through predefined formatting using a local language model.

Supported skills include:

- Text-to-table: Formats the prompt response into a structured table format, when appropriate.
- Summarize: Returns a concise summary of the prompt text.
- Rewrite: Rephrases the prompt text to optimize clarity, readability, and, when specified, tone (or style).

The following steps describe how to use Text Intelligence Skills.

1. Create a `LanguageModel` object

This object references the local Phi Silica language model (remember to confirm that the Phi Silica model is available on the device).

2. Instantiate the skill-specific object

Choose the appropriate class based on the skill you want to apply and pass the `LanguageModel` instance as a parameter.

3. Call the method to perform the skill

Each skill exposes an asynchronous method that processes the input and returns a formatted result.

4. Handle the response

The result is returned as a typed object, which you can print or log as needed.

This example demonstrates the text summarizing skill.

1. Create a [LanguageModel](#) instance (`languageModel`).
2. Pass that [LanguageModel](#) to the [TextSummarizer](#) constructor.
3. Pass some text to the [SummarizeAsync](#) method and print the result.

C#

```
using namespace Microsoft::Windows::AI::Text;

using LanguageModel languageModel = await LanguageModel.CreateAsync();

var textSummarizer = new TextSummarizer(languageModel);
string text = @"This is a large amount of text I want to have summarized.";
var result = await textSummarizer.SummarizeAsync(text);

Console.WriteLine(result.Text);
```

C++/WinRT

```
using namespace Microsoft::Windows::AI::Text;

auto languageModel = LanguageModel::CreateAsync().get();
auto textSummarizer = TextSummarizer(languageModel);
std::string prompt = "This is a large amount of text I want to have summarized.";
auto result = textSummarizer.SummarizeAsync(prompt);

std::wcout << result.get().Text() << std::endl;
```

Responsible AI

We've followed core principles and practices described in the [Microsoft Responsible AI Standards](#) to ensure these APIs are trustworthy, secure, and built responsibly. For more details on implementing AI features in your app, see [Responsible Generative AI Development on Windows](#).

See also

- [AI Dev Gallery](#)
- [Windows AI API samples](#)

Phi Silica walkthrough

ⓘ Important

The Phi Silica APIs are part of a Limited Access Feature (see [LimitedAccessFeatures class](#)). For more information or to request an unlock token, please use the [LAF Access Token Request Form](#).

This short tutorial walks through the [Windows AI API sample for .NET MAUI](#).

ⓘ Note

Phi Silica features are not available in China.

Prerequisites

Complete the steps for .NET MAUI described in the [Get started building an app with Windows AI APIs](#).

Introduction

This sample shows how to use various Windows AI APIs, including [LanguageModel](#) for text generation and [ImageScaler](#) for scaling and sharpening images.

The sample includes the following four files:

1. MauiWindowsAISample.csproj: Adds the required Windows App SDK package reference for the Windows AI APIs and sets the necessary [TargetFramework](#) for Windows.
2. Platforms/Windows/MainPage.cs: Implements partial methods from the shared [MainPage](#) class that show and handle the text generation and image scaling functionality.
3. MainPage.xaml: Defines controls for showing text generation and image scaling.
4. MainPage.xaml.cs: Defines partial methods that [MainPage.cs](#) implements.

In the second file listed above, you'll find the following function, which demonstrates text summarization functionality.

1. Create a [LanguageModel](#) instance (`languageModel1`).
2. Pass that [LanguageModel](#) to the [TextSummarizer](#) constructor.
3. Pass some text to the [SummarizeAsync](#) method and print the result.

C#

```
using Microsoft.Windows.AI;

using LanguageModel languageModel = await LanguageModel.CreateAsync();

string prompt = "This is a large amount of text I want to have summarized.';

LanguageModelOptions options = new LanguageModelOptions {
    Skill = LanguageModelSkill.Summarize
};

var result = await languageModel.GenerateResponseAsync(options, prompt);

Console.WriteLine(result.Text);
```

Build and run the sample

1. Clone the [WindowsAppSDK-Samples](#) repo.
2. Switch to the "release/experimental" branch.
3. Navigate to the [Samples/WindowsAIFoundry/cs-maui](#) folder.
4. Open MauiWindowsAISample.sln in Visual Studio 2022.
5. Ensure the debug toolbar has "Windows Machine" set as the target device.
6. Press F5 or select "Start Debugging" from the Debug menu to run the sample (the sample can also be run without debugging by selecting "Start Without Debugging" from the Debug menu or Ctrl+F5).
7. Click one of the "Scale" buttons to scale the image, or enter a text prompt and click the "Generate" button to generate a text response.

See also

- [AI Dev Gallery](#)
- [Windows AI API samples](#)

Last updated on 11/18/2025

LoRA Fine-Tuning for Phi Silica

Low Rank Adaptation (LoRA) can be utilized to fine-tune the [Phi Silica model](#) to enhance its performance for your specific use-case. By using LoRA to optimize Phi Silica, Microsoft Windows local language model, you can achieve more accurate results. This process involves training a LoRA adapter and then applying it during inference to improve the model's accuracy.

(!) Note

Phi Silica features are not available in China.

Prerequisites

- You have identified a use case for enhancing the response of Phi Silica.
- You have chosen an evaluation criteria to decide what a 'good response' is.
- You have tried the Phi Silica APIs and they do not meet your evaluation criteria.

Train your adapter

To train a LoRA adapter for fine-tuning the Phi Silica model with Windows 11, you must first generate a dataset that the training process will use.

Generate a dataset for use with a LoRA adapter

To generate a dataset, you will need to split data into two files:

- `train.json` – Used for training the adapter.
- `test.json` – Used for evaluating the adapter's performance during and after training.

Both files must use the JSON format, where each line is a separate JSON object representing a single sample. Each sample should contain a list of messages exchanged between a user and an assistant.

Every message object requires two fields:

- `content`: the text of the message.
- `role`: either `"user"` or `"assistant"`, indicating the sender.

See the following examples:

JSON

```
{"messages": [{"content": "Hello, how do I reset my password?", "role": "user"}, {"content": "To reset your password, go to the settings page and click 'Reset Password'.", "role": "assistant"}]}

{"messages": [{"content": "Can you help me find nearby restaurants?", "role": "user"}, {"content": "Sure! Here are some restaurants near your location: ...", "role": "assistant"}]}

{"messages": [{"content": "What is the weather like today?", "role": "user"}, {"content": "Today's forecast is sunny with a high of 25°C.", "role": "assistant"}]}
```

Training tips:

- There is no comma needed at the end of each sample line.
- Include as many high-quality and diverse examples as possible. For best results, collect at least a few thousand training samples in your `train.json` file.
- The `test.json` file can be smaller, but should cover the types of interactions you expect your model to handle.
- Create `train.json` and `test.json` files with one JSON object per line, each containing a brief back-and-forth conversation between a user and an assistant. The quality and quantity of your data will greatly affect the effectiveness of your LoRA adapter.

Training a LoRA adapter in the AI Toolkit

To train a LoRA adapter using the [AI Toolkit for Visual Studio Code](#), you will first need the follow required prerequisites:

- [Azure subscription](#) with available quota in [Azure Container Apps](#).
 - We recommend using A100 GPUs or better to efficiently run a fine-tuning job.
 - [Check that you have available quota in the Azure Portal](#). If you'd like help finding your quota, see [View Quotas](#).
- You will need to [install Visual Studio Code](#) if you don't already have it.

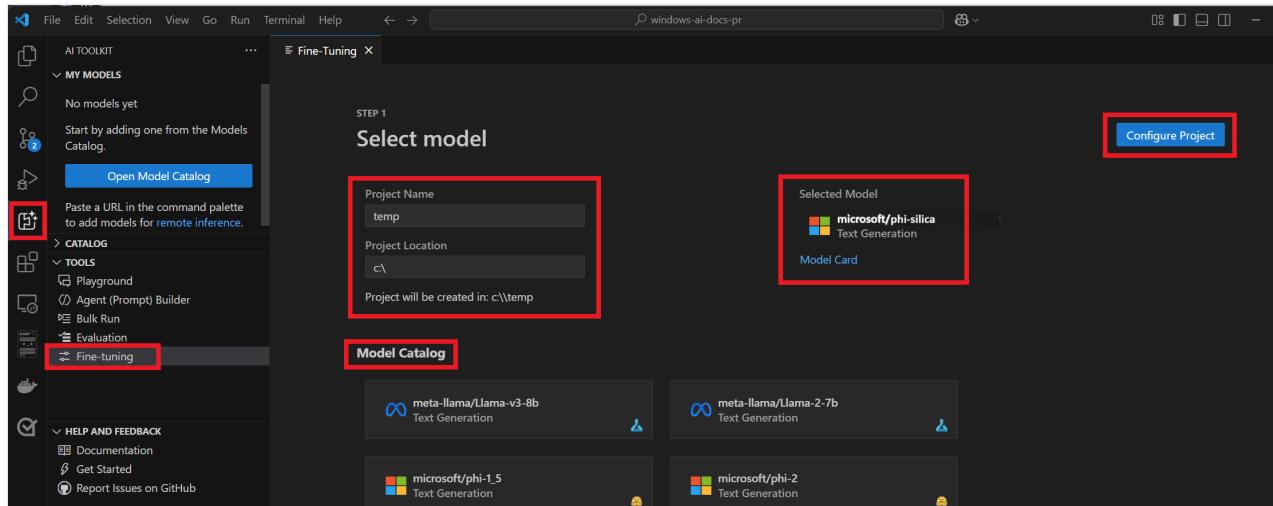
To install AI Toolkit for Visual Studio Code:

1. [Download the AI Toolkit extension in Visual Studio Code](#)
2. Once the AI Toolkit extension is downloaded, you will be able to access it from the left toolbar pane inside Visual Studio Code.
3. Navigate to **Tools > Fine-tuning**.

4. Enter a Project Name and Project Location.

5. Select "microsoft/phi-silica" from the Model Catalog.

6. Select "Configure project".



7. Select the latest version of Phi Silica.

8. Under Data > Training Dataset name and Test Dataset name, select your `train.json` and `test.json` files.

9. Select "Generate Project" - a new VS Code windows will open.

10. Ensure that the correct workload profile is selected in your bicep file so that the Azure job deploys and runs correctly. Add the following under `workloadProfiles`:

```
Bicep
{
    workloadProfileType: 'Consumption-GPU-NC24-A100'
    name: 'GPU'
}
```

11. Select "New Fine-tuning Job" and enter a name for the job.

12. A dialog box will appear asking you to select the Microsoft account with which to access your Azure subscription.

13. Once your account is selected, you will need to select a Resource Group from the subscription dropdown menu.

14. You will now see that your fine-tuning job has successfully started, along with a Job Status. Once the job has completed, you will have the option to download the newly

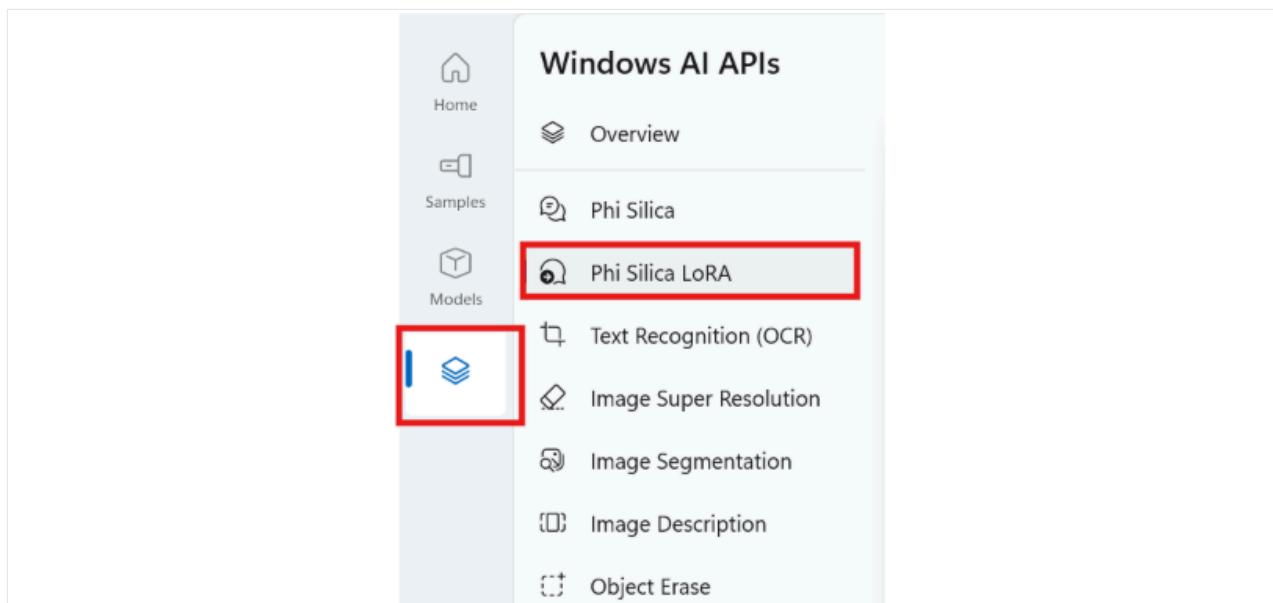
trained LoRA adapter by selecting the "Download" button. It typically takes an average of 45 - 60 minutes for a fine-tuning job to complete.

Inference

Training is the initial phase where the AI model learns from a large dataset, recognizing patterns and correlations. Inference, on the other hand, is the application phase where a trained model (Phi Silica in our case) uses new data (our LoRA adapter) to make predictions or decisions with which to generate more customized output.

To apply the trained LoRA adapter:

1. Use the [AI Dev Gallery app](#). AI Dev Gallery is an app which allows you to experiment with local AI models and API, in addition to viewing and exporting sample code. [Learn more about AI Dev Gallery](#).
2. Once you have installed AI Dev Gallery, open the app and select the "AI APIs" tab, then select "Phi Silica LoRA."



3. Select the adapter file. The default location for these to be stored is: `Desktop / lora_lab / trainedLora`.
4. Complete the "System Prompt" and "Prompt" fields. Then select "Generate" to see the difference between Phi Silica with and without the LoRA adapter.
5. Experiment with the Prompt and System Prompt to see how this makes a difference to your output.
6. Select "Export Sample" to download a standalone Visual Studio solution that only contains this sample code.

Generate responses

Once you've tested your new LoRA adapter using AI Dev Gallery, you can add the adapter to your Windows app using the code sample below.

C#

```
C#  
  
using Microsoft.Windows.AI.Text;  
using Microsoft.Windows.AI.Text.Experimental;  
  
// Path to the LoRA adapter file  
string adapterFilePath = "C:/path/to/adapter/file.safetensors";  
  
// Prompt to be sent to the LanguageModel  
string prompt = "How do I add a new project to my Visual Studio solution?";  
  
// Wait for LanguageModel to be ready  
if (LanguageModel.GetReadyState() == AIFeatureReadyState.NotReady)  
  
{  
    var languageModelDeploymentOperation = LanguageModel.EnsureReadyAsync();  
    await languageModelDeploymentOperation;  
}  
  
// Create the LanguageModel session  
var session = LanguageModel.CreateAsync();  
  
// Create the LanguageModelExperimental  
var languageModelExperimental = new LanguageModelExperimental(session);  
  
// Load the LoRA adapter  
LowRankAdaptation loraAdapter =  
languageModelExperimental.LoadAdapter(adapterFilePath);  
  
// Set the adapter in LanguageModelOptionsExperimental  
LanguageModelOptionsExperimental options = new LanguageModelOptionsExperimental  
  
{  
    LoraAdapter = loraAdapter  
};  
  
// Generate a response with the LoRA adapter provided in the options  
var response = await languageModelExperimental.GenerateResponseAsync(prompt,  
options);
```

Responsible AI - Risks and limitations of fine-tuning

When customers fine-tune Phi Silica, it can improve model performance and accuracy on specific tasks and domains, but it can also introduce new risks and limitations that customers should be aware of. Some of these risks and limitations are:

- **Data quality and representation:** The quality and representativeness of the data used for fine-tuning can affect the model's behavior and outputs. If the data is noisy, incomplete, outdated, or if it contains harmful content like stereotypes, the model can inherit these issues and produce inaccurate or harmful results. For example, if the data contains gender stereotypes, the model can amplify them and generate sexist language. Customers should carefully select and pre-process their data to ensure that it is relevant, diverse, and balanced for the intended task and domain.
- **Model robustness and generalization:** The model's ability to handle diverse and complex inputs and scenarios can decrease after fine-tuning, especially if the data is too narrow or specific. The model can overfit to the data and lose some of its general knowledge and capabilities. For example, if the data is only about sports, the model can struggle to answer questions or generate text about other topics. Customers should evaluate the model's performance and robustness on a variety of inputs and scenarios and avoid using the model for tasks or domains that are outside its scope.
- **Regurgitation:** While your training data is not available to Microsoft or any third-party customers, poorly fine-tuned models may regurgitate, or directly repeat, training data. Customers are responsible for removing any PII or otherwise protected information from their training data and should assess their fine-tuned models for over-fitting or otherwise low-quality responses. To avoid regurgitation, customers are encouraged to provide large and diverse datasets.
- **Model transparency and explainability:** The model's logic and reasoning can become more opaque and difficult to understand after fine-tuning, especially if the data is complex or abstract. A fine-tuned model can produce outputs that are unexpected, inconsistent, or contradictory, and customers may not be able to explain how or why the model arrived at those outputs. For example, if the data is about legal or medical terms, the model can generate outputs that are inaccurate or misleading, and customers may not be able to verify or justify them. Customers should monitor and audit the model's outputs and behavior and provide clear and accurate information and guidance to the end-users of the model.

See also

- Phi Silica
-

Last updated on 08/21/2025

Image Description

Important

Image Description is currently unavailable in China.

The Image Description APIs provide the ability to generate various types of text descriptions for an image.

For API details, see [API ref for AI imaging features](#).

For content moderation details, see [Content safety with generative AI APIs](#).

Important

Package Manifest Requirements: To use Windows AI imaging APIs, your app must be packaged as an MSIX package with the `SystemAIModels` capability declared in your `Package.appxmanifest`. Additionally, ensure your manifest's `MaxVersionTested` attribute is set to a recent Windows version (e.g., `10.0.26226.0` or later) to properly support the Windows AI features. Using older values may cause "Not declared by app" errors when loading the model.

XML

```
<Dependencies>
    <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.17763.0"
MaxVersionTested="10.0.26226.0" />
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17763.0"
MaxVersionTested="10.0.26226.0" />
</Dependencies>
```

Description types

The following types of text descriptions are supported:

- **Brief** - Provides a description suitable for charts and diagrams.
- **Detailed** - Provides a long description.
- **Diagram** - Provides a short description suitable for an image caption. The default if no value is specified.
- **Accessible** - Provides a long description with details intended for users with accessibility needs.

Limitations

Because these APIs use Machine Learning (ML) models, occasional errors can occur where the text does not describe the image correctly. Therefore, we do not recommend using these APIs for images in the following scenarios:

- Where the images contain potentially sensitive content and inaccurate descriptions could be controversial, such as flags, maps, globes, cultural symbols, or religious symbols.
- When accurate descriptions are critical, such as for medical advice or diagnosis, legal content, or financial documents.

Image Description example

The following example shows how to get a text description for an image based on the specified description type (optional) and level of content moderation (optional).

! Note

The image must be an `ImageBuffer` object as `SoftwareBitmap` is not currently supported (this example demonstrates how to convert `SoftwareBitmap` to `ImageBuffer`).

1. Ensure the Image Description model is available by calling the `GetReadyState` method and then waiting for the `EnsureReadyAsync` method to return successfully.
2. Once the Image Description model is available, create an `ImageDescriptionGenerator` object to reference it.
3. (Optional) Create a `ContentFilterOptions` object and specify your preferred values. If you choose to use default values, you can pass in a null object.
4. Get the image description (`LanguageModelResponse.Response`) by calling the `DescribeAsync` method specifying the original image, the `ImageDescriptionKind` (an optional value for the preferred description type), and the `ContentFilterOptions` object (optional).

C#

```
using Microsoft.Graphics.Imaging;
using Microsoft.Windows.Management.Deployment;
using Microsoft.Windows.AI;
using Microsoft.Windows.AI.ContentModeration;
using Windows.Storage.StorageFile;
using Windows.Storage.Streams;
using Windows.Graphics.Imaging;
```

```

if (ImageDescriptionGenerator.GetReadyState() == AIFeatureReadyState.NotReady)
{
    var result = await ImageDescriptionGenerator.EnsureReadyAsync();
    if (result.Status != AIFeatureReadyResultState.Success)
    {
        throw result.ExtendedError;
    }
}

ImageDescriptionGenerator imageDescriptionGenerator = await
ImageDescriptionGenerator.CreateAsync();

// Convert already available softwareBitmap to ImageBuffer.
ImageBuffer inputImage = ImageBuffer.CreateCopyFromBitmap(softwareBitmap);

// Create content moderation thresholds object.
ContentFilterOptions filterOptions = new ContentFilterOptions();
filterOptions.PromptMinSeverityLevelToBlock.ViolentContentSeverity =
SeverityLevel.Medium;
filterOptions.ResponseMinSeverityLevelToBlock.ViolentContentSeverity =
SeverityLevel.Medium;

// Get text description.
LanguageModelResponse languageModelResponse = await
imageDescriptionGenerator.DescribeAsync(inputImage,
ImageDescriptionScenario.Caption, filterOptions);
string response = languageModelResponse.Response;

```

C++/WinRT

```

#include <winrt/Microsoft.Graphics.Imaging.h>
#include <winrt/Microsoft.Windows.AI.Imaging.h>
#include <winrt/Microsoft.Windows.AI.ContentSafety.h>
#include <winrt/Microsoft.Windows.AI.h>
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Graphics.Imaging.h>
#include <winrt/Windows.Storage.Streams.h>
#include <winrt/Windows.Storage.StorageFile.h>

using namespace winrt::Microsoft::Graphics::Imaging;
using namespace winrt::Microsoft::Windows::AI;
using namespace winrt::Microsoft::Windows::AI::ContentSafety;
using namespace winrt::Microsoft::Windows::AI::Imaging;
using namespace winrt::Windows::Foundation;
using namespace winrt::Windows::Graphics::Imaging;
using namespace winrt::Windows::Storage::Streams;
using namespace winrt::Windows::Storage::StorageFile;

if (ImageDescriptionGenerator::GetReadyState() == AIFeatureReadyState::NotReady)
{
    auto loadResult = ImageDescriptionGenerator::EnsureReadyAsync().get();

```

```
if (loadResult.Status() != AIFeatureReadyResultState::Success)
{
    throw winrt::_hresult_error(loadResult.ExtendedError());
}
}

ImageDescriptionGenerator imageDescriptionGenerator =
    ImageDescriptionGenerator::CreateAsync().get();

// Convert already available softwareBitmap to ImageBuffer.
auto inputBuffer =
Microsoft::Graphics::Imaging::ImageBuffer::CreateForSoftwareBitmap(softwareBitmap);

// Create content moderation thresholds object.

ContentFilterOptions contentFilter{};
contentFilter.PromptMaxAllowedSeverityLevel().Violent(SeverityLevel::Medium);
contentFilter.ResponseMaxAllowedSeverityLevel().Violent(SeverityLevel::Medium);

// Get text description.
auto response = imageDescriptionGenerator.DescribeAsync(inputBuffer,
ImageDescriptionKind::BriefDescription, contentFilter).get();
string text = response.Description();
```

See also

- [AI Imaging overview](#)
- [AI Dev Gallery ↗](#)
- [Windows AI API samples ↗](#)

Last updated on 01/24/2026

Image Foreground Extractor

You can use [ImageForegroundExtractor](#) to segment the foreground of an input image and enable features such as background removal and sticker generation.

The returned mask is in greyscale-8 format. Pixel values range from 0 to 255, where 0 represents background pixels, 255 represents foreground pixels, and intermediate values indicate a blend of foreground and background pixels.

For API details, see [API ref for AI imaging features](#).

For content moderation details, see [Content safety with generative AI APIs](#).

ⓘ Important

Package Manifest Requirements: To use Windows AI imaging APIs, your app must be packaged as an MSIX package with the `systemAIModels` capability declared in your `Package.appxmanifest`. Additionally, ensure your manifest's `MaxVersionTested` attribute is set to a recent Windows version (e.g., `10.0.26226.0` or later) to properly support the Windows AI features. Using older values may cause "Not declared by app" errors when loading the model.

XML

```
<Dependencies>
    <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.17763.0"
        MaxVersionTested="10.0.26226.0" />
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17763.0"
        MaxVersionTested="10.0.26226.0" />
</Dependencies>
```

Generating a mask from a bitmap image

1. Call [GetReadyState](#) and wait for [EnsureReadyAsync](#) to complete successfully to confirm that the `ImageForegroundExtractor` object is ready.
2. After the model is ready, call [CreateAsync](#) to instantiate an `ImageForegroundExtractor` object.
3. Call [GetMaskFromSoftwareBitmap](#) with the input image to generate the foreground mask.

C#

```
using Microsoft.Windows.AI.Imaging;
using Microsoft.Windows.AI;
```

```

if (ImageForegroundExtractor.GetReadyState() == AIFeatureReadyState.NotReady)
{
    var result = await ImageForegroundExtractor.EnsureReadyAsync();
    if (result.Status != AIFeatureReadyResultState.Success)
    {
        throw result.ExtendedError;
    }
}

var model = await ImageForegroundExtractor.CreateAsync();

// Insert your own softwareBitmap here.
var foregroundMask = model.GetMaskFromSoftwareBitmap(softwareBitmap);

```

C++/WinRT

```

#include <winrt/Microsoft.Graphics.Imaging.h>
#include <winrt/Microsoft.Windows.AI.Imaging.h>
#include <winrt/Windows.Graphics.Imaging.h>
#include <winrt/Windows.Foundation.h>
using namespace winrt::Microsoft::Graphics::Imaging;
using namespace winrt::Microsoft::Windows::AI::Imaging;
using namespace winrt::Windows::Graphics::Imaging;
using namespace winrt::Windows::Foundation;

if (ImageForegroundExtractor::GetReadyState() == AIFeatureReadyState::NotReady)
{
    auto loadResult = ImageForegroundExtractor::EnsureReadyAsync().get();

    if (loadResult.Status() != AIFeatureReadyResultState::Success)
    {
        throw winrt::hresult_error(loadResult.ExtendedError());
    }
}

auto model = co_await ImageForegroundExtractor::CreateAsync();

// Insert your own softwareBitmap here.
auto foregroundMask = model.GetMaskFromSoftwareBitmap(softwareBitmap);

```

See also

- [AI Imaging overview](#)
- [AI Dev Gallery ↗](#)
- [Windows AI API samples ↗](#)

Get started with AI Image Generation

Microsoft Foundry on Windows supports AI Image Generation features through a set of artificial intelligence-backed, Stable Diffusion-powered (open-source AI model used for processing images) APIs that ship in the Windows App SDK. You can use these APIs in your Windows apps to create, transform, and enhance images and photos using natural language prompts and on-device generative models.

AI Image Generation is optimized for efficiency and performance on Windows Copilot+ PCs.

For API details, see [API ref for AI imaging features](#).

Important

Package Manifest Requirements: To use Windows AI imaging APIs, your app must be packaged as an MSIX package with the `systemAIModels` capability declared in your `Package.appxmanifest`. Additionally, ensure your manifest's `MaxVersionTested` attribute is set to a recent Windows version (e.g., `10.0.26226.0` or later) to properly support the Windows AI features. Using older values may cause "Not declared by app" errors when loading the model.

XML

```
<Dependencies>
  <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.17763.0"
    MaxVersionTested="10.0.26226.0" />
  <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17763.0"
    MaxVersionTested="10.0.26226.0" />
</Dependencies>
```

Prerequisites

- **Windows version:** Windows 11, version 24H2 (build 26100) or later
- **WinAppSDK version:** [Version 2.0 Experimental \(2.0.0-Experimental3\)](#)
- **Hardware:** NPU-enabled PC recommended

What can I do with AI Image Generation?

Use AI Image Generation to turn prompts into visual artifacts. Supported features include:

- **Text-to-Image**

Generate images from descriptive text prompts. Useful for illustrations, design, customized backgrounds, and conceptual visualization.

- **Image-to-Image**

Transform an existing image based on textual guidance while preserving structure. Useful for styling, theming, and other variations.

- **Magic Fill**

Fill masked regions of an image with AI-generated content. Useful for removing objects, repairing regions, and intuitive editing (complex revisions through text prompts instead of manual tools).

- **Coloring Book Style**

Convert images into simplified outlines that you can use for a coloring book or similar educational experience.

- **Restyle**

Apply artistic or visual styles to existing images while preserving structure. Useful for creative filters, artistic modes, or themed transformations.

Examples

Follow these basic steps when using the AI Image Generation APIs.

1. Ensure the model is ready using [EnsureReadyAsync](#).
2. Create an [ImageGenerator](#) instance.
3. Select the appropriate generation workflow (text prompt, image input, or mask).
4. Invoke the corresponding generation method.
5. Receive the output as an [ImageBuffer](#) for viewing, editing, or saving.

Generate an image from a text prompt (Text-to-Image)

This example shows how to generate an image from a text prompt. Specifically, "A beautiful sunset over a mountain lake".

```
C#
```

```
using Microsoft.Windows.AI.Imaging;
using Microsoft.Graphics.Imaging;

public async Task GenerateImageFromText()
```

```

{
    // Check if models are ready
    var readyState = ImageGenerator.GetReadyState();
    if (readyState != AIFeatureReadyState.Ready)
    {
        // Download models if needed
        var result = await ImageGenerator.EnsureReadyAsync();
        if (result.Status != AIFeatureReadyResultState.Success)
        {
            Console.WriteLine("Failed to prepare models");
            return;
        }
    }

    // Create ImageGenerator instance
    using var generator = await ImageGenerator.CreateAsync();

    // Configure generation options
    var options = new ImageGenerationOptions
    {
        MaxInferenceSteps = 6,
        Creativity = 0.8,
        Seed = 42
    };

    // Generate image
    var result = generator.GenerateImageFromTextPrompt("A beautiful sunset over a
mountain lake", options);

    if (result.Status == ImageGeneratorResultStatus.Success)
    {
        var imageBuffer = result.Image;
        // Use the generated image (save to file, display, etc.)
        await SaveImageBufferAsync(imageBuffer, "generated_image.png");
    }
    else
    {
        Console.WriteLine($"Image generation failed: {result.Status}");
    }
}

```

Transform an image style (Image-to-Image)

This example shows how to transform a photograph into an oil painting based on a text prompt. Specifically, "oil painting style, thick brush strokes, artistic".

C#

```

public async Task RestyleImage()
{
    using var generator = await ImageGenerator.CreateAsync();

```

```

// Load input image
var inputImage = await LoadImageBufferAsync("photo.jpg");

var options = new ImageGenerationOptions();
var styleOptions = new ImageFromImageGenerationOptions
{
    Style = ImageFromImageGenerationStyle.Restyle,
    ColorPreservation = 0.7f
};

var result = generator.GenerateImageFromImageBuffer(
    inputImage,
    "oil painting style, thick brush strokes, artistic",
    options,
    styleOptions);

if (result.Status == ImageGeneratorResultStatus.Success)
{
    await SaveImageBufferAsync(result.Image, "restyled_image.png");
}
}

```

Transform an image style (Image-to-Image complex)

This example shows how to transform a photograph into an oil painting based on a text prompt. Specifically, "An oil painting, thick brush strokes, rich color palette, traditional canvas texture, realistic lighting, classical fine art style, layered paint, high detail, dramatic contrast, impasto, textured canvas".

C#

```

using Microsoft.Windows.AI.Imaging;

public async Task CreateImageFromPrompt()
{
    using ImageGenerator model = await ImageGenerator.CreateAsync();

    // Using default values
    var options = new ImageGenerationOptions();

    // Set ImageFromImageGenerationOptions fields
    var imageFromImageOptions = new ImageFromImageGenerationOptions();
    imageFromImageOptions.Style = ImageFromImageGenerationStyle.Restyle;
    imageFromImageOptions.ColorPreservation = 0.5f; // range [0.0f, 1.0f]

    // Load an input image buffer
    using var inputImage = await
    Utils.LoadSampleImageBufferAsync("sdxl_input_horse.png");

    var textPrompt = "An oil painting, thick brush strokes, rich color palette,
traditional canvas texture, realistic lighting, classical fine art style, layered

```

```

paint, high detail, dramatic contrast, impasto, textured canvas";

var result = model.GenerateImageFromImageBuffer(inputImage, textPrompt, options,
imageFromImageOptions);
if (result.Status == ImageGeneratorResultStatus.Success)
{
    // Image generated successfully
    var imageBuffer = result.Image;
    // Process the imageBuffer as needed, e.g., save to file or display
}
else
{
    // Handle error cases based on result.Status
    Console.WriteLine($"Image generation failed with status: {result.Status}");
}
}

```

Magic Fill with Mask

This example shows how to use a mask to fill a region of an image. Specifically, "a red sports car".

C#

```

public async Task FillMaskedRegion()
{
    using var generator = await ImageGenerator.CreateAsync();

    var inputImage = await LoadImageBufferAsync("scene.jpg");
    var maskImage = await LoadImageBufferAsync("mask.png"); // GRAY8 format

    var options = new ImageGenerationOptions();

    var result = generator.GenerateImageFromImageBufferAndMask(
        inputImage,
        maskImage,
        "a red sports car",
        options);

    if (result.Status == ImageGeneratorResultStatus.Success)
    {
        await SaveImageBufferAsync(result.Image, "filled_image.png");
    }
}

```

Generate coloring-book style image

This example shows how to generate an image in a coloring book style. Specifically, a "Cat in spaceship".

```
c#  
  
using Microsoft.Windows.AI.Imaging;  
  
public async Task CreateImageFromPrompt()  
{  
    using ImageGenerator model = await ImageGenerator.CreateAsync();  
  
    // Using default values  
    var options = new ImageGenerationOptions();  
  
    // Set ImageFromTextGenerationOptions fields  
    var imageFromTextOptions = new ImageFromTextGenerationOptions();  
    imageFromTextOptions.Style = ImageFromTextGenerationStyle.ColoringBook;  
  
    var result = model.GenerateImageFromTextPrompt("Cat in spaceship", options,  
imageFromTextOptions);  
    if (result.Status == ImageGeneratorResultStatus.Success)  
    {  
        // Image generated successfully  
        var imageBuffer = result.Image;  
        // Process the imageBuffer as needed, e.g., save to file or display  
    }  
    else  
    {  
        // Handle error cases based on result.Status  
        Console.WriteLine($"Image generation failed with status: {result.Status}");  
    }  
}
```

Generate image using custom ImageGenerationOptions parameters

This example shows how to generate an image based on a set of content filters and restrictions. Specifically, a "Cat in spaceship" using a [TextContentFilterSeverity](#) of [Low](#) and an [ImageContentFilterSeverity](#) of [Minimum](#).

```
c#  
  
using Microsoft.Windows.AI.Imaging;  
using Microsoft.Windows.AI.ContentSafety;  
  
public async Task CreateImageFromPromptAndCustomOptions()  
{  
    using ImageGenerator model = await ImageGenerator.CreateAsync();  
  
    // Using default values  
    var options = new ImageGenerationOptions();  
  
    // Set custom ImageGenerationOptions fields  
    options.MaxInferenceSteps = 6;
```

```
options.Creativity = 0.8;
options.Seed = 1234;
ContentFilterOptions contentFilterOptions = new ContentFilterOptions();
contentFilterOptions.PromptMaxAllowedSeverityLevel =
    TextContentFilterSeverity(SeverityLevel.Low);
    contentFilterOptions.ImageMaxAllowedSeverityLevel =
        ImageContentFilterSeverity(SeverityLevel.Minimum);
options.ContentFilterOptions = contentFilterOptions;

var result = model.GenerateImageFromTextPrompt("Cat in spaceship", options);
if (result.Status == ImageGeneratorResultStatus.Success)
{
    // Image generated successfully
    var imageBuffer = result.Image;
    // Process the imageBuffer as needed, e.g., save to file or display
}
else
{
    // Handle error cases based on result.Status
    Console.WriteLine($"Image generation failed with status: {result.Status}");
}
}
```

Responsible AI

Follow responsible AI recommendations, including transparency and user trust, when using these APIs to modify or generate images in your Windows apps. To help users understand the origin and history of generated or modified images, provide Content Credentials as specified by the [Coalition for Content Provenance and Authenticity \(C2PA\)](#) standards.

See [Responsible Generative AI Development on Windows](#) for best practices when implementing AI features in Windows apps.

See also

- [API ref for AI imaging features](#)

Image Object Erase

Object Erase can be used to remove objects from images. The model takes both an image and a greyscale mask indicating the object to be removed, erases the masked area from the image, and replaces the erased area with the image background.

For API details, see [API ref for AI imaging features](#).

For content moderation details, see [Content safety with generative AI APIs](#).

Important

Package Manifest Requirements: To use Windows AI imaging APIs, your app must be packaged as an MSIX package with the `systemAIModels` capability declared in your `Package.appxmanifest`. Additionally, ensure your manifest's `MaxVersionTested` attribute is set to a recent Windows version (e.g., `10.0.26226.0` or later) to properly support the Windows AI features. Using older values may cause "Not declared by app" errors when loading the model.

XML

```
<Dependencies>
  <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.17763.0"
    MaxVersionTested="10.0.26226.0" />
  <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17763.0"
    MaxVersionTested="10.0.26226.0" />
</Dependencies>
```

Image Object Erase example

The following example shows how to remove an object from an image. The example assumes that you already have software bitmap objects (`softwareBitmap`) for both the image and the mask. The mask must be in Gray8 format with each pixel of the area to be removed set to 255 and all other pixels set to 0.

1. Ensure the Image Object Erase model is available by calling the `GetReadyState` method and waiting for the `EnsureReadyAsync` method to return successfully.
2. Once the Image Object Erase model is available, create an `ImageObjectRemover` object to reference it.
3. Finally, submit the image and the mask to the model using the `RemoveFromSoftwareBitmap` method, which returns the final result.

C#

```
using Microsoft.Graphics.Imaging;
using Microsoft.Windows.AI;
using Microsoft.Windows.Management.Deployment;
using Windows.Graphics.Imaging;

if (ImageObjectRemover.GetReadyState() == AIFeatureReadyState.NotReady)
{
    var result = await ImageObjectRemover.EnsureReadyAsync();
    if (result.Status != AIFeatureReadyResultState.Success)
    {
        throw result.ExtendedError;
    }
}
ImageObjectRemover imageObjectRemover = await ImageObjectRemover.CreateAsync();
SoftwareBitmap finalImage = imageObjectRemover.RemoveFromSoftwareBitmap(imageBitmap,
maskBitmap); // Insert your own imagebitmap and maskbitmap
```

C++/WinRT

```
#include <winrt/Microsoft.Graphics.Imaging.h>
#include <winrt/Microsoft.Windows.AI.Imaging.h>
#include <winrt/Windows.Graphics.Imaging.h>
#include <winrt/Windows.Foundation.h>
using namespace winrt::Microsoft::Graphics::Imaging;
using namespace winrt::Microsoft::Windows::AI::Imaging;
using namespace winrt::Windows::Graphics::Imaging;
using namespace winrt::Windows::Foundation;
if (ImageObjectRemover::GetReadyState() == AIFeatureReadyState::NotReady)
{
    auto loadResult = ImageObjectRemover::EnsureReadyAsync().get();

    if (loadResult.Status() != AIFeatureReadyResultState::Success)
    {
        throw winrt::hresult_error(loadResult.ExtendedError());
    }
}

ImageObjectRemover imageObjectRemover = ImageObjectRemover::CreateAsync().get();
// Insert your own imagebitmap and maskbitmap
Windows::Graphics::Imaging::SoftwareBitmap buffer =
    imageObjectRemover.RemoveFromSoftwareBitmap(imageBitmap, maskBitmap);
```

See also

- [AI Imaging overview](#)
- [Image Object Extractor](#) - Use this to generate masks for objects you want to remove
- [AI Dev Gallery](#) ↗
- [Windows AI API samples](#) ↗

Last updated on 01/24/2026

Image Object Extractor

You can use Image Object Extractor to identify specific objects in an image. The model takes both an image and a "hints" object and returns a mask of the identified object.

For API details, see [API ref for AI imaging features](#).

For content moderation details, see [Content safety with generative AI APIs](#).

Important

Package Manifest Requirements: To use Windows AI imaging APIs, your app must be packaged as an MSIX package with the `systemAIModels` capability declared in your `Package.appxmanifest`. Additionally, ensure your manifest's `MaxVersionTested` attribute is set to a recent Windows version (e.g., `10.0.26226.0` or later) to properly support the Windows AI features. Using older values may cause "Not declared by app" errors when loading the model.

XML

```
<Dependencies>
  <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.17763.0"
    MaxVersionTested="10.0.26226.0" />
  <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17763.0"
    MaxVersionTested="10.0.26226.0" />
</Dependencies>
```

Providing hints

You can provide hints through any combination of the following:

- Coordinates for points that belong to what you're identifying.
- Coordinates for points that don't belong to what you're identifying.
- A coordinate rectangle that encloses what you're identifying.

The more hints you provide, the more precise the model can be. Follow these hint guidelines to minimize inaccurate results or errors.

- Avoid using multiple rectangles in a hint as they can produce an inaccurate mask.
- Avoid using exclude points exclusively without include points or a rectangle.
- Don't specify more than the supported maximum of 32 coordinates (1 for a point, 2 for a rectangle) as this will return an error.

The returned mask is in greyscale-8 format with the pixels of the mask for the identified object having a value of 255 (all others having a value of 0).

Image Object Extractor example

The following examples show ways to identify an object within an image. The examples assume that you already have a software bitmap object (`softwareBitmap`) for the input.

1. Ensure the Image Object Extractor model is available by calling the [GetReadyState](#) method and waiting for the [EnsureReadyAsync](#) method to return successfully.
2. Once the Image Object Extractor model is available, create an [ImageObjectExtractor](#) object to reference it.
3. Pass the image to [CreateWithSoftwareBitmapAsync](#).
4. Create an [ImageObjectExtractorHint](#) object. Other ways to create a hint object with different inputs are demonstrated later.
5. Submit the hint to the model using the [GetSoftwareBitmapObjectMask](#) method, which returns the final result.

C#

```
using Microsoft.Graphics.Imaging;
using Microsoft.Windows.AI;
using Microsoft.Windows.Management.Deployment;
using Windows.Graphics.Imaging;

if (ImageObjectExtractor.GetReadyState() == AIFeatureReadyState.NotReady)
{
    var result = await ImageObjectExtractor.EnsureReadyAsync();
    if (result.Status != AIFeatureReadyResultState.Success)
    {
        throw result.ExtendedError;
    }
}

ImageObjectExtractor imageObjectExtractor = await
ImageObjectExtractor.CreateWithSoftwareBitmapAsync(softwareBitmap);

ImageObjectExtractorHint hint = new ImageObjectExtractorHint{
    includeRects: null,
    includePoints:
        new List<PointInt32> { new PointInt32(306, 212),
                                new PointInt32(216, 336)},
    excludePoints: null};
SoftwareBitmap finalImage = imageObjectExtractor.GetSoftwareBitmapObjectMask(hint);
```

C++/WinRT

```
#include <winrt/Microsoft.Graphics.Imaging.h>
#include <winrt/Microsoft.Windows.AI.Imaging.h>
#include <winrt/Windows.Graphics.Imaging.h>
#include <winrt/Windows.Foundation.h>
using namespace winrt::Microsoft::Graphics::Imaging;
using namespace winrt::Microsoft::Windows::AI::Imaging;
using namespace winrt::Windows::Graphics::Imaging;
using namespace winrt::Windows::Foundation;

if (ImageObjectExtractor::GetReadyState() == AIFeatureReadyState::NotReady)
{
    auto loadResult = ImageObjectExtractor::EnsureReadyAsync().get();

    if (loadResult.Status() != AIFeatureReadyResultState::Success)
    {
        throw winrt::hresult_error(loadResult.ExtendedError());
    }
}

ImageObjectExtractor imageObjectExtractor =
ImageObjectExtractor::CreateWithSoftwareBitmapAsync(softwareBitmap).get();

ImageObjectExtractorHint hint(
    {},
    {
        Windows::Graphics::PointInt32{306, 212},
        Windows::Graphics::PointInt32{216, 336}
    },
    {}
);

Windows::Graphics::Imaging::SoftwareBitmap finalImage =
imageObjectExtractor.GetSoftwareBitmapObjectMask(hint);
```

Specify hints with included and excluded points

This code snippet demonstrates how to use both included and excluded points as hints.

C#

```
ImageObjectExtractorHint hint(
    includeRects: null,
    includePoints:
        new List<PointInt32> { new PointInt32(150, 90),
                                new PointInt32(216, 336),
                                new PointInt32(550, 330)},
    excludePoints:
        new List<PointInt32> { new PointInt32(306, 212)});
```

C++/WinRT

```
ImageObjectExtractorHint hint(
    {},
    {
        PointInt32{150, 90},
        PointInt32{216, 336},
        PointInt32{550, 330}
    },
    {
        PointInt32{306, 212}
    }
);
```

Specify hints with rectangle

This code snippet demonstrates how to use a rectangle (RectInt32 is `x, Y, Width, Height`) as a hint.

C#

```
ImageObjectExtractorHint hint(
    includeRects:
        new List<RectInt32> {new RectInt32(370, 278, 285, 126)},
    includePoints: null,
    excludePoints: null );
```

C++/WinRT

```
ImageObjectExtractorHint hint(
    {
        RectInt32{370, 278, 285, 126}
    },
    {},
    {}
);
```

See also

- [AI Imaging overview](#)
- [AI Dev Gallery ↗](#)
- [Windows AI API samples ↗](#)

Image Super Resolution

The Image Super Resolution APIs enable image sharpening and scaling.

Scaling is limited to a maximum factor of 8x as higher scale factors can introduce artifacts and compromise image accuracy. If either the final width or height is greater than 8x their original values, an exception will be thrown.

For API details, see [API ref for AI imaging features](#).

For content moderation details, see [Content safety with generative AI APIs](#).

Important

Package Manifest Requirements: To use Windows AI imaging APIs, your app must be packaged as an MSIX package with the `SystemAIModels` capability declared in your `Package.appxmanifest`. Additionally, ensure your manifest's `MaxVersionTested` attribute is set to a recent Windows version (e.g., `10.0.26226.0` or later) to properly support the Windows AI features. Using older values may cause "Not declared by app" errors when loading the model.

XML

```
<Dependencies>
    <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.17763.0"
MaxVersionTested="10.0.26226.0" />
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17763.0"
MaxVersionTested="10.0.26226.0" />
</Dependencies>
```

Image Super Resolution example

The following example shows how to change the scale (`targetWidth`, `targetHeight`) of an existing software bitmap image (`softwareBitmap`) and improve the image sharpness using an `ImageScaler` object (to improve sharpness without scaling the image, simply specify the existing image width and height).

1. Ensure the Image Super Resolution model is available by calling the `GetReadyState` method and then waiting for the `EnsureReadyAsync` method to return successfully.
2. Once the Image Super Resolution model is available, create an `ImageScaler` object to reference it.

3. Get a sharpened and scaled version of the existing image by passing the existing image and the desired width and height to the model using the [ScaleSoftwareBitmap](#) method.

C#

```
using Microsoft.Graphics.Imaging;
using Microsoft.Windows.Management.Deployment;
using Microsoft.Windows.AI;
using Windows.Graphics.Imaging;

if (ImageScaler.GetReadyState() == AIFeatureReadyState.NotReady)
{
    var result = await ImageScaler.EnsureReadyAsync();
    if (result.Status != AIFeatureReadyResultState.Success)
    {
        throw result.ExtendedError;
    }
}
ImageScaler imageScaler = await ImageScaler.CreateAsync();
SoftwareBitmap finalImage = imageScaler.ScaleSoftwareBitmap(softwareBitmap,
targetWidth, targetHeight);
```

C++/WinRT

```
#include <winrt/Microsoft.Graphics.Imaging.h>
#include <winrt/Microsoft.Windows.AI.h>
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Graphics.Imaging.h>

using namespace winrt::Microsoft::Graphics::Imaging;
using namespace winrt::Microsoft::Windows::AI;
using namespace winrt::Windows::Foundation;
using namespace winrt::Windows::Graphics::Imaging;

if (ImageScaler::GetReadyState() == AIFeatureReadyState::NotReady)
{
    auto loadResult = ImageScaler::EnsureReadyAsync().get();

    if (loadResult.Status() != AIFeatureReadyResultState::Success)
    {
        throw winrt::hresult_error(loadResult.ExtendedError());
    }
}
int targetWidth = 100;
int targetHeight = 100;
ImageScaler imageScaler = ImageScaler::CreateAsync().get();
Windows::Graphics::Imaging::SoftwareBitmap finalImage =
    imageScaler.ScaleSoftwareBitmap(softwareBitmap, targetWidth, targetHeight);
```

See also

- [AI Imaging overview](#)
 - [AI Dev Gallery ↗](#)
 - [Windows AI API samples ↗](#)
-

Last updated on 01/24/2026

Image scaler walkthrough

This short tutorial will walk you through a sample that uses Image Scaler in a .NET MAUI app. To start, ensure you've completed the steps in the [Getting Started page](#) for .NET MAUI.

Introduction

This sample demonstrates use of some Windows AI APIs, including LanguageModel for text generation and ImageScaler for image super resolution to scale and sharpen images. Click one of the "Scale" buttons to scale the image (or reshew the original, unscaled image), or enter a text prompt and click the "Generate" button to generate a text response.

The changes from the ".NET MAUI App" template are split across four files:

1. MauiWindowsAISample.csproj: Adds the required Windows App SDK package reference for the Windows AI APIs. This reference needs to be conditioned only when building for Windows (see Additional Notes below for details). This file also sets the necessary TargetFramework for Windows.
2. Platforms/Windows/MainPage.cs: Implements partial methods from the shared MainPage class to show and handle the text generation and image scaling functionality.
3. MainPage.xaml: Defines controls to show text generation and image scaling.
4. MainPage.xaml.cs: Defines partial methods which the Windows-specific MainPage.cs implements.

In the second file listed above, you'll find the following function, which demonstrates some basic functionality for the ImageScaler method:

```
C#  
  
private async void DoScaleImage(double scale)  
{  
    // Load the original image  
    var resourceManager = new  
    Microsoft.Windows.ApplicationModel.Resources.ResourceManager();  
    var resource = resourceManager.MainResourceMap.GetValue("ms-  
resource:///Files/enhance.png");  
    if (resource.Kind ==  
    Microsoft.Windows.ApplicationModel.Resources.ResourceCandidateKind.FilePath)  
    {  
        // Load as a SoftwareBitmap  
        var file = await  
        Windows.Storage.StorageFile.GetFileFromPathAsync(resource.ValueAsString);  
        var fileStream = await file.OpenStreamForReadAsync();  
  
        var decoder = await  
        BitmapDecoder.CreateAsync(fileStream.AsRandomAccessStream());
```

```

var softwareBitmap = await decoder.GetSoftwareBitmapAsync();
int origWidth = softwareBitmap.PixelWidth;
int origHeight = softwareBitmap.PixelHeight;

SoftwareBitmap finalImage;
if (scale == 0.0)
{
    // just show the original image
    finalImage = softwareBitmap;
}
else
{
    // Scale the image to be the exact pixel size of the element displaying
it
    if (ImageScaler.GetReadyState() == AIFeatureReadyState.NotReady)
    {
        var op = await ImageScaler.EnsureReadyAsync();
        if (op.Status != AIFeatureReadyResultState.Success)
        {
            throw new Exception(op.ExtendedError.Message);
        }
    }
}

ImageScaler imageScaler = await ImageScaler.CreateAsync();

double imageSize = scale;
if (imageSize > imageScaler.MaxSupportedScaleFactor)
{
    imageSize = imageScaler.MaxSupportedScaleFactor;
}
System.Diagnostics.Debug.WriteLine($"Scaling to {imageSize}x...");

int newHeight = (int)(origHeight * imageSize);
int newWidth = (int)(origWidth * imageSize);
finalImage = imageScaler.ScaleSoftwareBitmap(softwareBitmap, newWidth,
newHeight);
}

// Display the scaled image. The if/else here shows two different approaches
to do this.
var mauiContext = scaledImage.Handler?.MauiContext;
if (mauiContext != null)
{
    // set the SoftwareBitmap as the source of the Image control
    var imageToShow = finalImage;
    if (imageToShow.BitmapPixelFormat != BitmapPixelFormat.Bgra8 ||
        imageToShow.BitmapAlphaMode == BitmapAlphaMode.Straight)
    {
        // SoftwareBitmapSource only supports Bgra8 and doesn't support
        Straight alpha mode, so convert
        imageToShow = SoftwareBitmap.Convert(imageToShow,
BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied);
    }
    var softwareBitmapSource = new SoftwareBitmapSource();
    _ = softwareBitmapSource.SetBitmapAsync(imageToShow);
}

```

```
        var nativeScaledImageView =
(Microsoft.UI.Xaml.Controls.Image)scaledImage.ToPlatform(mauiContext);
        nativeScaledImageView.Source = softwareBitmapSource;
    }
    else
    {
        // An alternative approach is to encode the image so a stream can be
        handed
        // to the Maui ImageSource.

        // Note: There's no "using(...)" here, since this stream needs to be
        kept alive for the image to be displayed
        var scaledStream = new InMemoryRandomAccessStream();
        {
            BitmapEncoder encoder = await
BitmapEncoder.CreateAsync(BitmapEncoder.PngEncoderId, scaledStream);
            encoder.SetSoftwareBitmap(finalImage);
            await encoder.FlushAsync();
            scaledImage.Source = ImageSource.FromStream(() =>
scaledStreamAsStream());
        }
    }
}
```

Build and run the sample

1. Clone the [WindowsAppSDK-Samples](#) repo.
2. Switch to the "release/experimental" branch.
3. Navigate to the [Samples/WindowsAIFoundry/cs-maui](#) folder.
4. Open MauiWindowsAISample.sln in Visual Studio 2022.
5. Ensure the debug toolbar has "Windows Machine" set as the target device.
6. Press F5 or select "Start Debugging" from the Debug menu to run the sample (the sample can also be run without debugging by selecting "Start Without Debugging" from the Debug menu or Ctrl+F5).

See also

- [AI Dev Gallery](#)
- [Windows AI API samples](#)

App Content Search Overview

The App Content Search feature enabled by the Windows AI APIs lets app developers integrate intelligent search capabilities into their Windows apps using the [AppContentIndexer API](#). By indexing in-app content and making it searchable through semantic queries, users can retrieve results based not only on exact keywords but also on semantic meaning. You can use this semantic index to enhance your own AI assistants with domain-specific knowledge, creating more personalized, context-specific experiences.

Use this API to:

- Build in-app search experiences that use both semantic and lexical search. Users can search by meaning, in addition to exact keyword matches, making it easier to find relevant information.
- Support Retrieval-Augmented Generation (RAG) by enabling local knowledge retrieval. When paired with a Large Language Model (LLM), this allows you to retrieve the most relevant content from your app's knowledge base and generate more accurate, context-aware responses.

The ApplicationContentIndexer API is currently only available in Windows App SDK release 2.0 Experimental 2.

[Open AI Dev Gallery to try App Content Search](#)

The AI Dev Gallery app offers an interactive sample of the AppContentIndexer API enabling you to experiment with the App Content Search feature. [Learn more about the AI Dev Gallery](#), including how to install from the Microsoft Store or from the source code on GitHub.

What is the AppContentIndexer API?

The [AppContentIndexer API](#) allows apps to make their text and image content searchable using both keyword-based (lexical) and meaning-based (semantic) search—without requiring developers to understand the underlying complexity.

Behind the scenes, it uses advanced techniques like embedding vectors, vector databases, and traditional text indexing, but these details are fully abstracted. Developers interact with a simple, high-level API. When content is indexed, the system stores embedding vectors (which capture semantic meaning) along with content identifiers. Search requests then return identifiers based on either keyword matches or semantic similarity. For example, searching for "kitten" might return related text about cats or images of kittens. Semantic searches work best

with descriptive phrases, so a query like "cats sitting on windowsills" is more likely to produce highly relevant results.

The index is persisted to disk, so re-indexing isn't needed on each app launch.

Semantic and lexical search

Internally, `ApplicationContentIndexer` uses a combination of traditional text indexing and modern vector-based search powered by embeddings. These details are abstracted away – developers do not need to manage embedding models, vector storage, or retrieval infrastructure directly.

You can query the index using a plain string. The query may return:

- **Lexical matches** – exact text matches (including text found within images).
- **Semantic matches** – content that is similar in meaning, even if the words are not identical.

For example, a query for "kitten" might return a reference to:

- Text entries about cats, even if the word "kitten" isn't explicitly mentioned.
- Images that visually contain kittens.
- Textual content in images that contain 'cat' or words with enough semantic relevance.

Supported content types

`ApplicationContentIndexer` supports adding the following types of content:

- **Text** – plain or structured text content.
- **Images** – including screenshots, photos, or image files that contain text or recognizable visual elements.

App-defined content identifiers

`AppContentIndexer` supports app-managed content by allowing apps to index items using app-defined content identifiers. Queries return these identifiers, which the app uses to retrieve the actual content from its own data store.

Text queries return `AppManagedTextQueryMatch` objects, and image queries return `AppManagedImageQueryMatch` objects—both include only the `ContentId`, not the content itself.

For guidance on how to integrate this feature into your app and use the ApplicationContentIndexer API, see: [Quickstart: App Content Search](#)

Privacy and security

Semantic and lexical indexes are generated on behalf of your app and stored in the app's local app data folder. As part of the private preview release, this feature is intended for indexing non-sensitive application content. For best security practices, do not use this feature to index user data that may contain personal, confidential, or sensitive information.

Responsible AI considerations

The semantic indexing and search capabilities in this preview do not apply any form of content moderation, nor do they attempt to detect or mitigate semantic bias introduced by the underlying models. Developers are responsible for evaluating and managing the potential risks when implementing AI-powered features.

We recommend reviewing the [Responsible Generative AI Development on Windows guidelines](#) for best practices when building AI experiences in your app.

Last updated on 11/18/2025

Get Started with App Content Search

Use App Content Search to create a semantic index of your in-app content. This allows users to find information based on meaning, rather than just keywords. The index can also be used to enhance AI assistants with domain-specific knowledge for more personalized and contextual results.

Specifically, you will learn how to use the [AppContentIndexer](#) API to:

- ✓ Create or open an index of the content in your app
- ✓ Add text strings to the index and then run a query
- ✓ Manage long text string complexity
- ✓ Index image data and then search for relevant images
- ✓ Enable RAG (Retrieval-Augmented Generation) scenarios
- ✓ Use AppContentIndexer on a background thread
- ✓ Close AppContentIndexer when no longer in use to release resources

Prerequisites

To learn about the Windows AI API hardware requirements and how to configure your device to successfully build apps using the Windows AI APIs, see [Get started building an app with Windows AI APIs](#).

Package Identity Requirement

Apps using [AppContentIndexer](#) must have package identity, which is only available to packaged apps (including those with external locations). To enable semantic indexing and [Text Recognition \(OCR\)](#), the app must also declare the `systemaimodels` capability.

Create or open an index of the content in your app

To create a semantic index of the content in your app, you must first establish a searchable structure that your app can use to store and retrieve content efficiently. This index acts as a local semantic and lexical search engine for your app's content.

To use the [AppContentIndexer](#) API, first call `GetOrCreateIndex` with a specified index name. If an index with that name already exists for the current app identity and user, it is opened; otherwise, a new one is created.

C#

```

public void SimpleGetOrCreateIndexSample()
{
    GetOrCreateIndexResult result = AppContentIndexer.GetOrCreateIndex("myindex");
    if (!result.Succeeded)
    {
        throw new InvalidOperationException($"Failed to open index. Status = '{result.Status}', Error = '{result.ExtendedError}'");
    }
    // If result.Succeeded is true, result.Status will either be CreatedNew or OpenedExisting
    if (result.Status == GetOrCreateIndexStatus.CreatedNew)
    {
        Console.WriteLine("Created a new index");
    }
    else if(result.Status == GetOrCreateIndexStatus.OpenedExisting)
    {
        Console.WriteLine("Opened an existing index");
    }
    using AppContentIndexer indexer = result.Indexer;
    // Use indexer...
}

```

This sample shows error handling the failure case for opening an index. For simplicity, other samples in this document may not show error handling.

Add text strings to the index and then run a query

This sample demonstrates how to add some text strings to the index created for your app and then run a query against that index to retrieve relevant information.

C#

```

// This is some text data that we want to add to the index:
Dictionary<string, string> simpleTextData = new Dictionary<string, string>
{
    {"item1", "Here is some information about Cats: Cats are cute and fluffy. Young cats are very playful." },
    {"item2", "Dogs are loyal and affectionate animals known for their companionship, intelligence, and diverse breeds." },
    {"item3", "Fish are aquatic creatures that breathe through gills and come in a vast variety of shapes, sizes, and colors." },
    {"item4", "Broccoli is a nutritious green vegetable rich in vitamins, fiber, and antioxidants." },
    {"item5", "Computers are powerful electronic devices that process information, perform calculations, and enable communication worldwide." },
    {"item6", "Music is a universal language that expresses emotions, tells stories, and connects people through rhythm and melody." },
};

public void SimpleTextIndexingSample()
{

```

```

AppContentIndexer indexer = GetIndexerForApp();
// Add some text data to the index:
foreach (var item in simpleTextData)
{
    IndexableAppContent.textContent =
AppManagedIndexableAppContent.CreateFromString(item.Key, item.Value);
    indexer.AddOrUpdate(textContent);
}
}

public void SimpleTextQueryingSample()
{
    AppContentIndexer indexer = GetIndexerForApp();
    // We search the index using a semantic query:
    AppIndexTextQuery queryCursor = indexer.CreateTextQuery("Facts about
kittens.");
    IReadOnlyList<TextQueryMatch> textMatches = queryCursor.GetNextMatches(5);
    // Nothing in the index exactly matches what we queried but item1 is similar
to the query so we expect
    // that to be the first match.
    foreach (var match in textMatches)
    {
        Console.WriteLine(match.ContentId);
        if (match.ContentKind == QueryMatchContentKind.AppManagedText)
        {
            AppManagedTextQueryMatch textResult =
(AppManagedTextQueryMatch)match;
            // Only part of the original string may match the query. So we can
use TextOffset and TextLength to extract the match.
            // In this example, we might imagine that the substring "Cats are
cute and fluffy" from "item1" is the top match for the query.
            string matchingData = simpleTextData[match.ContentId];
            string matchingString =
matchingData.Substring(textResult.TextOffset, textResult.TextLength);
            Console.WriteLine(matchingString);
        }
    }
}

```

`QueryMatch` includes only `ContentId` and `TextOffset/TextLength`, not the matching text itself. It is your responsibility as the app developer to reference the original text. Query results are sorted by relevancy, with the top result being most relevant. Indexing occurs asynchronously, so queries may run on partial data. You can check the indexing status as outlined below.

Manage long text string complexity

The sample demonstrates that it is not necessary for the app developer to divide the text content into smaller sections for model processing. The `AppContentIndexer` manages this aspect of complexity.

C#

```
Dictionary<string, string> textFiles = new Dictionary<string, string>
{
    {"file1", "File1.txt"},  
    {"file2", "File2.txt"},  
    {"file3", "File3.txt"},  
};  
public void TextIndexingSample2()  
{  
    AppContentIndexer indexer = GetIndexerForApp();  
    var folderPath =  
Windows.ApplicationModel.Package.Current.InstalledLocation.Path;  
    // Add some text data to the index:  
    foreach (var item in textFiles)  
    {  
        string contentId = item.Key;  
        string filename = item.Value;  
        // Note that the text here can be arbitrarily large. The  
AppContentIndexer will take care of chunking the text  
        // in a way that works effectively with the underlying model. We do not  
require the app author to break the text  
        // down into small pieces.  
        string text = File.ReadAllText(Path.Combine(folderPath, filename));  
        IndexableAppContent textContent =  
AppManagedIndexableAppContent.CreateFromString(contentId, text);  
        indexer.AddOrUpdate(textContent);  
    }  
}  
  
public void TextIndexingSample2_RunQuery()  
{  
    AppContentIndexer indexer = GetIndexerForApp();  
    var folderPath =  
Windows.ApplicationModel.Package.Current.InstalledLocation.Path;  
    // Search the index  
    AppIndexTextQuery query = indexer.CreateTextQuery("Facts about kittens.");  
    IReadOnlyList<TextQueryMatch> textMatches = query.GetNextMatches(5);  
    if (textMatches != null)  
    {  
        foreach (var match in textMatches)  
        {  
            Console.WriteLine(match.ContentId);  
            if (match is AppManagedTextQueryMatch textResult)  
            {  
                // We load the content of the file that contains the match:  
                string matchingFilename = textFiles[match.ContentId];  
                string fileContent = File.ReadAllText(Path.Combine(folderPath,  
matchingFilename));  
  
                // Find the substring within the loaded text that contains the  
match:  
                string matchingString =  
fileContent.Substring(textResult.TextOffset, textResult.TextLength);  
                Console.WriteLine(matchingString);  
            }  
        }  
    }  
}
```

```
        }
    }
}
```

Text data is sourced from files, but only the content is indexed, not the files themselves.

AppContentIndexer has no knowledge of the original files and does not monitor updates. If the file content changes, the app must update the index manually.

Index image data and then search for relevant images

This sample demonstrates how to index image data as `SoftwareBitmaps` and then search for relevant images using text queries.

C#

```
// We load the image data from a set of known files and send that image data to
// the indexer.
// The image data does not need to come from files on disk, it can come from
// anywhere.
Dictionary<string, string> imageFilesToIndex = new Dictionary<string, string>
{
    {"item1", "Cat.jpg" },
    {"item2", "Dog.jpg" },
    {"item3", "Fish.jpg" },
    {"item4", "Broccoli.jpg" },
    {"item5", "Computer.jpg" },
    {"item6", "Music.jpg" },
};
public void SimpleImageIndexingSample()
{
    AppContentIndexer indexer = GetIndexerForApp();

    // Add some image data to the index.
    foreach (var item in imageFilesToIndex)
    {
        var file = item.Value;
        var softwareBitmap = Helpers.GetSoftwareBitmapFromFile(file);
        IndexableAppContent imageContent =
            AppManagedIndexableAppContent.CreateFromBitmap(item.Key, softwareBitmap);
        indexer.AddOrUpdate(imageContent);
    }
}
public void SimpleImageIndexingSample_RunQuery()
{
    AppContentIndexer indexer = GetIndexerForApp();
    // We query the index for some data to match our text query.
    AppIndexImageQuery query = indexer.CreateImageQuery("cute pictures of
kittens");
```

```

IReadOnlyList<ImageQueryMatch> imageMatches = query.GetNextMatches(5);
// One of the images that we indexed was a photo of a cat. We expect this to
be the first match to match the query.
foreach (var match in imageMatches)
{
    Console.WriteLine(match.ContentId);
    if (match.ContentKind == QueryMatchContentKind.AppManagedImage)
    {
        AppManagedImageQueryMatch imageResult =
        (AppManagedImageQueryMatch)match;
        var matchingFileName = imageFilesToIndex[match.ContentId];

        // It might be that the match is at a particular region in the
        // image. The result includes
        // the subregion of the image that includes the match.

        Console.WriteLine($"Matching file: '{matchingFileName}' at location
{imageResult.Subregion}");
    }
}
}

```

Enable RAG (Retrieval-Augmented Generation) scenarios

RAG (Retrieval-Augmented Generation) involves augmenting user queries to language models with additional relevant data that can be used for generating responses. The user's query serves as input for semantic search, which identifies pertinent information in an index. The resulting data from the semantic search is then incorporated into the prompt given to the language model so that more accurate and context-aware responses can be generated.

This sample demonstrates how the **AppContentIndexer API** can be used to with an LLM to add contextual data to your app user's search query. The sample is generic, no LLM is specified and the example only queries the local data stored in the index created (no external calls to the internet). In this sample, `Helpers.GetUserPrompt()` and `Helpers.GetResponseFromChatAgent()` are not real functions and are just used to provide an example.

To enable RAG scenarios with the **AppContentIndexer API**, you can follow this example:

C#

```

public void SimpleRAGScenario()
{
    AppContentIndexer indexer = GetIndexerForApp();
    // These are some text files that had previously been added to the index.
    // The key is the contentId of the item.
    Dictionary<string, string> data = new Dictionary<string, string>
    {

```

```

        {"file1", "File1.txt" },
        {"file2", "File2.txt" },
        {"file3", "File3.txt" },
    };
    string userPrompt = Helpers.GetUserPrompt();
    // We execute a query against the index using the user's prompt string as
    // the query text.
    AppIndexTextQuery query = indexer.CreateTextQuery(userPrompt);
    IReadOnlyList<TextQueryMatch> textMatches = query.GetNextMatches(5);
    StringBuilder promptStringBuilder = new StringBuilder();
    promptStringBuilder.AppendLine("Please refer to the following pieces of
information when responding to the user's prompt:");
    // For each of the matches found, we include the relevant snippets of the
    // text files in the augmented query that we send to the language model
    foreach (var match in textMatches)
    {
        if (match is AppManagedTextQueryMatch textResult)
        {
            // We load the content of the file that contains the match:
            string matchingFilename = data[match.ContentId];
            string fileContent = File.ReadAllText(matchingFilename);
            // Find the substring within the loaded text that contains the
            // match:
            string matchingString = fileContent.Substring(textResult.TextOffset,
textResult.TextLength);
            promptStringBuilder.AppendLine(matchingString);
            promptStringBuilder.AppendLine();
        }
    }
    promptStringBuilder.AppendLine("Please provide a response to the following
user prompt:");
    promptStringBuilder.AppendLine(userPrompt);
    var response =
    Helpers.GetResponseFromChatAgent(promptStringBuilder.ToString());
    Console.WriteLine(response);
}

```

Use AppContentIndexer on a background thread

An **AppContentIndexer** instance is not associated with a particular thread; it is an agile object that can operate across threads. Certain methods of **AppContentIndexer** and its related types may require considerable processing time. Therefore, it is advisable to avoid invoking **AppContentIndexer** APIs directly from the application's UI thread and instead use a background thread.

Close AppContentIndexer when no longer in use to release resources

AppContentIndexer implements the `IClosable` interface to determine its lifetime. The application should close the indexer when it is no longer in use. This allows **AppContentIndexer** to release its underlying resources.

C#

```
public void IndexerDisposeSample()
{
    var indexer = AppContentIndexer.GetOrCreateIndex("myindex").Indexer;
    // use indexer
    indexer.Dispose();
    // after this point, it would be an error to try to use indexer since it is
    now Closed.
}
```

In C# code, the `IClosable` interface is projected as `IDisposable`. C# code can use the `using` pattern for **AppContentIndexer** instances.

C#

```
public void IndexerUsingSample()
{
    using var indexer = AppContentIndexer.GetOrCreateIndex("myindex").Indexer;
    // use indexer
    //indexer.Dispose() is automatically called
}
```

If you open the same index multiple times in your app, you must call `close` on each instance.

Opening and closing an index is an expensive operation, so you should minimize such operations in your application. For example, an application could store a single instance of the **AppContentIndexer** for the application and use that instance throughout the lifetime of the application instead of constantly opening and closing the index for each action that needs to be performed.

Last updated on 11/17/2025

Speech Recognition on Windows

Developers looking to use speech recognition in their Windows apps have two options:

- [Whisper via Foundry Local](#) (Windows 10+, new model, *performance varies, not available on all devices*)
 - [Speech Recognition via Windows SDK](#) (Windows 10+, older model, but available on all devices)
-

Last updated on 01/24/2026

Get Started with AI Text Recognition (OCR)

Text recognition, also known as optical character recognition (OCR), is supported by a set of Windows AI APIs that can detect and extract text within images and convert it into machine readable character streams.

These APIs can identify characters, words, lines, polygonal text boundaries, and provide confidence levels for each match. They are also exclusively supported by hardware acceleration in devices with a neural processing unit (NPU), making them faster and more accurate than the legacy Windows.Media.Ocr.OcrEngine APIs in the [Windows platform SDK](#).

For API details, see [API ref for Text Recognition \(OCR\)](#).

What can I do with AI Text Recognition?

Use AI Text Recognition features to identify and recognize text in an image. You can also get the text boundaries and confidence scores for the recognized text.

Note

Characters that are illegible or small in size can generate inaccurate results.

Create an ImageBuffer from a file

In this WinUI example we call a `LoadImageBufferFromFileAsync` function to get an `ImageBuffer` from an image file.

In the `LoadImageBufferFromFileAsync` function, we complete the following steps:

1. Create a `StorageFile` object from the specified file path.
2. Open a stream on the `StorageFile` using `OpenAsync`.
3. Create a `BitmapDecoder` for the stream.
4. Call `GetSoftwareBitmapAsync` on the bitmap decoder to get a `SoftwareBitmap` object.
5. Return an image buffer from `CreateBufferAttachedToBitmap`.

C#

```
using Microsoft.Windows.AI.Imaging;
using Microsoft.Graphics.Imaging;
using Windows.Graphics.Imaging;
using Windows.Storage;
using Windows.Storage.Streams;
```

```

public async Task<ImageBuffer> LoadImageBufferFromFileAsync(string filePath)
{
    StorageFile file = await StorageFile.GetFileFromPathAsync(filePath);
    IRandomAccessStream stream = await file.OpenAsync(FileAccessMode.Read);
    BitmapDecoder decoder = await BitmapDecoder.CreateAsync(stream);
    SoftwareBitmap bitmap = await decoder.GetSoftwareBitmapAsync();

    if (bitmap == null)
    {
        return null;
    }

    return ImageBuffer.CreateBufferAttachedToBitmap(bitmap);
}

```

C++/WinRT

```

#include <iostream>
#include <sstream>
#include <winrt/Microsoft.Windows.AI.Imaging.h>
#include <winrt/Windows.Graphics.Imaging.h>
#include <winrt/Microsoft.Graphics.Imaging.h>
#include <winrt/Microsoft.UI.Xaml.Controls.h>
#include <winrt/Microsoft.UI.Xaml.Media.h>
#include <winrt/Microsoft.UI.Xaml.Shapes.h>

using namespace winrt;
using namespace Microsoft::UI::Xaml;
using namespace Microsoft::Windows::AI;
using namespace Microsoft::Windows::AI::Imaging;
using namespace winrt::Microsoft::UI::Xaml::Controls;
using namespace winrt::Microsoft::UI::Xaml::Media;

winrt::Windows::Foundation::IAsyncOperation<winrt::hstring>
MainWindow::RecognizeTextFromSoftwareBitmap(
    Windows::Graphics::Imaging::SoftwareBitmap const& bitmap)
{
    winrt::Microsoft::Windows::AI::Imaging::TextRecognizer textRecognizer =
        EnsureModelIsReady().get();
    Microsoft::Graphics::Imaging::ImageBuffer imageBuffer =
        Microsoft::Graphics::Imaging::ImageBuffer::CreateForSoftwareBitmap(bitmap);
    RecognizedText recognizedText =
        textRecognizer.RecognizeTextFromImage(imageBuffer);
    std::wstringstream stringstream;
    for (const auto& line : recognizedText.Lines())
    {
        stringstream << line.Text().c_str() << std::endl;
    }
    co_return winrt::hstring{ stringstream.str()};
}

```

Recognize text in a bitmap image

The following example shows how to recognize some text in a `SoftwareBitmap` object as a single string value:

1. Create a `TextRecognizer` object through a call to the `EnsureModelIsReady` function, which also confirms there is a language model present on the system.
2. Using the bitmap obtained in the previous snippet, we call the `RecognizeTextFromSoftwareBitmap` function.
3. Call `CreateBufferAttachedToBitmap` on the image file to get an `ImageBuffer` object.
4. Call `RecognizeTextFromImage` to get the recognized text from the `ImageBuffer`.
5. Create a `wstringstream` object and load it with the recognized text.
6. Return the string.

! Note

The `EnsureModelIsReady` function is used to check the readiness state of the text recognition model (and install it if necessary).

C#

```
using Microsoft.Windows.AI.Imaging;
using Microsoft.Windows.AI;
using Microsoft.Graphics.Imaging;
using Windows.Graphics.Imaging;
using Windows.Storage;
using Windows.Storage.Streams;

public async Task<string> RecognizeTextFromSoftwareBitmap(SoftwareBitmap bitmap)
{
    TextRecognizer textRecognizer = await EnsureModelIsReady();
    ImageBuffer imageBuffer = ImageBuffer.CreateBufferAttachedToBitmap(bitmap);
    RecognizedText recognizedText =
    textRecognizer.RecognizeTextFromImage(imageBuffer);
    StringBuilder stringBuilder = new StringBuilder();

    foreach (var line in recognizedText.Lines)
    {
        stringBuilder.AppendLine(line.Text);
    }

    return stringBuilder.ToString();
}

public async Task<TextRecognizer> EnsureModelIsReady()
{
    if (TextRecognizer.GetReadyState() == AIFeatureReadyState.NotReady)
    {
```

```

        var loadResult = await TextRecognizer.EnsureReadyAsync();
        if (loadResult.Status != AIFeatureReadyResultState.Success)
        {
            throw new Exception(loadResult.ExtendedError().Message);
        }
    }

    return await TextRecognizer.CreateAsync();
}

```

C++/WinRT

```

winrt::Windows::Foundation::IAsyncOperation<winrt::Microsoft::Windows::AI::Imaging::TextRecognizer> MainWindow::EnsureModelIsReady()
{
    if (winrt::Microsoft::Windows::AI::Imaging::TextRecognizer::GetReadyState() == AIFeatureReadyState::NotReady)
    {
        auto loadResult = TextRecognizer::EnsureReadyAsync().get();

        if (loadResult.Status() != AIFeatureReadyResultState::Success)
        {
            throw winrt::hresult_error(loadResult.ExtendedError());
        }
    }

    return winrt::Microsoft::Windows::AI::Imaging::TextRecognizer::CreateAsync();
}

```

Get word bounds and confidence

Here we show how to visualize the **BoundingBox** of each word in a [SoftwareBitmap](#) object as a collection of color-coded [polygons](#) on a [Grid](#) element.

ⓘ Note

For this example we assume a **TextRecognizer** object has already been created and passed in to the function.

C#

```

using Microsoft.Windows.AI.Imaging;
using Microsoft.Graphics.Imaging;
using Windows.Graphics.Imaging;
using Windows.Storage;
using Windows.Storage.Streams;

public void VisualizeWordBoundariesOnGrid(

```

```

        SoftwareBitmap bitmap,
        Grid grid,
        TextRecognizer textRecognizer)
{
    ImageBuffer imageBuffer = ImageBuffer.CreateBufferAttachedToBitmap(bitmap);
    RecognizedText result = textRecognizer.RecognizeTextFromImage(imageBuffer);

    SolidColorBrush greenBrush = new SolidColorBrush(Microsoft.UI.Colors.Green);
    SolidColorBrush yellowBrush = new SolidColorBrush(Microsoft.UI.Colors.Yellow);
    SolidColorBrush redBrush = new SolidColorBrush(Microsoft.UI.Colors.Red);

    foreach (var line in result.Lines)
    {
        foreach (var word in line.Words)
        {
            PointCollection points = new PointCollection();
            var bounds = word.BoundingBox;
            points.Add(bounds.TopLeft);
            points.Add(bounds.TopRight);
            points.Add(bounds.BottomRight);
            points.Add(bounds.BottomLeft);

            Polygon polygon = new Polygon();
            polygon.Points = points;
            polygon.StrokeThickness = 2;

            if (word.Confidence < 0.33)
            {
                polygon.Stroke = redBrush;
            }
            else if (word.Confidence < 0.67)
            {
                polygon.Stroke = yellowBrush;
            }
            else
            {
                polygon.Stroke = greenBrush;
            }

            grid.Children.Add(polygon);
        }
    }
}

```

C++/WinRT

```

void MainWindow::VisualizeWordBoundariesOnGrid(
    Windows::Graphics::Imaging::SoftwareBitmap const& bitmap,
    Grid const& grid,
    TextRecognizer const& textRecognizer)
{
    Microsoft::Graphics::Imaging::ImageBuffer imageBuffer =
        Microsoft::Graphics::Imaging::ImageBuffer::CreateForSoftwareBitmap(bitmap);

```

```
RecognizedText result = textRecognizer.RecognizeTextFromImage(imageBuffer);

auto greenBrush = SolidColorBrush(winrt::Microsoft::UI::Colors::Green());
auto yellowBrush = SolidColorBrush(winrt::Microsoft::UI::Colors::Yellow());
auto redBrush = SolidColorBrush(winrt::Microsoft::UI::Colors::Red());
for (const auto& line : result.Lines())
{
    for (const auto& word : line.Words())
    {
        PointCollection points;
        const auto& bounds = word.BoundingBox();
        points.Append(bounds.TopLeft);
        points.Append(bounds.TopRight);
        points.Append(bounds.BottomRight);
        points.Append(bounds.BottomLeft);

        winrt::Microsoft::UI::Xaml::Shapes::Polygon polygon{};
        polygon.Points(points);
        polygon.StrokeThickness(2);
        if (word.MatchConfidence() < 0.33)
        {
            polygon.Stroke(redBrush);
        }
        else if (word.MatchConfidence() < 0.67)
        {
            polygon.Stroke(yellowBrush);
        }
        else
        {
            polygon.Stroke(greenBrush);
        }

        grid.Children().Append(polygon);
    }
}
}
```

Responsible AI

We've followed core principles and practices described in the [Microsoft Responsible AI Standards](#) to ensure these APIs are trustworthy, secure, and built responsibly. For more details on implementing AI features in your app, see [Responsible Generative AI Development on Windows](#).

See also

- [Access files and folders with Windows App SDK and WinRT APIs](#)
- [AI Dev Gallery](#)
- [Windows AI API samples](#)

Last updated on 01/21/2026

Text recognizer walkthrough

This short tutorial walks through the text recognition functionality included in the [Windows AI API samples](#) for WinForms. Specifically, it demonstrates how to use Windows AI APIs to perform text recognition on an image and summarize the recognized text.

Prerequisites

Complete the steps in the [Getting Started page](#) for WinForms.

Introduction

The **MainForm** class in `MainForm.cs` is the main user interface for the Windows AI API sample app that implements the following functionality:

- Select File: Lets the user select an image file from their file system and displays that image in a **PictureBox**.
- Process Image: Processes the selected image to extract text using Optical Character Recognition (OCR) and then summarizes the extracted text.

Key functions and event handlers

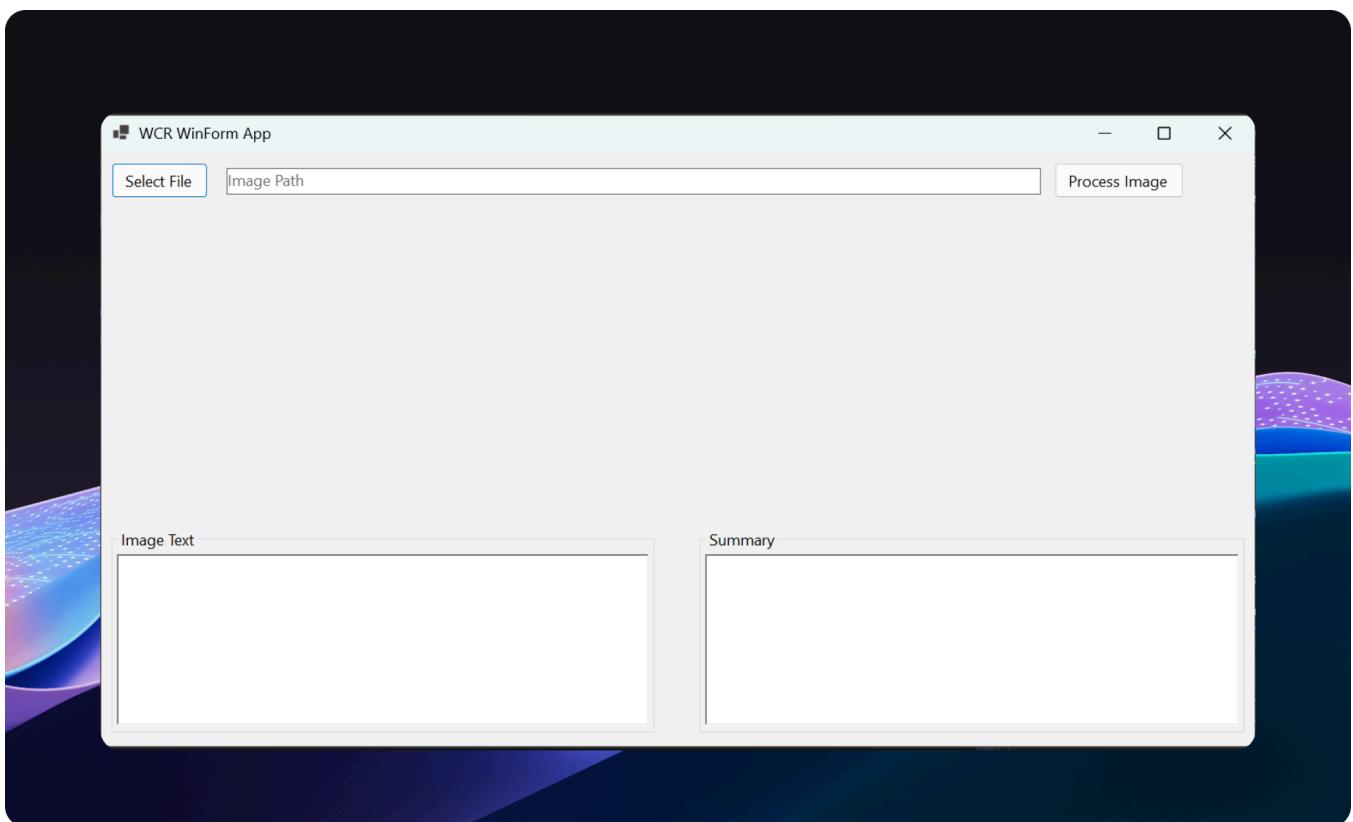
Some of the more significant functions and event handlers in the [Windows AI API samples](#) for WinForms include the following:

- `SelectFile_Click`: Opens a file dialog for the user to select an image file and displays the selected image.
- `ProcessButton_Click`: Handles the processing of the selected image, including loading AI models, performing text recognition, and summarizing the text.
- `LoadAIModels`: Loads the necessary AI models (`TextRecognizer` and `LanguageModel`) for text recognition and summarization.
- `PerformTextRecognition`: Uses the `TextRecognizer` to perform OCR on the selected image and extracts the text. This function is included in the following [Text recognition example](#).
- `SummarizeImageText`: Uses the `LanguageModel` to generate a summary of the extracted text given a prompt.

Text recognition example

The `PerformTextRecognition` function in this example

```
private void InitializeComponent()
{
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.SuspendLayout();
    //
    // textBox1
    //
    this.textBox1.AcceptsReturn = true;
    this.textBox1.AcceptsTab = true;
    this.textBox1.Dock = System.Windows.Forms.DockStyle.Fill;
    this.textBox1.Multiline = true;
    this.textBox1.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
    //
    // Form1
    //
    this.ClientSize = new System.Drawing.Size(284, 264);
    this.Controls.Add(this.textBox1);
    this.Text = "TextBox Example";
    this.ResumeLayout(false);
    this.PerformLayout();
}
}
```



C#

```
private async Task<string> PerformTextRecognition()
{
    using TextRecognizer textRecognizer = await TextRecognizer.CreateAsync();
    ImageBuffer? imageBuffer = await LoadImageBufferFromFileAsync(pathToImage);

    if (imageBuffer == null)
    {
```

```

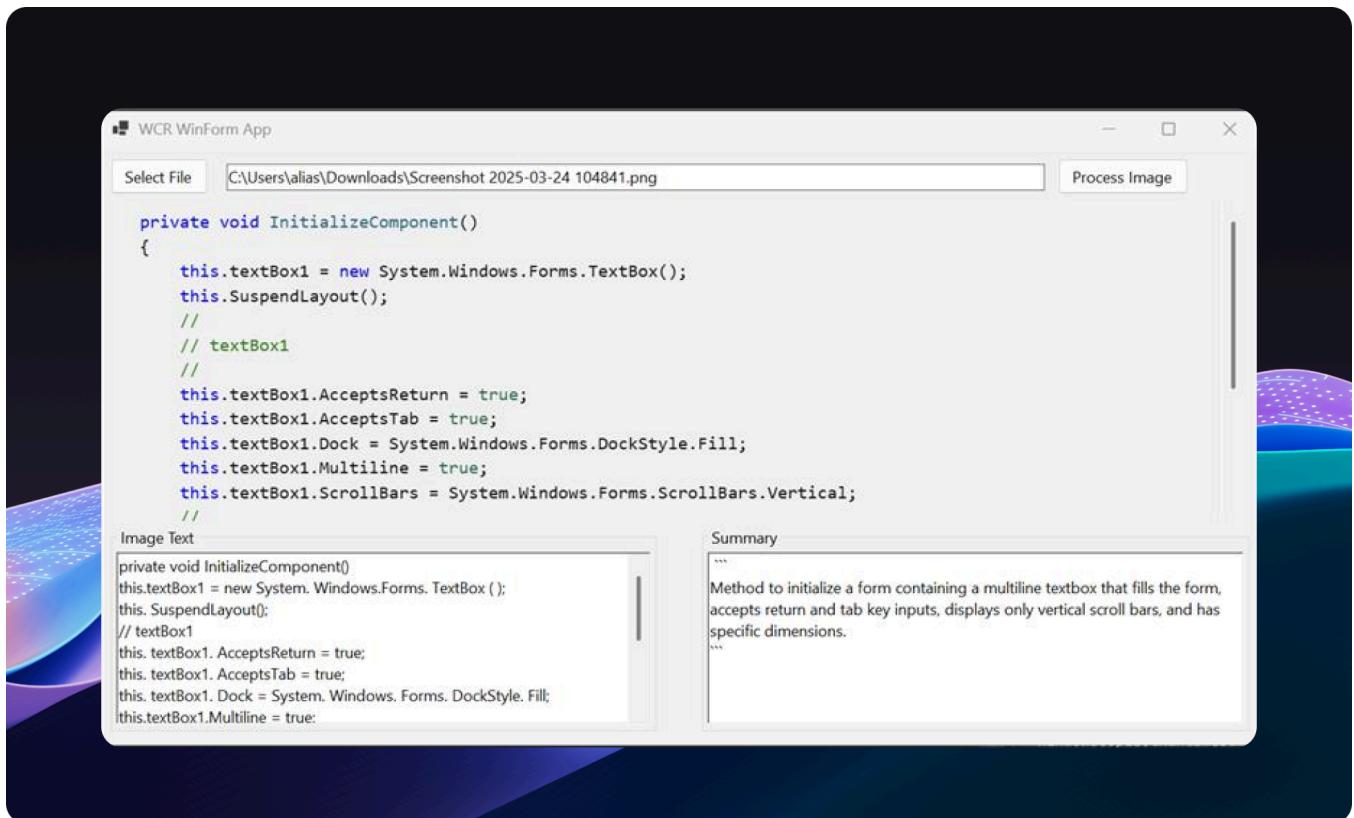
        throw new Exception("Failed to load image buffer.");
    }

    RecognizedText recognizedText =
        textRecognizer!.RecognizeTextFromImage(imageBuffer);

    var recognizedTextLines = recognizedText.Lines.Select(line => line.Text);
    string text = string.Join(Environment.NewLine, recognizedTextLines);

    richTextBoxForImageText.Text = text;
    return text;
}

```



Build and run the sample

1. Clone the [WindowsAppSDK-Samples](#) repo.
2. Switch to the "release/experimental" branch.
3. Navigate to the [Samples/WindowsAIFoundry/cs-winforms-pckg](#) folder.
4. Open WindowsAISample.sln in Visual Studio 2022.
5. Change the Solution Platform to match the architecture of your Copilot+ PC.
6. Right-click on the solution in Solution Explorer and select "Build" to build solution.
7. Once the build is successful, right-click on the project in Solution Explorer and select "Set as Startup Project".
8. Press F5 or select "Start Debugging" from the Debug menu to run the sample (the sample can also be run without debugging by selecting "Start Without Debugging" from the Debug menu or Ctrl+F5).

See also

- [AI Dev Gallery ↗](#)
 - [Windows AI API samples ↗](#)
-

Last updated on 11/18/2025

Get Started with AI Video Super Resolution (VSR)

Video Super Resolution (VSR) is an AI-based video up-sampling technology that intelligently upscales low-resolution video streams of people, restoring sharpness and detail that would otherwise be lost due to bandwidth limitations, poor network conditions, compression, or lower-quality source content.

Adding VSR capabilities to your app enables scenarios including the following:

- Improving video quality over poor network connections
- Bandwidth optimization to reduce CDN costs
- High-bandwidth scenarios like group video calls with multiple participants
- Improving social media video quality in editing, upload or viewing

The VSR feature currently requires a **Copilot+ PC** with an NPU. For more information, see [Develop AI applications for Copilot+ PCs](#).

These VSR APIs use Machine Learning (ML) models, were designed specifically for scenarios such as video calling and conferencing apps and social and short-form videos that feature human faces speaking

VSR currently supports the following resolution, format, and FPS ranges:

[] [Expand table](#)

Attribute	Supported Content
Input resolution	240p – 1440p
Output resolution	480p – 1440p
Frames-per-second (FPS) range	15 fps – 60 fps
Input pixel format	BGR (ImageBuffer API), NV12 (Direct3D API)
Output pixel format	BGR (ImageBuffer API), BGRA (Direct3D API)

Create a VideoScaler session

The following example shows how to create a VSR session. First, get an instance of [ExecutionProviderCatalog](#) and call [EnsureAndRegisterCertifiedAsync](#) to load the available models. Call [GetReadyState](#) on the [VideoScalar](#) class to determine if the video scaler is ready to process frames. If not, call [EnsureReadyAsync](#) to initialize the video scaler.

C#

```
private VideoScaler? _videoScaler;

protected override async Task LoadModelAsync(SampleNavigationParameters
sampleParams)
{
    try
    {

        var catalog = ExecutionProviderCatalog.GetDefault();
        await catalog.EnsureAndRegisterCertifiedAsync();

        var readyState = VideoScaler.GetReadyState();
        if (readyState == AIFeatureReadyState.NotReady)
        {
            var operation = await VideoScaler.EnsureReadyAsync();

            if (operation.Status != AIFeatureReadyResultState.Success)
            {
                ShowException(null, "Video Scaler is not available.");
            }
        }

        _videoScaler = await VideoScaler.CreateAsync();
    }
    catch (Exception ex)
    {
        ShowException(ex, "Failed to load model.");
    }

    sampleParams.NotifyCompletion();
}
```

Scale a VideoFrame

The following code example uses the `VideoScaler.ScaleFrame` method to upscale image data contained in a `VideoFrame` object. You can get `VideoFrame` from a camera by using the `MediaFrameReader` class. For more information, see [Process media frames with MediaFrameReader](#). You can also use the WinUI Community Toolkit `CameraPreview` control to get `VideoFrame` objects from the camera.

Next, a `Direct3DSurface` is obtained from the input video frame and another `Direct3DSurface` is created for the output of the upscaling. `VideoScaler.ScaleFrame` is called to upscale the frame. In this example, an `Image` control in the app's UI is updated with the upscaled frame.

C#

```
private async Task ProcessFrame(VideoFrame videoFrame)
{
    // Process the frame with super resolution model
    var processedBitmap = await Task.Run(async () =>
    {
        int width = 0;
        int height = 0;
        var inputD3dSurface = videoFrame.Direct3DSurface;
        if (inputD3dSurface != null)
        {
            Debug.Assert(inputD3dSurface.Description.Format ==
Windows.Graphics.DirectX.DirectXPixelFormat.NV12, "input in NV12 format");
            width = inputD3dSurface.Description.Width;
            height = inputD3dSurface.Description.Height;
        }
        else
        {
            var softwareBitmap = videoFrame.SoftwareBitmap;
            if (softwareBitmap == null)
            {
                return null;
            }

            Debug.Assert(softwareBitmap.BitmapPixelFormat == BitmapPixelFormat.Nv12,
"input in NV12 format");

            width = softwareBitmap.PixelWidth;
            height = softwareBitmap.PixelHeight;
        }

        try
        {
            if (inputD3dSurface == null)
            {
                // Create Direct3D11-backed VideoFrame for input
                using var inputVideoFrame =
VideoFrame.CreateAsDirect3D11SurfaceBacked(
                    Windows.Graphics.DirectX.DirectXPixelFormat.NV12,
                    width,
                    height);

                if (inputVideoFrame.Direct3DSurface == null)
                {
                    return null;
                }

                // Copy the software bitmap to the Direct3D-backed frame
                await videoFrame.CopyToAsync(inputVideoFrame);

                inputD3dSurface = inputVideoFrame.Direct3DSurface;
            }

            // Create or resize output surface (BGRA8 format for display)
            if (_outputD3dSurface == null || _outputWidth != width || _outputHeight

```

```

!= height)
{
    _outputD3dSurface?.Dispose();

    // DXGI_FORMAT_B8G8R8A8_UNORM = 87
    _outputD3dSurface = Direct3DExtensions.CreateDirect3DSurface(87,
width, height);
    _outputWidth = width;
    _outputHeight = height;
}

// Scale the frame using VideoScaler
var result = _videoScaler!.ScaleFrame(inputD3dSurface,
_outputD3dSurface, new VideoScalerOptions());

if (result.Status == ScaleFrameStatus.Success)
{
    var outputBitmap = await SoftwareBitmap.CreateCopyFromSurfaceAsync(
        _outputD3dSurface,
        BitmapAlphaMode.Premultiplied);

    return outputBitmap;
}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine($"ProcessFrame error: {ex.Message}");
}

return null;
});

if (processedBitmap == null)
{
    return;
}

DispatcherQueue.TryEnqueue(async () =>
{
    using (processedBitmap)
    {
        var source = new SoftwareBitmapSource();
        await source.SetBitmapAsync(processedBitmap);
        ProcessedVideoImage.Source = source;
    }
});
}

```

Scale a SoftwareBitmap using ImageBuffer

The following code example demonstrates the use of **VideoScalar** class to upscale a **SoftwareBitmap**. This example does not represent a typical usage of the VSR APIs. It is less

performant than using Direct3D. But you can use this example to experiment with the VSR APIs without setting up a camera or video streaming pipeline. Because the video scaler requires a **BGR8** when using an **ImageBuffer**, some helper methods are required to convert the pixel format of the supplied **SoftwareBitmap**.

The example code in this article is based on the VSR component of the [Windows AI API samples](#)

C#

```
public SoftwareBitmap ScaleVideoFrame(SoftwareBitmap inputFrame)
{
    ImageBuffer inputImageBuffer =
SoftwareBitmapExtensions.ConvertToBgr8ImageBuffer(inputFrame);
    var size = (uint)(inputFrame.PixelWidth * inputFrame.PixelHeight * 3);
    IBuffer outputBuffer = new global::Windows.Storage.Streams.Buffer(size);
    outputBuffer.Length = size;
    ImageBuffer outputImageBuffer = ImageBuffer.CreateForBuffer(
        outputBuffer,
        ImageBufferPixelFormat.Bgr8,
        inputFrame.PixelWidth,
        inputFrame.PixelHeight,
        inputFrame.PixelWidth * 3);
    var result = Session.ScaleFrame(inputImageBuffer, outputImageBuffer, new
VideoScalerOptions());
    if (result.Status != ScaleFrameStatus.Success)
    {
        throw new Exception($"Failed to scale video frame: {result.Status}");
    }

    return
SoftwareBitmapExtensions.ConvertBgr8ImageBufferToBgra8SoftwareBitmap(outputImageBuff
er);
}
```

Software bitmap extension methods

The following helper methods convert a **SoftwareBitmap** between **BGRA8** and **BGR8** formats to match the input and output requirements of the video scalar.

C#

```
public static ImageBuffer ConvertToBgr8ImageBuffer(SoftwareBitmap input)
{
    var bgraBitmap = input;
    if (input.BitmapPixelFormat != BitmapPixelFormat.Bgra8)
    {
        bgraBitmap = SoftwareBitmap.Convert(input, BitmapPixelFormat.Bgra8,
BitmapAlphaMode.Premultiplied);
    }
```

```

        int width = bgraBitmap.PixelWidth;
        int height = bgraBitmap.PixelHeight;

        byte[] bgraBuffer = new byte[width * height * 4];
        bgraBitmap.CopyToBuffer(bgraBuffer.AsBuffer());

        byte[] bgrBuffer = new byte[width * height * 3];
        for (int i = 0, j = 0; i < bgraBuffer.Length; i += 4, j += 3)
        {
            bgrBuffer[j] = bgraBuffer[i];
            bgrBuffer[j + 1] = bgraBuffer[i + 1];
            bgrBuffer[j + 2] = bgraBuffer[i + 2];
        }

        return ImageBuffer.CreateForBuffer(
            bgrBuffer.AsBuffer(),
            ImageBufferPixelFormat.Bgr8,
            width,
            height,
            width * 3);
    }

    public static SoftwareBitmap
ConvertBgr8ImageBufferToBgra8SoftwareBitmap(ImageBuffer bgrImageBuffer)
{
    if (bgrImageBuffer.PixelFormat != ImageBufferPixelFormat.Bgr8)
    {
        throw new ArgumentException("Input ImageBuffer must be in Bgr8 format");
    }

    int width = bgrImageBuffer.PixelWidth;
    int height = bgrImageBuffer.PixelHeight;

    // Get BGR data from ImageBuffer
    byte[] bgrBuffer = new byte[width * height * 3];
    bgrImageBuffer.CopyToArray(bgrBuffer);

    // Create BGRA buffer (4 bytes per pixel)
    byte[] bgraBuffer = new byte[width * height * 4];

    for (int i = 0, j = 0; i < bgrBuffer.Length; i += 3, j += 4)
    {
        bgraBuffer[j] = bgrBuffer[i];      // B
        bgraBuffer[j + 1] = bgrBuffer[i + 1]; // G
        bgraBuffer[j + 2] = bgrBuffer[i + 2]; // R
        bgraBuffer[j + 3] = 255;           // A (full opacity)
    }

    // Create SoftwareBitmap and copy data
    var softwareBitmap = new SoftwareBitmap(
        BitmapPixelFormat.Bgra8,
        width,
        height,
        BitmapAlphaMode.Premultiplied);
}

```

```
softwareBitmap.CopyFromBuffer(bgraBuffer.AsBuffer());  
  
    return softwareBitmap;  
}
```

Responsible AI

We've followed core principles and practices described in the [Microsoft Responsible AI Standards](#) to ensure these APIs are trustworthy, secure, and built responsibly. For more details on implementing AI features in your app, see [Responsible Generative AI Development on Windows](#).

These VSR APIs use Machine Learning (ML) models, were designed specifically for scenarios such as video calling and conferencing apps and social and short-form videos that feature human faces speaking. Therefore, we do not recommend using these APIs for images in the following scenarios:

- Where the images contain potentially sensitive content and inaccurate descriptions could be controversial, such as flags, maps, globes, cultural symbols, or religious symbols.
- When accurate descriptions are critical, such as for medical advice or diagnosis, legal content, or financial documents.

See also

- [AI Dev Gallery](#)
- [Windows AI API samples](#)

Last updated on 01/21/2026

What are Windows AI APIs?



A suite of hardware-abstraction AI APIs powered by [Windows Machine Learning \(ML\)](#) supports a variety of artificial intelligence (AI) features. The Windows AI APIs enable AI capabilities without the need to find, run, or optimize your own machine learning (ML) model. The models that power the Windows AI APIs on Copilot+ PCs run locally and can run continuously in the background.

See the [Windows AI APIs with WinUI sample app](#) for how to use Microsoft Foundry on Windows with WinUI.

Important

The following is a list of Windows AI features and the Windows App SDK release in which they are currently supported. See [Overview of available APIs](#) later in this topic for brief descriptions.

[Version 1.8.0 \(1.8.250907003\)](#) - [Phi Silica \(Limited Access Feature\)](#), [Conversation Summarization \(Text Intelligence\)](#), [Object Erase](#)

[Version 1.8 Preview \(1.8.0-preview\)](#) - [LoRA fine-tuning for Phi Silica](#), [Text Rewriter Tone \(Text Intelligence\)](#).

[Private preview](#) - Semantic Search

[Version 1.7.1 \(1.7.250401001\)](#) - All other APIs

Build your first AI-powered Windows app

Tip

To improve accessibility and readability, this page displays still images by default. In some cases, you can click an image to see an animated version.

To build your first Windows app with Visual Studio and some simple Windows AI APIs, just meet the prerequisites and use the provided example code in [Get started building an app with Windows AI APIs](#).

From there, you can jump into short tutorials that build an app leveraging specific Windows AI APIs such as the [Phi Silica walkthrough](#), [Imaging walkthrough](#) and [OCR walkthrough](#).

Try the APIs and models on your PC

AI Dev Gallery is a demo app—available from the Microsoft Store—that lets you quickly download, try out, and use Windows AI APIs and models.

[Install AI Dev Gallery from the Microsoft Store](#)

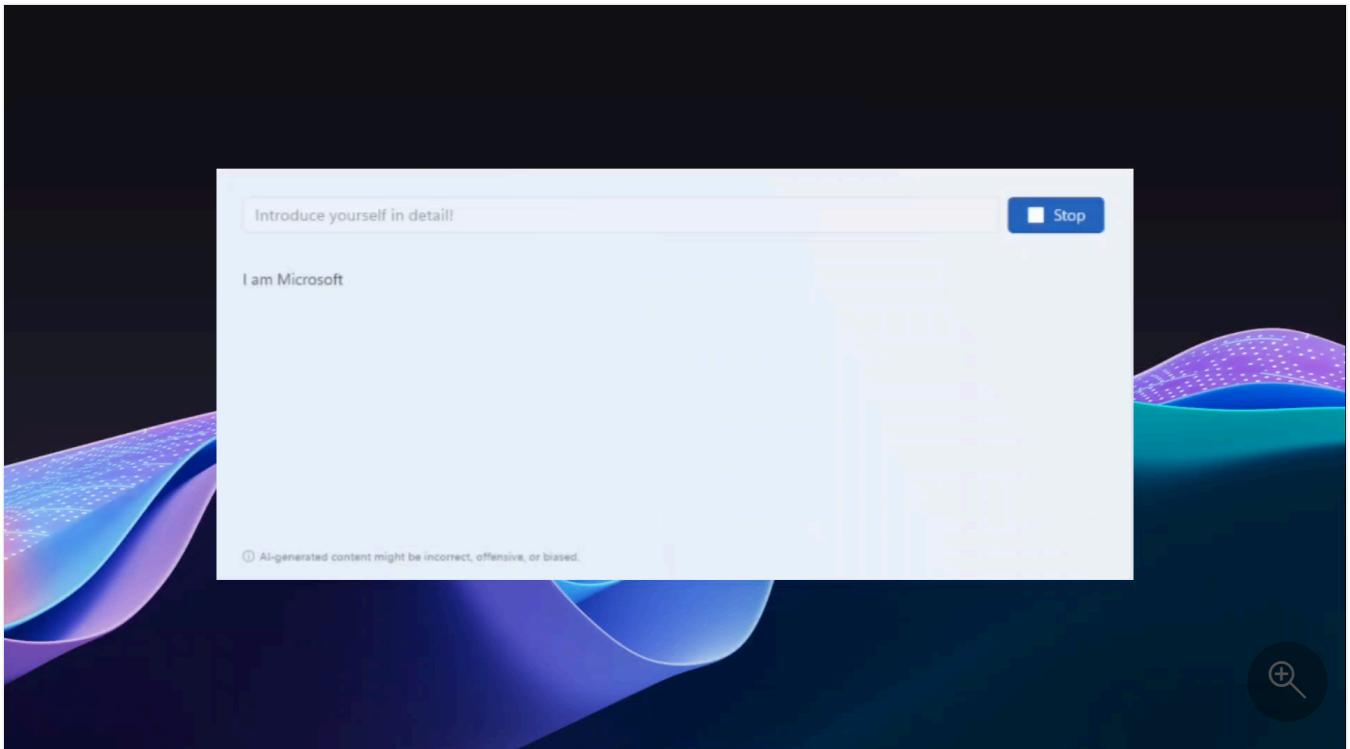
In AI Dev Gallery, select the **Windows AI APIs tab** menu item, then select the *Phi Silica* sample. If the model is already available on your device, then that sample will run straight away. Otherwise, select **Request model** to download the model. Once downloaded, that sample will be activated. Learn more about the AI Dev Gallery in [What is the AI Dev Gallery?](#).

Overview of available APIs

Here are a few ready-to-use AI features that you can tap into from your Windows app:

Phi Silica

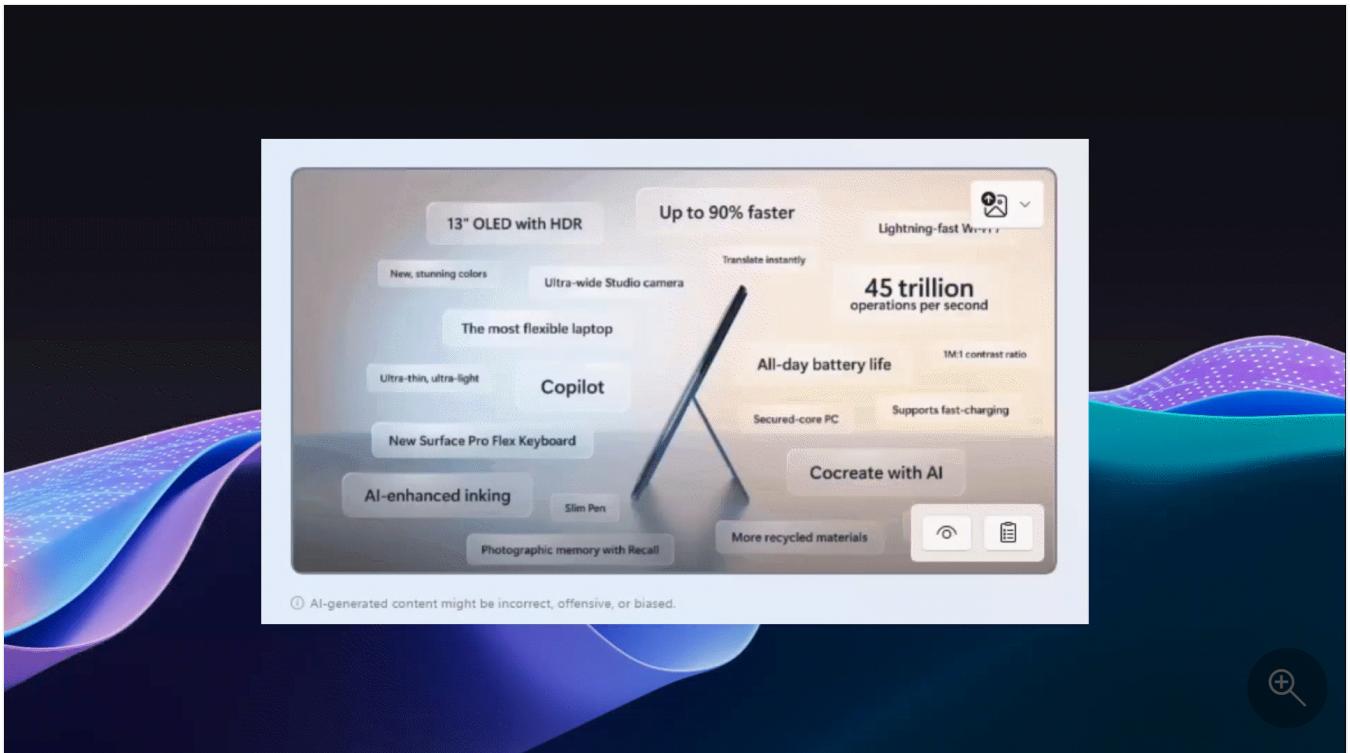
Similar to OpenAI's GPT Large Language Model (LLM), which powers ChatGPT, Phi Silica is a Small Language Model (SLM) developed by Microsoft Research to perform language-processing tasks on a local device (see [Get started with Phi Silica](#)). Phi Silica is specifically designed for Windows devices that have a Neural Processing Unit (NPU), allowing text generation and conversation features to run in a high performance, hardware-accelerated way directly on the device. *Phi Silica is not available in China.*



[Try it in AI Dev Gallery](#)

Text recognition

The text recognition APIs enable the recognition of text in an image, and the conversion on a local device of different types of documents (such as scanned paper documents, PDF files, and images captured by a digital camera) into editable and searchable data (see [Get started with AI text recognition](#)).



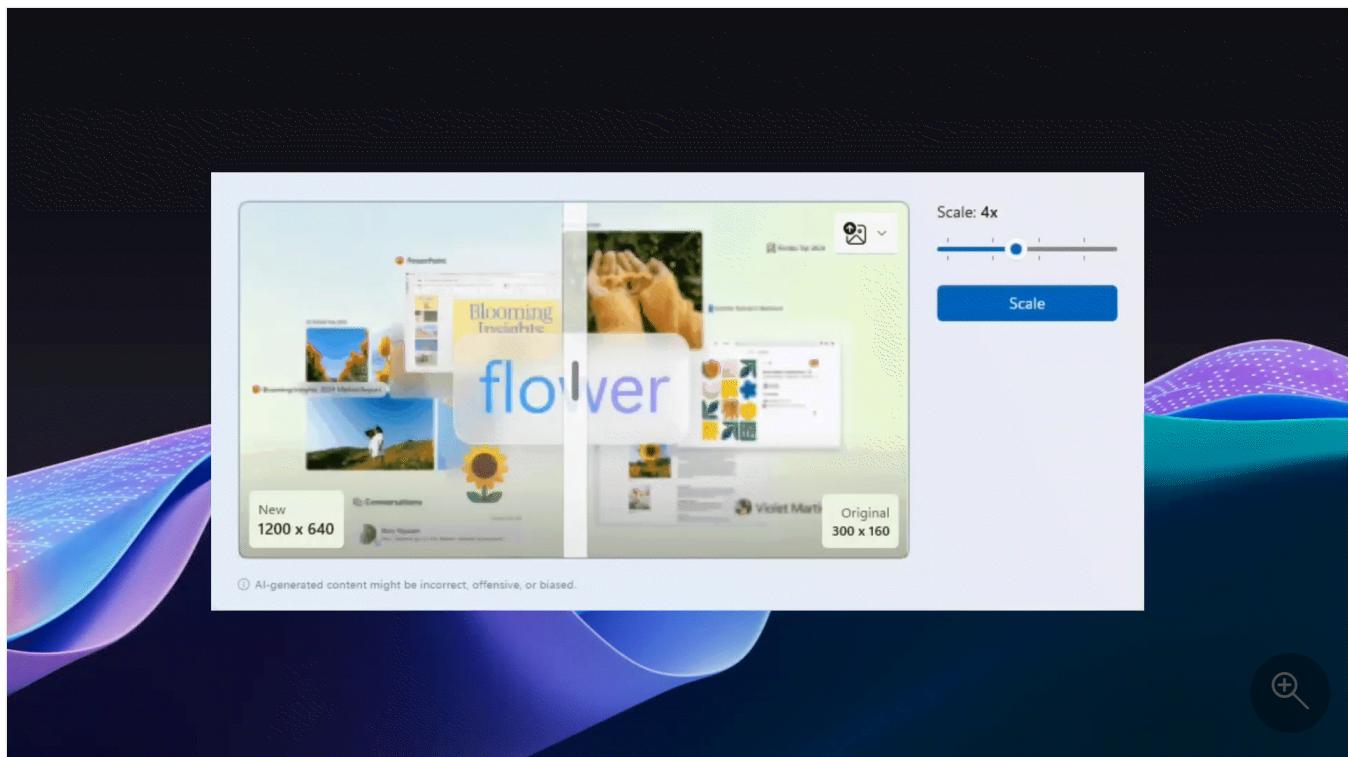
[Try it in AI Dev Gallery](#)

Imaging

Scale and sharpen images (Image Super Resolution), identify objects within an image (Image Object Extractor), generate natural-language descriptions of images (Image Description), and remove objects from images (Object Erase). See [Get Started with AI imaging](#).

Image Super Resolution

The Image Super Resolution APIs enable image sharpening and scaling.

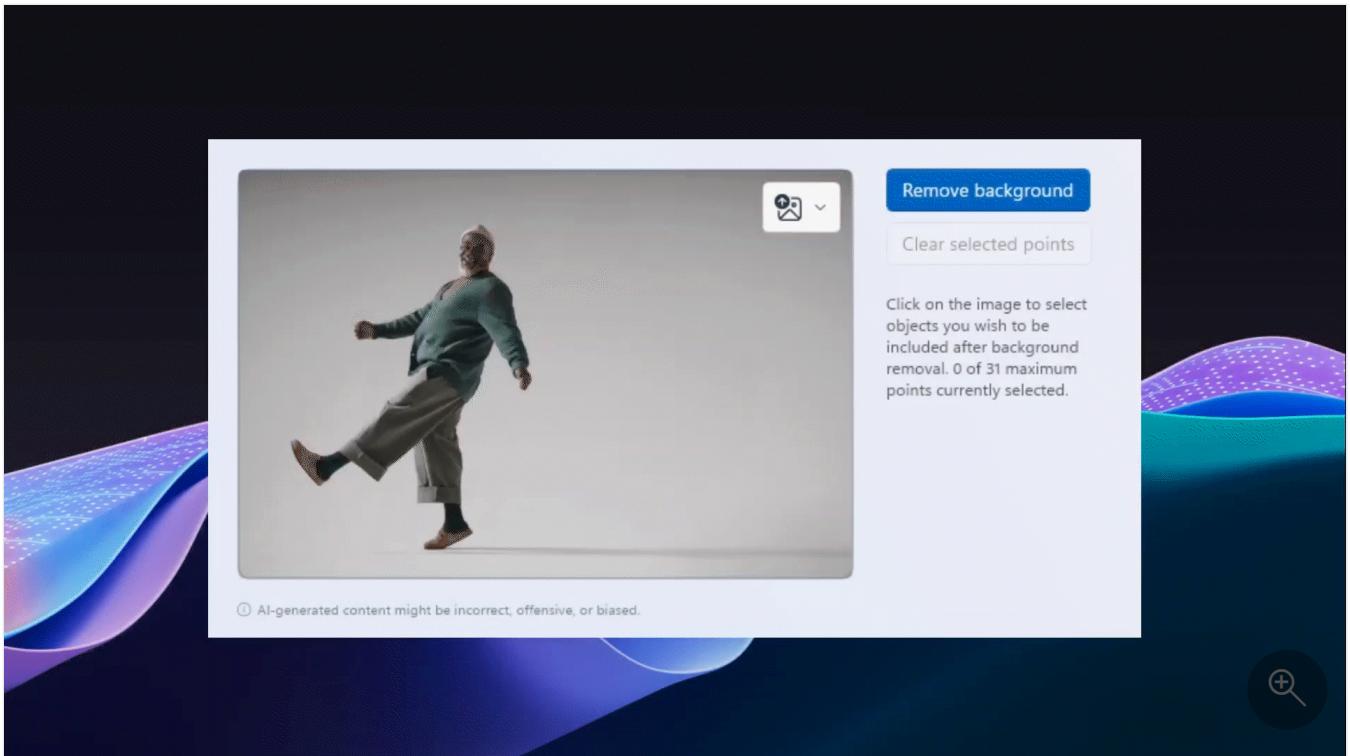


[Try it in AI Dev Gallery](#)

Also see [Image Super Resolution](#).

Image Object Extractor

The Image Object Extractor APIs enable identifying objects within images.



[Try it in AI Dev Gallery](#)

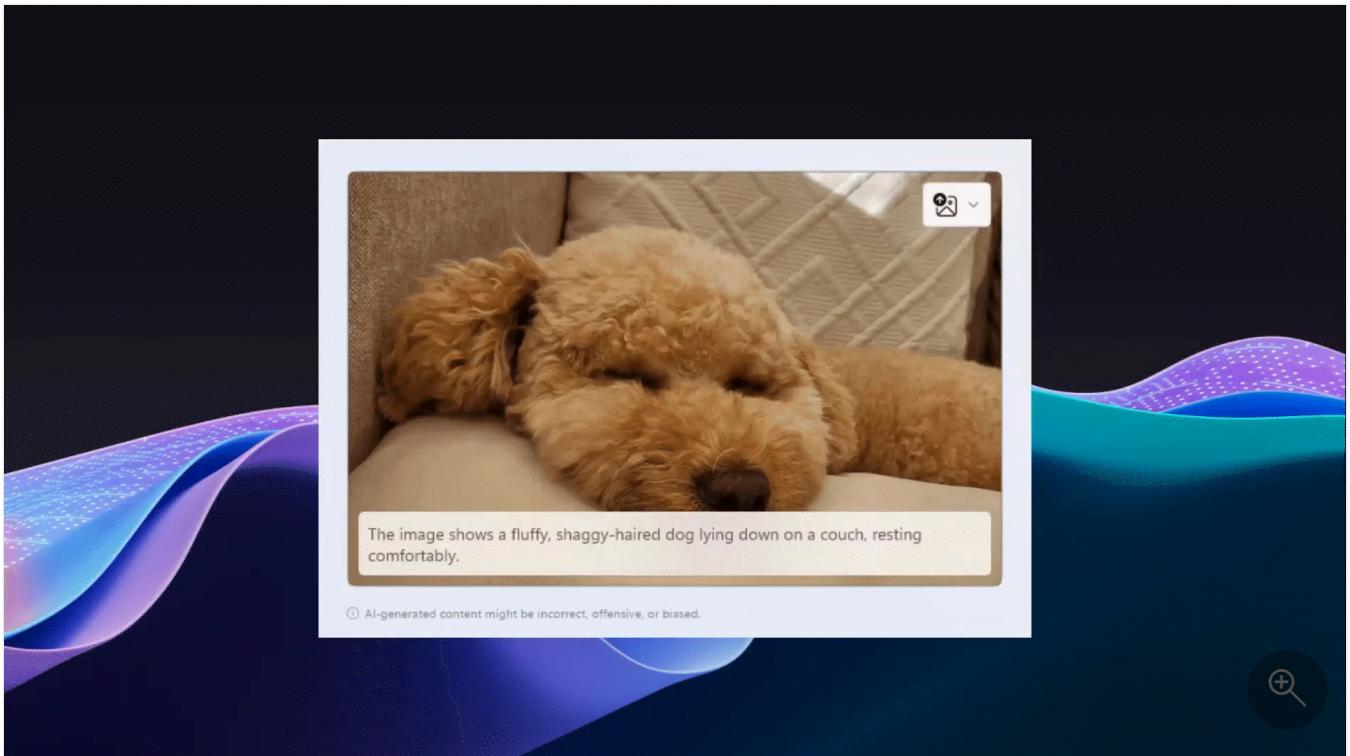
Also see [Image Object Extractor](#).

Image Description

The Image Description APIs describes images in natural language.

 **Note**

Image Description features are not available in China.



[Try it in AI Dev Gallery](#)

Also see [Image Description](#)

Object Erase

You can use the Object Erase APIs to remove objects from images.



[Try it in AI Dev Gallery](#)

Also see [Object Erase](#)

Additional AI features

- **Live Caption Translations (Not yet supported).** Help everyone using Windows—including those who are deaf or hard of hearing—better understand audio by viewing captions of spoken content (even when the audio content is in a language that's different from the system's preferred language).

Content moderation

Learn how content is moderated by the Windows AI APIs, and how you can adjust sensitivity filters. See [Content safety moderation with the Windows AI APIs](#).

When utilizing AI features, we recommend that you review: [Developing Responsible Generative AI Applications and Features on Windows](#).

Additional resources

- [Code samples and tutorials](#). A collection of samples that demonstrate a variety of ways to use AI to enhance your Windows apps.
- [Integrate AI in enterprise apps using Windows AI APIs](#). Watch the demo session from the November 2024 Microsoft Ignite conference.
- Provide **feedback** on these APIs and their functionality by creating a [new Issue](#) in the Windows App SDK GitHub repo or by responding to an [existing issue](#).

See also

- [AI Dev Gallery](#)
- [Windows AI API samples](#)

Last updated on 01/24/2026

Get started building an app with Windows AI APIs

Learn about the Windows AI API hardware requirements and how to configure your device to successfully build apps using the Windows AI APIs.

Dependencies

Ensure that your PC supports Windows AI APIs and that all dependencies are installed. You can choose to do this automatically (recommended) or manually.

Automated dependency installation (recommended)

1. Confirm that your device is a Copilot+ PC (we recommend the devices listed in the [Copilot+ PCs developer guide](#)).
2. Run the following command in [Windows Terminal](#).

Windows Command Prompt

```
winget configure https://raw.githubusercontent.com/microsoft/winget-dsc/refs/heads/main/samples/Configuration%20files/Learn%20tutorials/Windows%20AI/learn_wcr.winget
```

This runs a [WinGet Configuration file](#) that performs the following tasks:

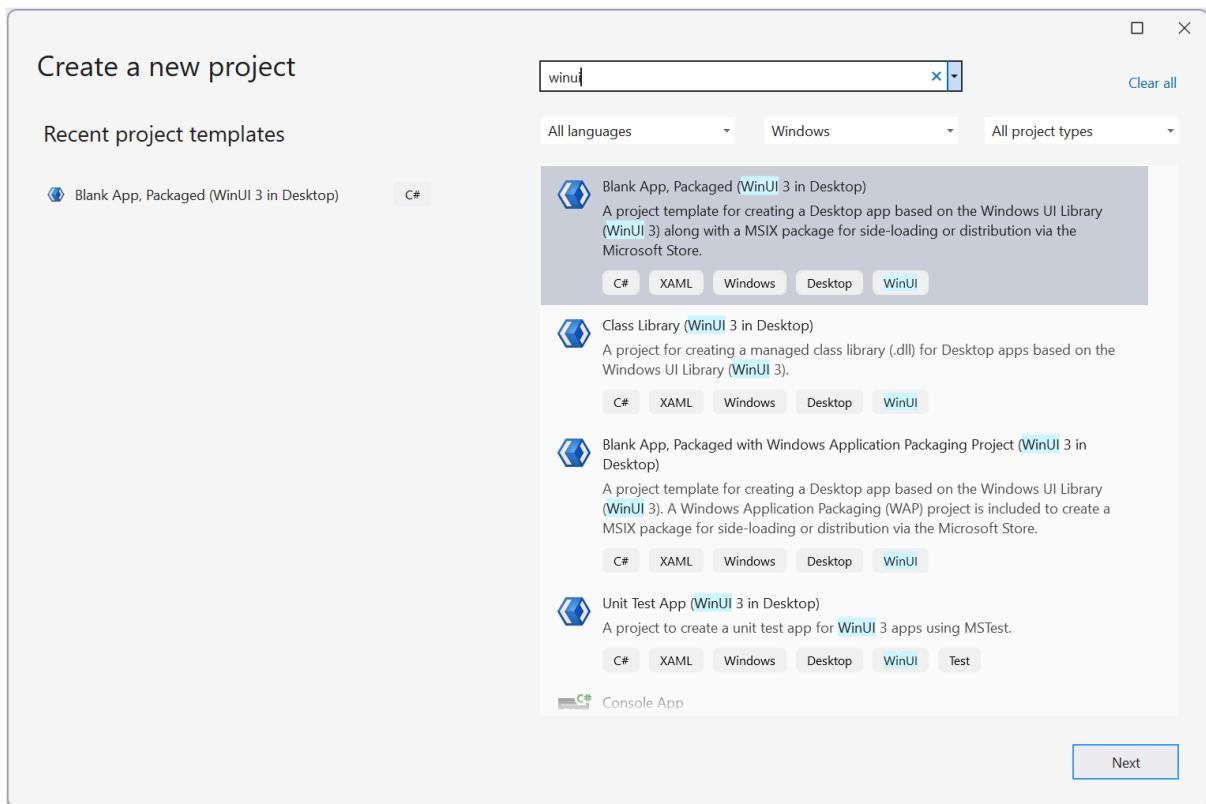
- Checks for minimum OS version.
- Enables Developer Mode.
- Installs Visual Studio Community Edition with WinUI and other required workloads.
- Installs the Windows App SDK.

Build a new app

The following steps describe how to build an app that uses Windows AI APIs (select the tab for your preferred UI framework).

WinUI

1. In Visual Studio, create a new WinUI project by selecting the **Blank App, Packaged (WinUI 3 in Desktop)** template.



2. In **Solution Explorer**, right-click the project node, select **Properties > Application > General**, and ensure that the target framework is set to **.NET 8.0**, and the target OS is set to **10.0.22621 or later**.

Search properties

Application

General

Win32 Resources
Dependencies
Packaging

Global Usings
Build
Package
Code Analysis
Debug
Resources

Output type
Specifies the type of application to build.
Windows Application

Target framework ?
Specifies the version of .NET that the application targets. This option can have different values depending on which versions of .NET are installed on your computer.
.NET 8.0

[Install other frameworks](#)

Target OS ?
Specifies the operating system that this project will target.
Windows

Target OS version ?
Specifies the version of the operating system this project will target.
10.0.22621.0

Supported OS version ?
Specifies the minimum OS version that the project will run on. When unspecified, the target OS version value is implied. Using an earlier version here requires code to add guards around later version APIs.
10.0.17763.0

Windows Forms ?
 Enable Windows Forms for this project.

3. Edit the Package.appxmanifest file (right click and select **View code**) and add the following snippets.

- The `systemAIModels` capability to the `<Capabilities>` node:

XML

```
<Capabilities>
    <systemai:Capability Name="systemAIModels"/>
</Capabilities>
```

- The `systemai` namespace specifier to "IgnorableNamespaces" in the `<Package>` node:

XML

```
xmlns:systemai="http://schemas.microsoft.com/appx/manifest/systemai/windows10"
IgnorableNamespaces="uap rescap systemai"
```

- The max version tested in the `TargetDeviceFamily` element of the `<Dependencies>` node needs to be at least 10.0.26226.0:

XML

```
<TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17763.0"
MaxVersionTested="10.0.26226.0" />
```

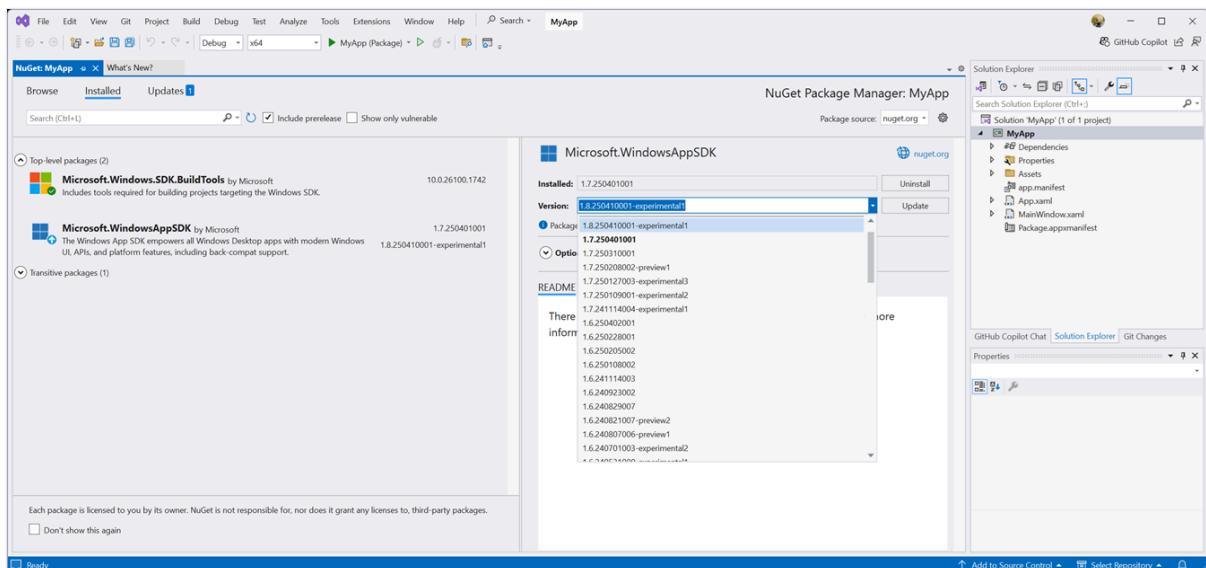
4. Add the following to your .waproj, .csproj, or .vcxproj file. This step is necessary to ensure visual studio doesn't override the max version tested.

XML

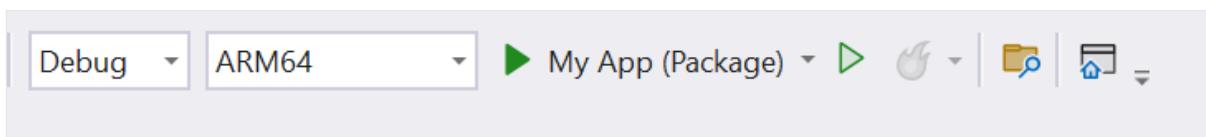
```
<AppxOSMinVersionReplaceManifestVersion>false</AppxOSMinVersionReplaceManifestVersion>
<AppxOSMaxVersionTestedReplaceManifestVersion>false</AppxOSMaxVersionTestedReplaceManifestVersion>
```

5. Right-click the project node and select **Manage NuGet Packages....**

6. In **NuGet Package Manager**, check the **Include prerelease** checkbox, and select Windows App SDK version **1.8.250410001-experimental1**. Click **Install** or **Update**.



7. Ensure that your build configuration is set to **ARM64**.



8. Build and run your app.

9. If the app launches successfully, then continue to [Add your first AI API](#). Otherwise, see [Troubleshooting](#).

Add your first AI API

When implementing a feature using Windows AI APIs, your app should first check for the availability of the AI model that supports that feature.

The following snippet shows how to check for model availability and generate a response.

WinUI

1. In `MainWindow.xaml`, add a `TextBlock` to display the `LanguageModel` response.

XML

```
<TextBlock x:Name="OutputText" HorizontalAlignment="Center"  
VerticalAlignment="Center" />
```

2. At the top of `MainWindow.xaml.cs`, add the following `using Microsoft.Windows.AI` directive.

C#

```
using Microsoft.Windows.AI;
```

3. In `MainWindow.xaml.cs`, replace the `MainWindow` class with the following code, which confirms the `LanguageModel` is available and then submits a prompt asking for the model to respond with the molecular formula of glucose.

C#

```
public sealed partial class MainWindow : Window  
{  
    public MainWindow()  
    {  
        this.InitializeComponent();  
        InitAI();  
    }  
  
    private async void InitAI()  
    {  
        OutputText.Text = "Loading..";  
  
        if (LanguageModel.GetReadyState() ==  
            AIFeatureReadyState.EnsureNeeded)  
        {  
            var result = await LanguageModel.EnsureReadyAsync();  
            if (result.Status != PackageDeploymentStatus.CompletedSuccess)  
            {  
                OutputText.Text = "Model failed to load";  
            }  
        }  
    }  
}
```

```
        throw new Exception(result.ExtendedError().Message);
    }
}

using LanguageModel languageModel =
    await LanguageModel.CreateAsync();

string prompt = "Provide the molecular formula of glucose.";
var result = await languageModel.GenerateResponseAsync(prompt);
OutputText.Text = result.Response;
}
}
```

4. Build and run the app.

5. The formula for glucose should appear in the text block.

Advanced tutorials and APIs

Now that you've successfully checked for model availability, explore the APIs further in the various Windows AI API tutorials.

- [Learn more about available Windows AI APIs](#)
- [Phi Silica API Walkthrough](#)
- [Text Recognition API Walkthrough](#)
- [Imaging API Walkthrough](#)

Troubleshooting

If you encounter any errors, it's typically because of your hardware or the absence of a required model.

- The **GetReadyState** method checks whether the model required by an AI feature is available on the user's device. You must call this method before any call to the model.
- If the model isn't available on the user's device, then you can call the method **EnsureReadyAsync** to install the required model. Model installation runs in the background, and the user can check the install progress on the **Windows Settings > Windows Update Settings** page.
- The **EnsureReadyAsync** method has a status option that can show a loading UI. If the user has unsupported hardware, then **EnsureReadyAsync** will fail with an error.

See [Windows AI API troubleshooting and FAQ](#) for more assistance.

See also

- [Developing responsible generative AI apps and features on Windows](#)
 - [API ref for AI imaging features](#)
 - [Windows App SDK](#)
 - [Latest release notes for the Windows App SDK](#)
 - [AI Dev Gallery ↗](#)
 - [Windows AI API samples ↗](#)
-

Last updated on 11/17/2025

Content safety moderation with the Windows AI APIs

Windows AI APIs, such as [Phi Silica](#) and [Imaging](#), use content moderation to classify and filter out potentially harmful content from user prompts or in responses returned by the generative models. By default, these API filter out content classified as potentially harmful, but sensitivity levels can be configured.

For API details, see [API ref for content safety moderation](#).

Prerequisites

Complete the steps in [Get started building an app with Windows AI APIs](#).

Adjust content safety moderation

You can adjust content moderation on the input prompt to the generative model and the AI generated output. Windows AI API content moderation is designed and implemented similarly to the one provided by [Azure AI Content Safety](#).

Harm categories

The harm categories supported by Windows AI APIs align with those defined by [Azure AI Content Safety](#). Harm categories include *Hate and fairness*, *Sexual*, *Violence*, and *Self-harm* (multiple categories can be assigned to the same content).

[] Expand table

Category	Description	API name
Hate	Content that attacks or uses discriminatory language with reference to a person or identity group based on certain differentiating attributes of these groups.	<code>HateContentSeverity</code>
Sexual	Content related to anatomical organs and genitals, romantic relationships and sexual acts, acts portrayed in erotic or affectionate terms, including those portrayed as an assault or a forced sexual violent act against one's will.	<code>SexualContentSeverity</code>
Violence	Content related to physical actions intended to hurt, injure, damage, or kill someone or something; describes weapons, guns, and related entities.	<code>ViolentContentSeverity</code>

Category	Description	API name
Self harm	Content related to physical actions intended to purposely hurt, injure, damage one's body or kill oneself.	SelfHarmContentSeverity

Severity levels

By default, all calls to generative Windows AI APIs use content moderation, but severity levels can be adjusted.

- `high`: Not available. Content classified as severity level 3+ (high-risk for potential harm) is currently blocked from being returned by the generative AI model.
- `medium`: The default severity level is set to `medium`. Content classified as severity level 0 - 3 will be returned.
- `low`: Further lowers the risk of returning potentially harmful content. Only content classified as severity level 0 - 1 will be returned.

To learn more about severity levels, see [Azure AI Content Safety Harm Categories](#).

Text Content Moderation code sample

To configure the Text Content Moderation severity filters, you must pass the [ContentFilterOptions](#) struct as a parameter to the API used for response generation (such as the [Phi Silica API](#)).

The following code sample demonstrates adding Text Content Moderation severity filters to the Microsoft Windows Generative AI [LanguageModel](#):

C#

```
LanguageModelOptions options = new LanguageModelOptions();
ContentFilterOptions filterOptions = new ContentFilterOptions();

// prompt
filterOptions.PromptMaxAllowedSeverityLevel.Violent = SeverityLevel.Minimum;
filterOptions.PromptMaxAllowedSeverityLevel.Hate = SeverityLevel.Low;
filterOptions.PromptMaxAllowedSeverityLevel.SelfHarm = SeverityLevel.Medium;
filterOptions.PromptMaxAllowedSeverityLevel.Sexual = SeverityLevel.High;

//response
filterOptions.ResponseMaxAllowedSeverityLevel.Violent = SeverityLevel.Medium;

//image
filterOptions.ImageMaxAllowedSeverityLevel.AdultContentLevel = SeverityLevel.Medium;
```

```
filterOptions.ImageMaxAllowedSeverityLevel.RacyContentLevel = SeverityLevel.Medium;  
options.ContentFilterOptions = filterOptions;  
  
var result = await languageModel.GenerateResponseAsync(prompt, options);  
  
Console.WriteLine(result.Text);
```

See also

- [Developing Responsible Generative AI Applications and Features on Windows](#)
- [API ref for Phi Silica in the Windows App SDK](#)
- [API ref for AI imaging features](#)
- [Windows App SDK](#)
- [Latest release notes for the Windows App SDK](#)
- [AI Dev Gallery ↗](#)
- [Windows AI API samples ↗](#)

Last updated on 11/18/2025

Windows AI API troubleshooting

Important

Apps using the Phi Silica APIs might encounter issues with Limited Access Feature support (see [LimitedAccessFeatures](#)).

At this time, we recommend using [experimental releases](#) as they do not require LAF tokens.

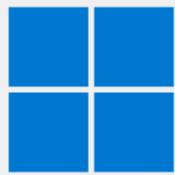
Requirements

Hardware

Phi Silica features require a Copilot+ PC (laptop keyboards have a dedicated Copilot key). See [Copilot+ PCs developer guide](#) for a list of recommended devices.

Software

Windows 11, version 25H2 (build 10.0.26200.7309) or later must be installed on the Copilot+ PC. This can be verified by opening the Start menu, typing "winver", and pressing Enter (a dialog box with the OS details, similar to the following, should appear).



Windows 11

Microsoft Windows

Version 25H2 (OS Build 26200.7392)

© Microsoft Corporation. All rights reserved.

The Windows 11 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.

This product is licensed under the [Microsoft Software License Terms](#) to:

user name

org name

OK

Ensure that you can use Phi Silica features:

1. Download and install the AI Dev Gallery app from Microsoft Store.
2. Run the app.
3. Select **AI APIs** from the left pane.
4. Click on **Phi Silica > Text Generation**.
5. Confirm that Phi Silica responds to a prompt (the sample app starts with a default prompt).

The screenshot shows the Windows AI APIs AI Dev Gallery interface. The left sidebar lists categories like Home, Samples, Models, Contribute, and Settings. The main area is titled "Text Generation" under "Windows AI APIs". It features a text input field with placeholder "Tell me a fun fact about space.", a "Generate" button, and a large text preview area containing a response about Venus's day length. Below the preview is a note about AI-generated content being incorrect, offensive, or biased. The bottom section contains API code samples in C# and JavaScript, documentation links, and a note about the Phi Silica APIs being a Limited Access Feature.

Other recommendations

Enable **Developer Mode** in **Settings > System > For developers > Developer Mode**.

Install [Visual Studio](#) with the workloads and components required for developing with WinUI and the Windows App SDK. For more details, see [Required workloads and components](#).

Debugging

If Phi Silica does not appear to be working on your system, there are a few things you can do.

1. Verify that you have all required models installed by going to **Settings > System > AI Components**, which shows the AI models available on the system. If the required model is not listed, go to **Settings > Windows Update** and click "Check for updates" (a restart might be required).
2. Verify that your app is using a valid WinAppSDK minimum version.
 - Stable: 1.8.3
 - Experimental: 2.0-Exp3

3. Check **Settings > Updates** to ensure Windows updates aren't paused.
4. Collect traces through the Windows Feedback Hub app and share them with the Windows team to review.
 - a. Open the Feedback Hub app (Windows key + F).
 - b. Sign in with your preferred account.
 - c. Enter bug details in section 1 (prefix the bug title with "[Windows AI APIs]").
 - d. In section 2, click **Problem**, select **Developer Platform** from the first drop down, and then select **Windows AI Foundry** from the second drop down.
 - e. Section 3 will display any existing feedback that is similar to yours. If existing feedback is not found, select **New feedback** and click **Next**.
 - f. In section 4 you can:
 - i. Click **Recreate my problem**, then click **Start recording**, run AI Dev Gallery, reproduce the issue, stop recording.
 - ii. Include additional info and attachments (such as images). Provide as much detail as possible, including reproduction steps and error messages.
 - g. Submit the feedback.

Other guidance

- Apps that use Windows AI APIs need to be granted package identity at runtime. For details, see [Advantages and disadvantages of packaging your app](#). If you are seeing an **UnauthorizedAccessException** error (or having other access issues), ensure that your app is packaged and the **systemAIModels** capability was added to your manifest file (see [Get started building an app with Windows AI APIs](#)).
- Limited Access Feature (LAF) requirements:
 - Verify that you've requested and received a LAF token from Microsoft
 - The following status values indicate that LAF has failed.
 - [LimitedAccessFeatureStatus.Unavailable](#)
 - [LimitedAccessFeatureStatus.Unknown](#)
- Self-contained apps cannot run from the **Downloads** folder (or from anywhere under the `C:\Users` folder). For more details, see [Advantages and disadvantages of packaging your app](#) and [Windows App SDK deployment overview](#).
- The Windows App SDK experimental channel includes APIs and features in early stages of development. All APIs in the experimental channel are subject to extensive revisions and breaking changes and may be removed from subsequent releases at any time. Experimental features are not supported for use in production environments and apps that use them cannot be published to the Microsoft Store.

- Provide feedback on these APIs and their functionality by creating a [new Issue ↗](#) in the Windows App SDK GitHub repo (include **Phi Silica** in the title) or by responding to an [existing issue ↗](#).

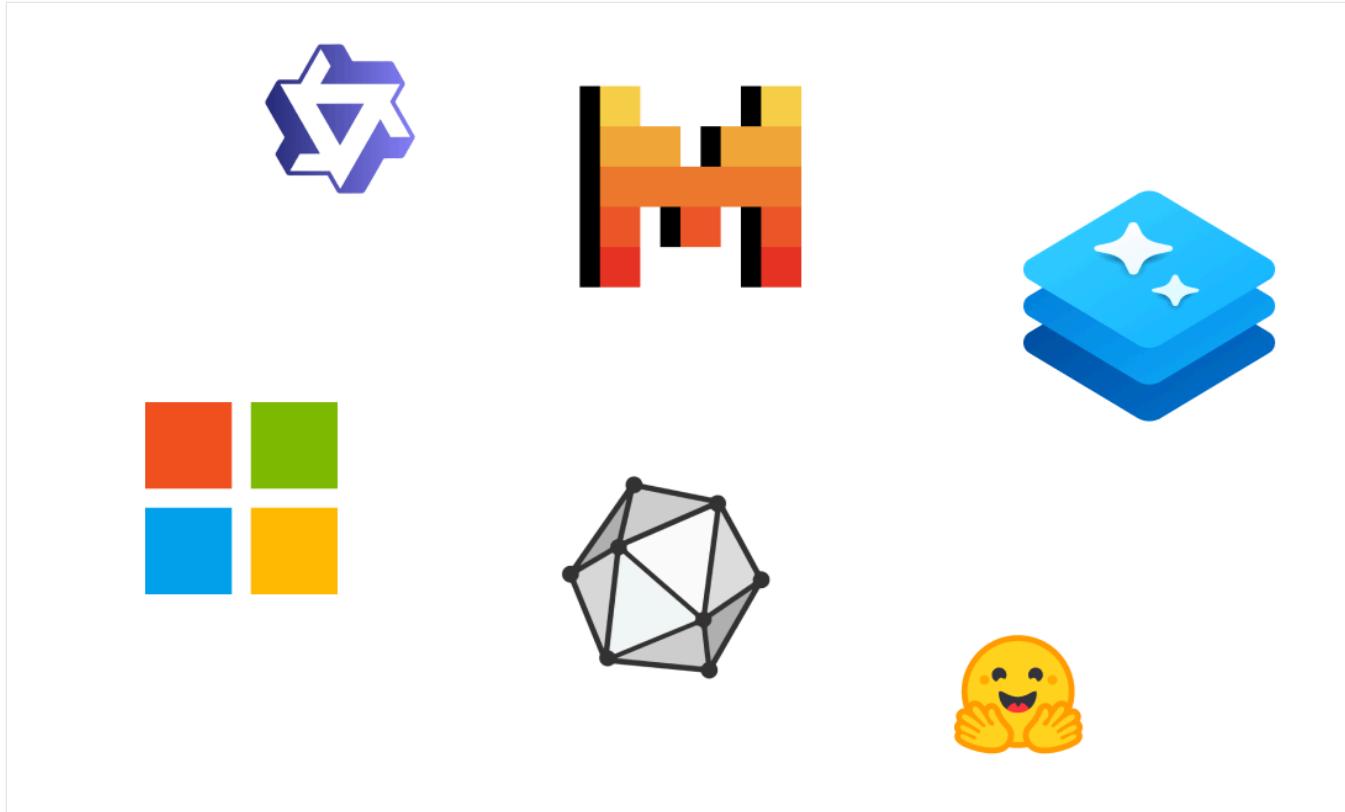
See also

- [AI Dev Gallery ↗](#)
- [Windows AI API samples ↗](#)

Last updated on 12/15/2025

Get started with Foundry Local

Foundry Local is a local version of Microsoft Foundry that enables local execution of large language models (LLMs) directly on your Windows device. This is a good alternative if you want to go deeper and implement an AI scenario that's not available with the Windows AI APIs.

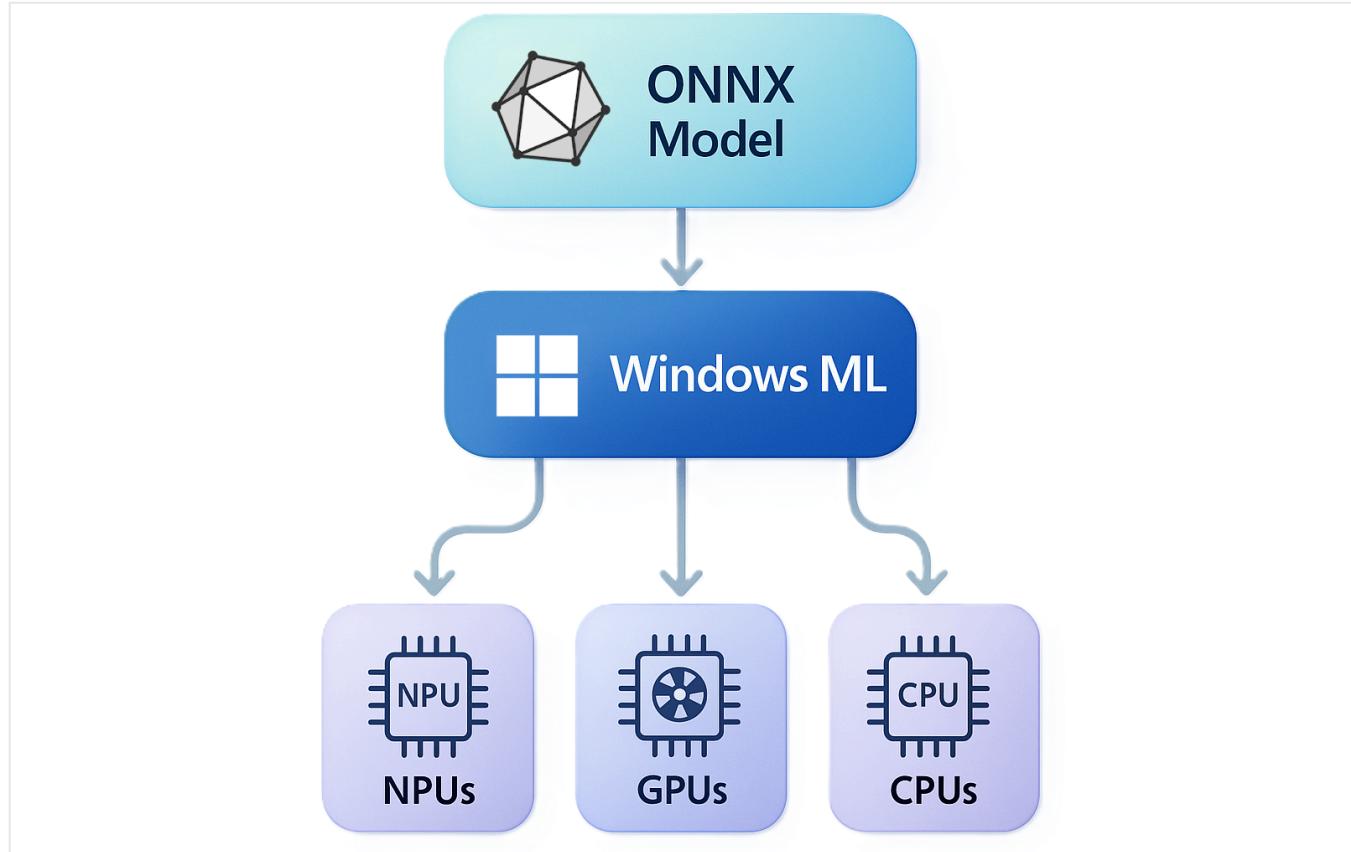


This on-device AI inference solution provides privacy, customization, and cost benefits compared to cloud-based alternatives. Best of all, it fits into your existing workflows and applications with an easy-to-use CLI and REST API. For more information on Foundry Local, see the [Foundry Local documentation](#).

Last updated on 11/18/2025

What is Windows ML?

Windows Machine Learning (ML) enables C#, C++, and Python developers to run ONNX AI models locally on Windows PCs via the [ONNX Runtime](#), with automatic execution provider management for different hardware (CPUs, GPUs, NPUs). You can use models from PyTorch, Tensorflow/Keras, TFLite, scikit-learn, and other frameworks with ONNX Runtime.



If you're not already familiar with the ONNX Runtime, we suggest reading the [ONNX Runtime docs](#). In short, Windows ML provides a shared Windows-wide copy of the ONNX Runtime, plus the ability to dynamically download execution providers (EPs).

Key benefits

- **Dynamically get latest EPs** - Automatically downloads and manages the latest hardware-specific execution providers
- **Shared ONNX Runtime** - Uses system-wide runtime instead of bundling your own, reducing app size
- **Smaller downloads/installs** - No need to carry large EPs and the ONNX Runtime in your app
- **Broad hardware support** - Runs on Windows PCs (x64 and ARM64) and Windows Server with any hardware configuration

System requirements

- **OS:** Version of Windows that [Windows App SDK supports](#)
- **Architecture:** x64 or ARM64
- **Hardware:** Any PC configuration (CPUs, integrated/discrete GPUs, NPUs)

What is an execution provider?

An execution provider (EP) is a component that enables hardware-specific optimizations for machine learning (ML) operations. Execution providers abstract different compute backends (CPU, GPU, specialized accelerators) and provide a unified interface for graph partitioning, kernel registration, and operator execution. To learn more, see the [ONNX Runtime docs](#).

You can [see the list of EPs that Windows ML supports here](#).

How it works

Windows ML includes a copy of the [ONNX Runtime](#) and allows you to dynamically download vendor-specific **execution providers** (EPs), so your model inference can be optimized across the wide variety of CPUs, GPUs, and NPUs in the Windows ecosystem.

Automatic deployment

1. **App installation** - Windows App SDK bootstrapper initializes Windows ML
2. **Hardware detection** - Runtime identifies available processors
3. **EP download** - Automatically downloads optimal execution providers
4. **Ready to run** - Your app can immediately use AI models

This eliminates the need to:

- Bundle execution providers for specific hardware vendors
- Create separate app builds for different execution providers
- Handle execution provider updates manually

Note

You're still responsible for optimizing your models for different hardware. Windows ML handles execution provider distribution, not model optimization. See [AI Toolkit](#) and the [ONNX Runtime Tutorials](#) for more info on optimization.

Performance optimization

The latest version of Windows ML works directly with dedicated execution providers for GPUs and NPUs, delivering to-the-metal performance that's on par with dedicated SDKs of the past such as TensorRT for RTX, AI Engine Direct, and Intel's Extension for PyTorch. We've engineered Windows ML to have best-in-class GPU and NPU performance, while retaining the write-once-run-anywhere benefits that the previous DirectML-based solution offered.

Using execution providers with Windows ML

The Windows ML runtime provides a flexible way to access machine learning (ML) execution providers (EPs), which can optimize ML model inference on different hardware configurations. Those EPs are distributed as separate packages that can be updated independently from the operating system. See the [initialize execution providers with Windows ML](#) docs for more info about dynamically downloading and registering EPs.

Converting models to ONNX

You can convert models from other formats to ONNX so that you can use them with Windows ML. See the Visual Studio Code AI Toolkit's docs about how to [convert models to the ONNX format](#) to learn more. Also see the [ONNX Runtime Tutorials](#) for more info on converting PyTorch, TensorFlow, and Hugging Face models to ONNX.

Model management

Windows ML provides flexible options for managing AI models:

- [Model Catalog](#) - Dynamically download models from online catalogs without bundling large files
- [Local models](#) - Include model files directly in your application package

Integration with Windows AI ecosystem

Windows ML serves as the foundation for the broader Windows AI platform:

- [Windows AI APIs](#) - Built-in models for common tasks
- [Foundry Local](#) - Ready-to-use AI models
- [Custom models](#) - Direct Windows ML API access for advanced scenarios

Providing feedback

Found an issue or have suggestions? Search or create issues on the [Windows App SDK GitHub](#).

Next steps

- **Get started:** [Get started with Windows ML](#)
- **Model management:** [Model Catalog overview](#)
- **Learn more:** [ONNX Runtime documentation](#)
- **Convert models:** [VS Code AI Toolkit model conversion](#)

Last updated on 01/27/2026

Get started with Windows ML

This topic shows you how to install and use Windows ML to discover, download, and register execution providers (EPs) for use with the ONNX Runtime shipped with Windows ML. Windows ML handles the complexity of package management and hardware selection, automatically downloading the latest execution providers compatible with your device's hardware.

If you're not already familiar with the ONNX Runtime, we suggest reading the [ONNX Runtime docs](#). In short, Windows ML provides a shared Windows-wide copy of the ONNX Runtime, plus the ability to dynamically download execution providers (EPs).

Prerequisites

- Version of Windows that [Windows App SDK supports](#)
- Language-specific prerequisites seen below

C#

- .NET 6 or greater
- Targeting a Windows 10-specific TFM like `net6.0-windows10.0.19041.0` or greater

Step 1: Install or update the Windows App SDK

The Model Catalog APIs are included in the [experimental](#) version of [Windows App SDK 2.0.0 or greater](#).

C#

See [use the Windows App SDK in an existing project](#) for how to add the Windows App SDK to your project, or if you're already using Windows App SDK, update your packages.

Step 2: Download and register EPs

The simplest way to get started is to let Windows ML automatically discover, download, and register the latest version of all compatible execution providers. Execution providers need to be registered with the ONNX Runtime inside of Windows ML before you can use them. And if they haven't been downloaded yet, they need to be downloaded first. Calling `EnsureAndRegisterCertifiedAsync()` will do both of these in one step.

C#

```
using Microsoft.ML.OnnxRuntime;
using Microsoft.Windows.AI.MachineLearning;

// First we create a new instance of EnvironmentCreationOptions
EnvironmentCreationOptions envOptions = new()
{
    logId = "WinMLDemo", // Use an ID of your own choice
    logLevel = OrtLoggingLevel.ORT_LOGGING_LEVEL_ERROR
};

// And then use that to create the ORT environment
using var ortEnv = OrtEnv.CreateInstanceWithOptions(ref envOptions);

// Get the default ExecutionProviderCatalog
var catalog = ExecutionProviderCatalog.GetDefault();

// Ensure and register all compatible execution providers with ONNX Runtime
// This downloads any necessary components and registers them
await catalog.EnsureAndRegisterCertifiedAsync();
```

Tip

In production applications, wrap the `EnsureAndRegisterCertifiedAsync()` call in a try-catch block to handle potential network or download failures gracefully.

Next steps

After registering execution providers, you're ready to use the ONNX Runtime APIs within Windows ML! You will want to...

1. [Select execution providers](#) - Tell the runtime which execution providers you want to use
2. [Get your models](#) - Use Model Catalog to dynamically download models, or include them locally
3. [Run model inference](#) - Compile, load, and inference your model

See also

- [Model Catalog](#) - Dynamically download models from online catalogs
- [Initialize execution providers](#) - Additional ways you can handle download of EPs

- [Distribute your app](#) - Info about distributing an app using Windows ML
 - [ONNX versions in Windows ML](#) - Info about which ONNX Runtime version ships with Windows ML
 - [Tutorial](#) - Full end-to-end tutorial using Windows ML with the ResNet-50 model
 - [Code samples ↗](#) - Our code samples using Windows ML
-

Last updated on 01/09/2026

Find or train models for Windows ML

Windows ML works with ONNX format models, since Windows ML is simply a distribution mechanism providing the ONNX Runtime and hardware-specific execution providers. This means you can use millions of existing pre-trained models from various sources, or train your own models. This guide covers where to find, convert, or train ONNX models.

 Expand table

Options	Details
1. Use models from AI Toolkit	Choose from over 20+ OSS models (including LLMs and other types of models) that are ready-to-optimize for use with Windows ML using AI Toolkit's Conversion tool
2. Use other existing ONNX models	Browse over 30,000+ pre-trained ONNX models from Hugging Face or other sources
3. Convert existing models to ONNX format	Browse over 2,400,000+ pre-trained PyTorch / TensorFlow / etc models from Hugging Face or other sources and convert them to ONNX
4. Fine-tune existing models	Fine-tune over 2,400,000+ pre-trained PyTorch / TensorFlow / etc models from Hugging Face or other sources to work better for your scenario (and convert them to ONNX format)
5. Train models	Train your own models in PyTorch, TensorFlow, or other frameworks, and convert them to ONNX

You can also choose from dozens of ready-to-use AI models and APIs in Microsoft Foundry on Windows, which run via Windows ML. See [Use local AI with Microsoft Foundry on Windows](#) to learn more.

Option 1: Use models from AI Toolkit

With [AI Toolkit's Conversion tool](#), there are dozens of LLMs and other types of models that are ready-to-optimize for use with Windows ML. By obtaining a model through AI Toolkit, you'll get a converted ONNX model that is optimized for the variety of hardware that Windows ML runs on.

To browse the available models, see [AI Toolkit's Model List](#).

Option 2: Use other existing ONNX models

[Hugging Face](#) hosts thousands of ONNX models that you can use with Windows ML. You can find ONNX models by:

1. Browsing the [Hugging Face Model Hub](#)
2. Filtering by "ONNX" in the library filter

You will need to find a model that is compatible with the ONNX Runtime version included in the version of Windows ML you are using. See [ONNX Runtime versions shipped in Windows ML](#) to find out what version of ONNX Runtime you are using with Windows ML.

Option 3: Convert existing models to ONNX format

Models from PyTorch, TensorFlow, or other frameworks can be converted to ONNX format and used with Windows ML.

[Hugging Face](#) hosts millions of models that you can convert and use with Windows ML.

You will need to convert the model to run with the ONNX Runtime version included in the version of Windows ML you are using. See [ONNX Runtime versions shipped in Windows ML](#) to find out what version of ONNX Runtime you are using with Windows ML.

To convert a model to ONNX format, see framework-specific documentation, for example:

- [Convert PyTorch models tutorial](#)
- [Convert TensorFlow models tutorial](#)

Option 4: Fine-tune existing models

Many models on [Hugging Face](#) or other sources can be fine-tuned (following instructions on the model cards on Hugging Face). You can then subsequently convert the fine-tuned model to ONNX following the instructions in Option 3 above.

A popular way to fine-tune models is to use the [olive finetune command](#). See the [Olive documentation](#) to learn more about using Olive.

Option 5: Train models

If you need a model for a specific task and can't find an existing model, you can train your own in PyTorch, TensorFlow, or other frameworks.

Once you've trained your model, follow the instructions in Option 3 above to convert your model to ONNX format.

Next steps

Once you have an ONNX model, you can run it with Windows ML on your target devices.

- [Initialize execution providers](#) - Download and register execution providers in Windows ML
- [Run ONNX models](#) - Learn how to run inference with Windows ML

Other solutions

As part of Microsoft Foundry on Windows, you can also choose from dozens of ready-to-use AI models and APIs, which run via Windows ML. See [Use local AI with Microsoft Foundry on Windows](#) to learn more.

Last updated on 01/24/2026

Supported execution providers in Windows ML

Windows ML supports the following execution providers. To learn more about execution providers, [see the ONNX Runtime docs ↗](#).

Included execution providers

The following execution providers are included with the ONNX Runtime that ships with Windows ML:

- CPU
- DirectML ↗

Available execution providers

The execution providers listed below are available on **Windows 11 PCs running version 24H2 (build 26100) or greater** (depending on device and driver compatibility) for dynamic download and registration via the Windows ML `ExecutionProviderCatalog` APIs (see [Initialize execution providers](#)). Updated versions of the execution providers are made available via [Windows Update's optional nonsecurity preview releases, a.k.a. "D week releases"](#).

Before your app uses an execution provider, please be sure to read the licenses corresponding to the execution provider.

MIGraphX (AMD)

- **EpName:** `"MIGraphXExecutionProvider"`
- **Windows App SDK compatible versions:** 1.8.3 - 2.0.0-experimental3
- **Requirements:**
 - GPU with version 25.10.13.09 (exactly)
 - *This execution provider is not supported for GenAI scenarios today.*
- **Documentation:** [Documentation ↗](#)
- **Support:** [Support ↗](#)
- **License terms:** [Ryzen AI Licensing Information ↗](#)

▼ Release history

 Note

Release dates are in the format of "2025 11D", or "[YEAR] [MONTH][WEEK]". "2025 11D" means it was released on the "D" week (4th week) of November 2025.

Previews are released to Windows Insiders and are often released on the "A" week (1st week) of the month. Users must be in the Windows Insider program to get those versions.

[+] Expand table

Version	Windows Update release
1.8.43.0	2026 1D
1.8.35.0	2025 11D

NvTensorRtRtx (NVIDIA)

- **EpName:** "NvTensorRtRtxExecutionProvider"
- **Windows App SDK compatible versions:** 1.8.1 - 2.0.0-experimental3
- **Requirements:**
 - NVIDIA GeForce RTX 30XX and above with minimum recommended driver version 32.0.15.5585 + Cuda version 12.5
- **Documentation:** [Documentation ↗](#)
- **Support:** [Support ↗](#)
- **License terms:** [eula-12Aug2025.pdf ↗](#) and [License Agreement for NVIDIA Software Development Kits — EULA ↗](#)

▼ Release history

! Note

Release dates are in the format of "2025 11D", or "[YEAR] [MONTH][WEEK]". "2025 11D" means it was released on the "D" week (4th week) of November 2025.

Previews are released to Windows Insiders and are often released on the "A" week (1st week) of the month. Users must be in the Windows Insider program to get those versions.

[+] Expand table

Version	Windows Update release
1.8.22.0	2026 1D

Version	Windows Update release
1.8.14.0	2025 9D

OpenVINO (Intel)

- **EpName:** "OpenVINOExecutionProvider"
- **Windows App SDK compatible versions:** 1.8.1 - 2.0.0-experimental4
- **Requirements:**
 - CPU: Intel TigerLake (11th Gen) and above with min recommended driver 32.0.100.9565
 - GPU: Intel AlderLake (12th Gen) and above with min recommended driver 32.0.101.1029
 - NPU: Intel ArrowLake (15th Gen) and above with min recommended driver 32.0.100.4239
- **Documentation:** [Documentation ↗](#)
- **Support:** [Support ↗](#)
- **License terms:** [Intel OBL Distribution Commercial Use License Agreement v2025.02.12 ↗](#)

▼ Release history

(!) Note

Release dates are in the format of "2025 11D", or "[YEAR] [MONTH][WEEK]". "2025 11D" means it was released on the "D" week (4th week) of November 2025.

Previews are released to Windows Insiders and are often released on the "A" week (1st week) of the month. Users must be in the Windows Insider program to get those versions.

[] [Expand table](#)

Version	Windows Update release
1.8.63.0	2026 1D
1.8.26.0	2025 11D
1.8.18.0	2025 10D
1.8.15.0	2025 9D

QNN (Qualcomm)

- EpName: "QNNExecutionProvider"
- Windows App SDK compatible versions: 1.8.1 - 2.0.0-experimental3
- Requirements:
 - Snapdragon(R) X Elite - X1Exxxx
 - Qualcomm(R) Hexagon(TM) NPU with minimum driver version 30.0.140.0 and above
 - Snapdragon(R) X Plus - X1Pxxxx
 - Qualcomm(R) Hexagon(TM) NPU with minimum driver version 30.0.140.0 and above
- Documentation: [Documentation ↗](#)
- Support: [Support ↗](#)
- License terms: To view the QNN License, [download the Qualcomm® Neural Processing SDK ↗](#), extract the ZIP, and open the *LICENSE.pdf* file.

▼ Release history

⚠ Note

Release dates are in the format of "2025 11D", or "[YEAR] [MONTH][WEEK]". "2025 11D" means it was released on the "D" week (4th week) of November 2025.

Previews are released to Windows Insiders and are often released on the "A" week (1st week) of the month. Users must be in the Windows Insider program to get those versions.

[+] [Expand table](#)

Version	Windows Update release
1.8.30.0	2026 1D
1.8.21.0	2025 11D
1.8.14.0	2025 10D
1.8.13.0	2025 9D

VitisAI (AMD)

- EpName: "VitisAIExecutionProvider"
- Windows App SDK compatible versions: 1.8.1 - 2.0.0-experimental3
- Requirements:
 - Min: Adrenalin Edition 25.6.3 with NPU driver 32.00.0203.280

- Max: Adrenalin Edition 25.9.1 with NPU driver 32.00.0203.297
- Documentation: [Documentation ↗](#)
- Support: [Support ↗](#)
- License terms: [Ryzen AI Licensing Information ↗](#)

▼ Release history

!**Note**

Release dates are in the format of "2025 11D", or "[YEAR] [MONTH][WEEK]". "2025 11D" means it was released on the "D" week (4th week) of November 2025.

Previews are released to Windows Insiders and are often released on the "A" week (1st week) of the month. Users must be in the Windows Insider program to get those versions.

 [Expand table](#)

Version	Windows Update release
1.8.50.0	2026 1D
1.8.43.0	2025 11D (Windows Insiders)
1.8.31.0	2025 11A (Windows Insiders)
1.8.26.0	2025 10D
1.8.24.0	2025 9D

See also

- [Initialize execution providers](#)
- [Select execution providers](#)
- [Distribute your app that uses Windows ML](#)

Last updated on 01/31/2026

Initialize execution providers with Windows ML

This page discusses more advanced ways your app can gracefully handle downloading and registering execution providers (EPs) using Windows ML. Even if an EP is already downloaded on the device, you must register the EPs every time your app runs so they will appear in ONNX Runtime.

Download and register in one call

For initial development, it can be nice to simply call `EnsureAndRegisterCertifiedAsync()`, which will ensure EPs compatible with your device are present (and will download the EPs if they're not present), and then it registers all present EPs with the ONNX Runtime. Note that on first run, this method can take multiple seconds or even minutes depending on your network speed and EPs that need to be downloaded.

C#

```
// Get the default ExecutionProviderCatalog
var catalog = ExecutionProviderCatalog.GetDefault();

// Ensure execution providers compatible with device are present (downloads if
// necessary)
// and then registers all present execution providers with ONNX Runtime
await catalog.EnsureAndRegisterCertifiedAsync();
```

💡 Tip

In production applications, wrap the `EnsureAndRegisterCertifiedAsync()` call in a try-catch block to handle potential network or download failures gracefully.

Register existing providers only

If you want to avoid downloading and only register execution providers that are already present on the machine:

C#

C#

```
var catalog = ExecutionProviderCatalog.GetDefault();

// Register only providers already present on the machine
// This avoids potentially long download times
await catalog.RegisterCertifiedAsync();
```

Discover if there are EPs (without downloading)

If you want to see if there are not-present EPs compatible with your device and drivers, but don't want to start the download, you can use the `FindAllProviders()` method and see if any providers have a **ReadyState** of **NotPresent**. You can then decide to handle this however you would like (launching your users into an "Installing screen", asking them if they want to install, etc). You can choose to continue using any already-downloaded EPs (by calling `RegisterCertifiedAsync()` as shown above) if you don't want to make your users wait right now.

C#

```
var catalog = ExecutionProviderCatalog.GetDefault();

// Check if there are new EPs that need to be downloaded
if (catalog.FindAllProviders().Any(provider => provider.ReadyState ==
ExecutionProviderReadyState.NotPresent))
{
    // TODO: There are new EPs, decide how your app wants to handle that
}
else
{
    // All EPs are already present, just register them
    await catalog.RegisterCertifiedAsync();
}
```

Download and register a specific EP

If there is a specific execution provider your app wants to use, you can download and register a particular execution provider without downloading all compatible EPs.

You'll first use `FindAllProviders()` to get all compatible EPs, and then you can call `EnsureReadyAsync()` on a particular `ExecutionProvider` to download the specific execution provider, and call `TryRegister()` to register the specific execution provider.

C#

```
C#  
  
var catalog = ExecutionProviderCatalog.GetDefault();  
  
// Get the QNN provider, if present  
var qnnProvider = catalog.FindAllProviders()  
    .FirstOrDefault(i => i.Name == "QNNExecutionProvider");  
  
if (qnnProvider != null)  
{  
    // Download it  
    var result = await qnnProvider.EnsureReadyAsync();  
  
    // If download succeeded  
    if (result != null && result.Status ==  
        ExecutionProviderReadyResultState.Success)  
    {  
        // Register it  
        bool registered = qnnProvider.TryRegister();  
    }  
}
```

Production app example

For production applications, here's an example of what your app might want to do to give yourself and your users control over when downloads occur. You can check if new execution providers are available and conditionally download them:

C#

```
C#  
  
using Microsoft.Windows.AI.MachineLearning;  
  
var catalog = ExecutionProviderCatalog.GetDefault();
```

```

// Filter to the EPs our app supports/uses
var providers = catalog.FindAllProviders().Where(p =>
    p.Name == "MIGraphXExecutionProvider" ||
    p.Name == "VitisAIEExecutionProvider" ||
    p.Name == "OpenVINOExecutionProvider" ||
    p.Name == "QNNExecutionProvider" ||
    p.Name == "NvTensorRtRtxExecutionProvider"
);

if (providers.Any(p => p.ReadyState == ExecutionProviderReadyState.NotPresent))
{
    // Show UI to user asking if they want to download new execution providers
    bool userWantsToDownload = await ShowDownloadDialogAsync();

    if (userWantsToDownload)
    {
        // Download all EPs
        foreach (var p in providers)
        {
            if (p.ReadyState == ExecutionProviderReadyState.NotPresent)
            {
                // Ignore result handling here; production code could inspect
                status
                await p.EnsureReadyAsync();
            }
        }

        // And register all EPs
        await catalog.RegisterCertifiedAsync();
    }
    else
    {
        // Register only already-present EPs
        await catalog.RegisterCertifiedAsync();
    }
}

```

See also

- [Supported execution providers and release history](#)
- [Update execution providers](#)
- [Check execution provider versions](#)
- [Common execution provider download issues](#)

Update execution providers in Windows ML

After an execution provider (EP) is installed (as seen in [initialize execution providers](#)), the execution provider is updated through Windows Update's optional nonsecurity preview releases, a.k.a. "D week releases".

These compatible updates are automatically made available to your app, without your app manually updating. This allows apps to benefit from performance improvements and support for new operators without requiring changes to your app.

There is currently no way for your app to programmatically install an updated EP. The updated EP will automatically be installed through Windows Update.

To see the release history of EPs, see [supported execution providers](#) and expand the "Release history" section underneath each available EP.

To check what version of EP a device has, see [check execution provider versions](#).

See also

- [Supported execution providers and release history](#)
- [Initialize execution providers](#)
- [Check execution provider versions](#)
- [Common execution provider download issues](#)

Last updated on 01/09/2026

Check execution provider versions in Windows ML

Most execution providers in Windows ML are dynamically acquired via Windows Update at runtime as seen in [initialize execution providers](#), and updated versions are automatically updated (with compatible updates) as described in [update execution providers](#), meaning the version of the EP might vary over time.

See the [supported execution providers docs](#) to see what execution providers are available and their release history.

Check your end-user's EP version

You can programmatically check the version of an execution provider (EP) that's present on the device by inspecting the [Packageld](#) property on [ExecutionProvider](#).

If the EP is not yet present, Packageld will return null.

C#

```
// Get all EPs compatible with this device
var providers = ExecutionProviderCatalog.GetDefault().FindAllProviders();

// For each provider
foreach (var provider in providers)
{
    // Log the name
    Debug.WriteLine($"Windows ML EP: {provider.Name}");

    // Log the version
    if (provider.PackageId != null)
    {
        var v = provider.PackageId.Version;
        Debug.WriteLine($"Version: {v.Major}.{v.Minor}.{v.Build}.{v.Revision}");
    }
    else
    {
        Debug.WriteLine("Version: Not installed");
    }
}
```

On a device with the QNN EP installed, this code outputs the following...

Windows ML EP: QNNExecutionProvider
Version: 1.8.27.0

Check your own device's EP version

You can also easily check which version of an EP is installed on your development device by using PowerShell.

PowerShell

```
Get-AppxPackage MicrosoftCorporationII.WinML.*
```

On a device with the QNN EP installed, this outputs the following...

PowerShell

```
Name          : MicrosoftCorporationII.WinML.Qualcomm.QNN.EP.1.8
Publisher    : CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond,
S=Washington, C=US
Architecture  : Arm64
ResourceId   :
Version      : 1.8.27.0
...
```

See also

- [Supported execution providers and release history](#)
- [Initialize execution providers](#)
- [Update execution providers](#)
- [Common execution provider download issues](#)

Last updated on 01/09/2026

Troubleshoot execution provider download issues with Windows ML

There are some cases where you may see an error downloading and registering execution providers to use with Windows ML. This page covers common issues and potential ways you can resolve the issue.

If you can't resolve the issue, then please see the [How to file feedback](#) section below. A Feedback Hub will automatically capture the logs that our team needs in order to investigate your issue.

Upstream updates

In some cases, upstream Windows Updates will require a device reboot to resume EP download. If you encounter download errors, check if your device has pending updates that require a restart.

Solution: Restart your device to complete any pending Windows Updates, then try downloading the execution providers again.

Windows updates paused

Devices with Windows Updates paused will not be able to download execution providers. Execution provider downloads rely on the Windows Update infrastructure to deliver components.

Solution: To resume Windows Updates, navigate to **Settings > Windows Update > Resume Updates**.

Managed devices

In some cases, enterprise policies may prevent execution providers from being downloaded on managed devices. IT administrators may have configured policies that restrict component downloads or limit access to Windows Update.

Solution: Contact your IT administrator to verify whether enterprise policies are blocking execution provider downloads. They may need to adjust policies to allow these components.

How to file feedback

If your issue wasn't resolved with the above suggestions, then you can file a report through Feedback Hub.

1. Launch Feedback Hub.

- Select Start, enter Feedback Hub in the search bar, and select the Feedback Hub app from the list.

2. Select "Report a problem".

3. Describe your issue.

4. Select **Developer Platform > Windows Machine Learning** for the category.

5. Follow the steps to reproduce your problem and submit your issue.

See also

- [Initialize execution providers with Windows ML](#)
- [Get started with Windows ML](#)

Last updated on 01/24/2026

ONNX Runtime versions shipped in Windows ML

Each Windows App SDK release includes Windows ML, which includes a copy of the ONNX Runtime, so that your app can depend on a shared system-wide copy of the ONNX Runtime rather than distributing your own copy.

Versions of ONNX Runtime in Windows ML

The following table clarifies which ONNX Runtime commit was shipped with each Windows App SDK release (which contains Windows ML).

[+] Expand table

Windows App SDK version	Windows App SDK release date	ONNX Runtime commit hash	ONNX Runtime date
2.0.0-Experimental4	1/13/2026	d5379f5 ⓘ (~1.24.0)	12/4/2025
1.8.4	1/13/2026	a83fc4d ⓘ (~1.23.2)	10/21/2025
1.8.3	11/11/2025	a83fc4d ⓘ (~1.23.2)	10/21/2025
1.8.2	10/14/2025	d9b2048 ⓘ (~1.23.1)	9/26/2025
1.8.1	9/22/2025	a922003 ⓘ (~1.23.0)	9/10/2025
1.8.0-Experimental4	7/8/2025	1.22.0 (with minor changes)	5/9/2025

Automatic updates for framework-dependent apps

If your app uses the [framework-dependent version](#) of Windows App SDK, your app will automatically receive updates across the revision version number without re-compiling and updating your app, but not across minor or major versions.

The following table shows how automatic updates work across different Windows App SDK version number changes:

[+] Expand table

Version Component	Example Change	Automatic Update?	Description
Revision (x.y.Z)	1.8.0 → 1.8.1	<input checked="" type="checkbox"/> Yes	Bug fixes and patches - automatically applied
Revision (x.y.Z)	1.8.1 → 1.8.2	<input checked="" type="checkbox"/> Yes	Bug fixes and patches - automatically applied
Minor (x.Y.z)	1.8.2 → 1.9.0	<input type="checkbox"/> No	Significant update - requires manual update
Major (X.y.z)	1.9.0 → 2.0.0	<input type="checkbox"/> No	Breaking changes - requires manual update

Version breakdown examples

[\[+\] Expand table](#)

App Targets	Latest Available	App Actually Uses	Update Type
1.8.0	1.8.3	1.8.3	<input checked="" type="checkbox"/> Automatic (revision)
1.8.0	1.9.0	1.8.3*	<input type="checkbox"/> Manual required (minor)
1.8.0	2.0.0	1.8.3*	<input type="checkbox"/> Manual required (major)

*Latest revision within the same minor version

This means that if you target 1.8.0 of Windows App SDK and 1.8.1 is released, your app will automatically use 1.8.1 (and the corresponding Windows ML ONNX Runtime version). However, when 1.9.0 is released, your app will continue using 1.8.1 until you manually update your app to target 1.9.0.

Last updated on 01/13/2026

Use ONNX APIs in Windows ML

Windows Machine Learning (ML) includes a shared copy of the [ONNX Runtime](#), including its APIs. That means when you install Windows ML via Windows App SDK, your app will have access to the full ONNX API surface.

To see what version of ONNX Runtime is included in specific Windows ML versions, see [ONNX Runtime versions shipped in Windows ML](#).

This page covers how to use the ONNX APIs included in Windows ML.

Prerequisites

- Follow all the steps in [Get started with Windows ML](#).

Namespaces / headers

The namespaces / headers for the ONNX APIs within Windows ML are as follows:

C#

In C#, the namespaces of the ONNX APIs are the same as when using ONNX Runtime directly.

C#

```
using Microsoft.ML.OnnxRuntime;
```

APIs

The ONNX APIs are the same as when using ONNX Runtime directly. For example, to create an inference session:

C#

```
// Create inference session using compiled model
using InferenceSession session = new(compiledModelPath, sessionOptions);
```

We suggest reading the [ONNX Runtime docs](#) for more info about how to use the ONNX Runtime APIs within Windows ML.

See also

- [ONNX Runtime versions shipped in Windows ML](#)
 - [Select execution providers](#)
 - [Run ONNX models](#)
 - [Distribute your app that uses Windows ML](#)
-

Last updated on 09/23/2025

Select execution providers using the ONNX Runtime included in Windows ML

The ONNX Runtime shipped with Windows ML allows apps to configure execution providers (EPs) either based on [Device Policies](#) or explicitly, which provides more control over provider options and which devices should be used.

We recommend starting with explicit selection of EPs so that you can have more predictability in the results. After you have this working, you can experiment with [using Device Policies](#) to select execution providers in a natural, outcome-oriented way.

Explicit selection of EPs

To explicitly select an EP, use the environment's `GetEpDevices` function to enumerate all available devices, and select the EP devices you want to use. Then use `AppendExecutionProvider` (C#) or `AppendExecutionProvider_v2` (C++) to append specific devices and provide custom provider options to the desired EP. You can see all of our [supported EPs here](#).

Important

List of devices can dynamically change: The list of EpDevices can dynamically change at runtime when Windows ML execution providers are automatically updated, or when drivers update. Your code should be resilient to handle new or unexpected EP devices appearing, or EP devices you were previously using no longer being present.

C#

```
C#  
  
using Microsoft.ML.OnnxRuntime;  
using System;  
using System.Linq;  
using System.Collections.Generic;  
  
// Assuming you've created an OrtEnv named 'ortEnv'  
// 1. Enumerate devices  
var epDevices = ortEnv.GetEpDevices();  
  
// 2. Filter to your desired execution provider and device type  
var selectedEpDevices = epDevices  
    .Where(d =>  
        d.EpName == "ReplaceWithExecutionProvider"
```

```

    && d.HardwareDevice.Type == OrtHardwareDeviceType.NPU)
    .ToList();

if (selectedEpDevices.Count == 0)
{
    throw new InvalidOperationException("ReplaceWithExecutionProvider is not
available on this system.");
}

// 3. Configure provider-specific options (varies based on EP)
// and append the EP with the correct devices (varies based on EP)
var sessionOptions = new SessionOptions();
var epOptions = new Dictionary<string, string>{ ["provider_specific_option"] =
"4" };
sessionOptions.AppendExecutionProvider(ortEnv, new[] { selectedEpDevices.First() },
epOptions);

```

Browse all available EPs in the [supported EPs](#) docs. For more details on EP selection, see the [ONNX Runtime OrtApi documentation](#).

Using Device Policies for execution provider selection

In addition to explicitly selecting EPs, you can also use Device Policies, which are a natural, outcome-oriented way to specify how you want your AI workload to run. To do this, use `SessionOptions.SetEpSelectionPolicy`, passing in `OrtExecutionProviderDevicePolicy` values. There are a variety of values you can use for automatic selection, like `MAX_PERFORMANCE`, `PREFER_NPU`, `MAX EFFICIENCY`, and more. See the [ONNX OrtExecutionProviderDevicePolicy docs](#) for other values you can use.

C#

```

C#
// Configure the session to select an EP and device for MAX_EFFICIENCY which
// typically
// will choose an NPU if available with a CPU fallback.
var sessionOptions = new SessionOptions();
sessionOptions.SetEpSelectionPolicy(ExecutionProviderDevicePolicy.MAX_EFFICIENCY
);

```

Next steps

After selecting execution providers, you're ready to [run inference on a model using ONNX Runtime!](#)

Last updated on 11/25/2025

Run ONNX models using the ONNX Runtime included in Windows ML

The ONNX Runtime shipped with Windows ML allows apps to run inference on ONNX models locally.

If you're using Generative AI models like Large Language Models (LLMs) and speech-to-text, see [Run LLMs and other generative models](#).

Creating an inference session

The APIs are the same as when using ONNX Runtime directly. For example, to create an inference session:

C#

C#

```
// Create inference session using compiled model
using InferenceSession session = new(compiledModelPath, sessionOptions);
```

We suggest reading the [ONNX Runtime docs](#) for more info about how to use the ONNX Runtime APIs within Windows ML. Model inference code will be different for every model.

Compile models

Before using an ONNX model in an inference session, it often must be compiled into an optimized representation that can be executed efficiently on the device's underlying hardware.

As of ONNX Runtime 1.22, there are new APIs that better encapsulate the compilation steps. More details are available in the ONNX Runtime compile documentation (see [OrtCompileApi struct](#)).

C#

C#

```
// Prepare compilation options
OrtModelCompilationOptions compileOptions = new(sessionOptions);
compileOptions.SetInputModelPath(modelPath);
```

```
compileOptions.SetOutputModelPath(compiledModelPath);  
  
// Compile the model  
compileOptions.CompileModel();
```

Note

Compilation can take several minutes to complete. So that any UI remains responsive, consider doing this as a background operation in your application.

Tip

For optimal performance, compile your models once, and reuse the compiled version. Store compiled models in your app's local data folder for subsequent runs. Note that updates to the EPs or runtime might require recompiling.

See also

- [Run GenAI ONNX models](#)
- [Use ONNX APIs in Windows ML](#)
- [ONNX Runtime versions shipped in Windows ML](#)
- [Initialize execution providers](#)
- [Select execution providers](#)
- [Distribute your app that uses Windows ML](#)

Last updated on 01/24/2026

Run LLMs and other generative models using ONNX Runtime and Windows ML

When using Generative AI (GenAI) models like Large Language Models (LLMs) or generative speech-to-text models in Windows ML, you can use the [ONNX Runtime GenAI Windows ML library](#) with Windows ML.

The GenAI API gives you an easy, flexible and performant way of running generative models on device. It implements the generative AI loop for ONNX models, including pre and post processing, inference with ONNX Runtime, logits processing, search and sampling, KV cache management, and grammar specification for tool calling.

Supported models

The GenAI API supports various LLMs and generative speech-to-text models.

See the [onnxruntime-genai GitHub](#) page for more details.

Installation

Install the [ONNX Runtime GenAI Windows ML NuGet package](#) in your project.

Usage

See the [ONNX Runtime GenAI docs](#) for more details.

- [C# API reference](#)
- [C++ API reference](#)

See also

- [Run ONNX models](#)
- [Use ONNX APIs in Windows ML](#)
- [ONNX Runtime versions shipped in Windows ML](#)
- [Initialize execution providers](#)
- [Select execution providers](#)
- [Distribute your app that uses Windows ML](#)

Use multiple versions of ONNX Runtime in an app

Some apps might need to use multiple different versions of ONNX Runtime. For example, some of your models might require specific ONNX Runtime versions, or your app might run unknown plugins from developers where the plugin developer needs to control their own ONNX Runtime version.

In these cases, you can run Windows ML's ONNX Runtime alongside another ONNX Runtime by running Windows ML in a separate process.

As an example of how to do this, say you wanted to take advantage of Windows ML's ability to run SqueezeNet. You would first build [this sample](#).

This sample accepts an image file as a command-line argument and outputs its interpretation of the image contents.

Within your own application code, you can launch that sample's process, collect its output, and use it in your app.

C#

```
// Spin up another process which uses Windows ML
var process = new System.Diagnostics.Process();
process.StartInfo.FileName = @"CSharpConsoleDesktop.exe";
process.StartInfo.Arguments = "--ep_policy CPU --image_path myimage.jpg";
process.StartInfo.RedirectStandardOutput = true;
process.StartInfo.UseShellExecute = false;
process.StartInfo.CreateNoWindow = true;

process.OutputDataReceived += (sender, e) =>
{
    // parse output
};

process.Start();
process.BeginOutputReadLine();
process.WaitForExit();
```

This method is effective for executing a model a single time which accepts a string as input and produces strings as output. For scenarios that require a persistent AI worker process or the exchange of data types other than strings, Windows offers a variety of [interprocess communication mechanisms](#) to support these needs.

See also

- ONNX Runtime versions shipped in Windows ML
 - Run ONNX models using the ONNX Runtime included in Windows ML
-

Last updated on 01/09/2026

Download and share models on-device with the Windows ML Model Catalog APIs

The Windows ML Model Catalog APIs allow your app or library to dynamically download large AI model files to a shared on-device location from your own online model catalogs without shipping those large files directly with your app or library. Additionally, the model catalog will help filter which models are compatible with the Windows device it's running on, so that the right model is downloaded to the device.

What are the Model Catalog APIs?

The Model Catalog APIs are a set of APIs that can be connected to one or many cloud model catalogs to facilitate downloading and storing those models locally on the device so that they can be used by any Windows applications on the device. The APIs have a few core features:

- **Add catalogs:** Add one or many online catalogs
- **Discover compatible models:** Automatically find models that work with the user's hardware and execution providers
- **Download models:** Download and store models from various sources
- **Share models across apps:** If multiple applications are requesting the same model (same SHA256 hash), the model will be shared on disk without duplicating downloads

Key features

Automatic compatibility matching

Model Catalog automatically matches models to your system's available execution providers (CPU, GPU, NPU, etc.). When you request a model, the catalog only returns models that are compatible with your current hardware configuration.

Model storage

Downloaded models are stored in a user-specific location. If multiple applications request the same model (same SHA256 hash), the already downloaded model will be shared among those applications.

Multiple catalog sources

Your application can configure multiple catalog sources, allowing you to:

- Use models from multiple vendors or repositories
- Prioritize certain sources over others
- Include your own private model catalogs alongside public ones

How it works

The Model Catalog system consists of several components:

1. **Catalog sources:** Define where models can be found (URLs to catalog JSON files)
2. **Model matching:** Filters available models based on execution provider compatibility
3. **Download management:** Handles download and storage of model files
4. **Instance management:** Provides access to downloaded models while your app is running

Model identification

Models in the catalog have two types of identifiers:

- **Name:** A common name like "gpt2" (multiple model variations can share the same name)
- **Id:** A unique-in-the-catalog identifier that typically includes execution provider information, like "gpt2-cpu" or "gpt2-npu"

Applications typically use `FindModelAsync` with the Name for simplicity, letting the catalog select the best available model variant for the current system based on execution provider compatibility.

Execution provider support

Model Catalog supports a variety of execution providers. See the [supported execution providers in Windows ML docs](#) for more info.

Catalog Source schema

Model catalog sources use a standardized JSON schema that defines:

- Model metadata (name, id, version, publisher)
- Supported execution providers
- Download URLs and file information
- License information
- Model size details

For detailed schema information, see [Model Catalog Source](#).

Getting started

To start using Model Catalog in your Windows ML application:

1. Configure your catalog sources
2. Create a `ModelCatalog` instance
3. Query and download models
4. Inference your models with your desired runtime!

For a complete walkthrough, see [Get started with Model Catalog](#).

Next steps

- [Get started with Model Catalog](#)
- [Model Catalog Source schema reference](#)
- [Windows ML overview](#)

Last updated on 11/12/2025

Get started with Windows ML Model Catalog APIs

This guide shows you how to use the Windows ML Model Catalog APIs to manage AI models in your Windows applications. You'll learn how to set up catalog sources, find compatible models, and download them to a shared location on the device for all apps to use.

Prerequisites

- Windows 10 PC running Windows 10 1809 or greater
- One or more model catalog sources
- [Windows App SDK 1.8.3 or greater](#) in your compatible project

Step 1: Create a model catalog source

You must first create or obtain a model catalog source, which is an index of models including information about how to access the model. See the [Model Catalog Source docs](#) for more information.

The Model Catalog Source JSON file can either be hosted online, referenced via `https://` endpoints, or used from a local file, referenced via file paths like `C:\Users\....`.

Step 2: Initialize your catalog with your source

Initialize a catalog with a single catalog source:

C#

```
using Microsoft.Windows.AI.MachineLearning;
using Windows.Foundation;
using System;
using System.Threading.Tasks;

// Create a catalog source from a URI
var source = await ModelCatalogSource.CreateFromUriAsync(
    new Uri("https://contoso.com/models"));

// Create the catalog with the source
var catalog = new ModelCatalog(new ModelCatalogSource[] { source });
```

You can also configure multiple catalog sources with different priorities:

C#

```
var catalog = new ModelCatalog(new ModelCatalogSource[0]);  
  
// Add sources in order of preference (highest priority first)  
catalog.Sources.Add(await ModelCatalogSource.CreateFromUriAsync(  
    new Uri("https://contoso.com/models")));  
catalog.Sources.Add(await ModelCatalogSource.CreateFromUriAsync(  
    new Uri("https://contoso.com/secondaryModels")));
```

Step 3: Find a model by name

Search for a model across all configured sources:

C#

```
// Find a model by its name  
CatalogModelInfo model = await catalog.FindModelAsync("phi-3.5-reasoning");  
  
if (model != null)  
{  
    Console.WriteLine($"Found model: {model.Name}");  
    Console.WriteLine($"Version: {model.Version}");  
    Console.WriteLine($"Publisher: {model.Publisher}");  
    Console.WriteLine($"Size: {model.ModelSizeInBytes / (1024 * 1024)} MB");  
    Console.WriteLine($"Supported execution providers: {string.Join(", ",  
        model.ExecutionProviders)}");  
}
```

Step 4: Download and use a model

Get a model instance and use it with your desired framework:

C#

```
// Get an instance of the model (downloads if necessary)  
var progress = new Progress<double>(percent =>  
    Console.WriteLine($"Download progress: {percent:P}"));  
  
CatalogModelInstanceResult result = await model.GetInstanceAsync().AsTask(progress);  
  
if (result.Status == CatalogModelInstanceState.Available)  
{  
    CatalogModelInstance instance = result.GetInstance();  
  
    // Get the model path  
    string modelPath = instance.ModelPaths[0];  
  
    Console.WriteLine($"Model path: {modelPath}");
```

```

    // Inference your model using your own code
}
else
{
    Console.WriteLine($"Failed to get model: {result.ExtendedError}");
    Console.WriteLine($"Details: {result.DiagnosticText}");
}

```

Complete example

Here's a complete example that puts it all together:

C#

```

using Microsoft.Windows.AI.MachineLearning;
using System;
using System.Threading.Tasks;

try
{
    // Create catalog with source
    var source = await ModelCatalogSource.CreateFromUriAsync(
        new Uri("https://contoso.com/models"));
    var catalog = new ModelCatalog(new ModelCatalogSource[] { source });

    // Find a model
    Console.WriteLine("Searching for model...");
    CatalogModelInfo model = await catalog.FindModelAsync("phi-3.5-reasoning");

    if (model != null)
    {
        Console.WriteLine($"Found model: {model.Name}");
        Console.WriteLine($"Version: {model.Version}");
        Console.WriteLine($"Publisher: {model.Publisher}");
        Console.WriteLine($"Size: {model.ModelSizeInBytes / (1024 * 1024)} MB");
        Console.WriteLine($"Supported execution providers: {string.Join(", ", 
model.ExecutionProviders)}");

        // Get an instance of the model (downloads if necessary)
        var progress = new Progress<double>(percent =>
            Console.WriteLine($"Download progress: {percent:P}"));

        CatalogModelInstanceResult result = await
model.GetInstanceAsync().AsTask(progress);

        if (result.Status == CatalogModelInstanceState.Available)
        {
            CatalogModelInstance instance = result.GetInstance();

            // Get the model path
            string modelPath = instance.ModelPaths[0];

```

```

        Console.WriteLine($"Model path: {modelPath}");

        // Inference your model using your own code
    }
    else
    {
        Console.WriteLine($"Failed to get model: {result.ExtendedError}");
        Console.WriteLine($"Details: {result.DiagnosticText}");
    }
}
else
{
    Console.WriteLine("Model not found");
}
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}

```

Filtering by execution providers

Customize which execution providers to consider:

C#

```

public async Task FilterByExecutionProvidersAsync(ModelCatalog catalog)
{
    // Only look for CPU-compatible models
    catalog.ExecutionProviders.Clear();
    catalog.ExecutionProviders.Add("cpuexecutionprovider");

    var cpuModels = await catalog.FindAllModelAsync();
    Console.WriteLine($"Found {cpuModels.Count} CPU-compatible models");

    // Look for DirectML-compatible models
    catalog.ExecutionProviders.Clear();
    catalog.ExecutionProviders.Add("dmlexecutionprovider");

    var dmlModels = await catalog.FindAllModelAsync();
    Console.WriteLine($"Found {dmlModels.Count} DirectML-compatible models");
}

```

Checking model status

Check if a model is already available locally:

C#

```

public void CheckModelStatus(ModelCatalog catalog)
{
    var availableModels = catalog.GetAvailableModels();

    foreach (var model in availableModels)
    {
        var status = model.GetStatus();
        Console.WriteLine($"Model {model.Name}: {status}");

        switch (status)
        {
            case CatalogModelStatus.Ready:
                Console.WriteLine(" ✓ Available locally");
                break;
            case CatalogModelStatus.NotReady:
                Console.WriteLine(" ⚠ Needs to be downloaded");
                break;
        }
    }
}

```

Using local catalog files

Create a catalog source from a local file:

C#

```

public async Task<ModelCatalog> CreateLocalCatalogAsync()
{
    // Load catalog from a local JSON file
    var localFile = Path.Combine(Package.Current.EffectivePath, "my-models.json");
    var source = await ModelCatalogSource.CreateFromUriAsync(new Uri(localFile));

    var catalog = new ModelCatalog(new ModelCatalogSource[] { source });
    return catalog;
}

```

Adding custom headers for downloads

Provide authentication or custom headers for model downloads:

C#

```

public async Task DownloadWithCustomHeadersAsync(CatalogModelInfo model)
{
    var headers = new Dictionary<string, string>
    {
        ["Authorization"] = "Bearer your-token-here",
        ["User-Agent"] = "MyApp/1.0"
    }
}

```

```

};

var result = await model.GetInstanceAsync(headers);
// Handle result...
}

```

Error handling

Always include proper error handling when working with Model Catalog:

C#

```

public async Task<bool> SafeModelUsageAsync(string modelName)
{
    try
    {
        var source = await ModelCatalogSource.CreateFromUriAsync(
            new Uri("https://contoso.com/models"));
        var catalog = new ModelCatalog(new ModelCatalogSource[] { source });

        var model = await catalog.FindModelAsync(modelName);
        if (model == null)
        {
            Console.WriteLine($"Model '{modelName}' not found");
            return false;
        }

        var result = await model.GetInstanceAsync();
        if (result.Status != CatalogModelInstanceState.Available)
        {
            Console.WriteLine($"Failed to get model: {result.ExtendedError}");
            if (!string.IsNullOrEmpty(result.DiagnosticText))
            {
                Console.WriteLine($"Details: {result.DiagnosticText}");
            }
            return false;
        }

        using var instance = result.GetInstance();
        // Use the model...
        return true;
    }
    catch (UnauthorizedAccessException)
    {
        Console.WriteLine("Access denied - check permissions");
        return false;
    }
    catch (System.Net.Http.HttpRequestException ex)
    {
        Console.WriteLine($"Network error: {ex.Message}");
        return false;
    }
}

```

```
        catch (Exception ex)
    {
        Console.WriteLine($"Unexpected error: {ex.Message}");
        return false;
    }
}
```

Best practices

1. **Reuse catalog instances:** Reuse `ModelCatalog` instances across your app
2. **Handle download progress:** Provide user feedback during model downloads
3. **Dispose model instances:** Use `using` statements to properly dispose of model instances
4. **Check compatibility:** Verify model execution providers match your requirements
5. **Handle failures gracefully:** Always check result status before using models
6. **Use model names:** Use `FindModelAsync` with model names for automatic selection of the best model variant based on device capabilities

Next steps

- Learn about [Model Catalog Source schema](#)
- Explore [Windows ML overview](#)

Last updated on 11/12/2025

Windows ML Model Catalog Source schema reference

This page provides reference documentation for the JSON schema used by Windows ML Model Catalog Source to define model catalog sources. Model catalog sources are JSON files that describe available AI models, their compatibility, and download information.

Your Model Catalog Source file can be hosted online at <https://> URLs, or referenced as a local file from your app.

Schema overview

A model catalog is a JSON file that contains an array of model definitions and optional metadata. The schema follows standard JSON Schema conventions and is designed to be both human-readable and machine-parsable.

The catalog supports two types of model distribution:

- **File-based models:** Models distributed as individual files (ONNX models, configs, etc.)
- **Package-based models:** Models distributed as Windows packages (MSIX)

Root catalog structure

JSON

```
{  
  "models": [  
    // Array of model objects  
  ]  
}
```

Root properties

[Expand table](#)

Property	Type	Required	Description
models	array	Yes	Array of model definitions

Model object structure

Each model in the `models` array follows this structure:

JSON

```
{  
  "id": "phi-3.5-cpu",  
  "name": "phi-3.5",  
  "version": "1.0.0",  
  "publisher": "Publisher Name",  
  "executionProviders": [  
    {  
      "name": "CPUExecutionProvider"  
    }  
  ],  
  "modelSizeBytes": 13632917530,  
  "license": "MIT",  
  "licenseUri": "https://opensource.org/licenses/MIT",  
  "licenseText": "License text content",  
  "uri": "https://models.example.com/model-base",  
  "files": [  
    {  
      "name": "model.onnx",  
      "uri": "https://models.example.com/model.onnx",  
      "sha256": "d7f93e79ba1284a3ff2b4cea317d79f3e98e64acfce725ad5f4e8197864aef73"  
    }  
  ]  
}
```

Model properties

[] [Expand table](#)

Property	Type	Required	Description
<code>id</code>	string	Yes	Unique-in-the-catalog identifier for this specific model
<code>name</code>	string	Yes	Common name of the model (can be shared across variants)
<code>version</code>	string	Yes	Model version number
<code>publisher</code>	string	Yes	Publisher or organization that created the model
<code>executionProviders</code>	array	Yes	Array of execution provider objects supported by the model
<code>modelSizeBytes</code>	integer	No	Size of the model in bytes (minimum: 0)
<code>license</code>	string	Yes	License type (e.g., "MIT", "BSD", "Apache")

Property	Type	Required	Description
licenseUri	string	No	URI to the license document
licenseText	string	No	Text content of the license
uri	string	No	Base URI where the model can be accessed
files	array	Conditional	Array of files associated with the model
packages	array	Conditional	Array of packages required for the model

Note: A model must have either `files` OR `packages`, but not both.

Execution providers

The `executionProviders` field is an array of execution provider objects. Each execution provider object must have at least a `name` property:

JSON

```
"executionProviders": [
  {
    "name": "CPUExecutionProvider"
  },
  {
    "name": "DmlExecutionProvider"
  }
]
```

See the [Supported execution providers in Windows ML docs page](#) for a comprehensive list of all available execution provider names.

File object

Files are used to distribute individual model components (ONNX files, configurations, etc.):

JSON

```
{
  "name": "model.onnx",
  "uri": "https://models.example.com/model.onnx",
  "sha256": "d7f93e79ba1284a3ff2b4cea317d79f3e98e64acfce725ad5f4e8197864aef73"
}
```

[+] Expand table

Property	Type	Required	Description
name	string	Yes	Name of the file
uri	string	No	URI where the file can be downloaded
sha256	string	Yes	SHA256 hash (64-character hex string) for integrity verification and deduping of identical models

Note: If `uri` is not specified, the file URI is constructed from the model's base `uri` property.

Package object

Packages are used to distribute models as Windows packages or applications:

JSON

```
{  
  "packageFamilyName": "Microsoft.Example_8wekyb3d8bbwe",  
  "uri": "https://example.com/packages/model.msix",  
  "sha256": "a1b2c3d4e5f6789..."  
}
```

[+] Expand table

Property	Type	Required	Description
packageFamilyName	string	Yes	Windows package family name identifier
uri	string	Yes	URI where the package can be obtained (HTTPS or local file path)
sha256	string	Conditional	SHA256 hash for integrity verification (required for HTTPS URLs)

Distribution methods

The catalog supports several distribution methods:

File-based distribution:

- Direct HTTPS downloads
- Files hosted on GitHub, Azure, or other web servers

- Model files (`.onnx`), configuration files (`.json`), and supporting assets

Package-based distribution:

- Direct package downloads via HTTPS
- Local file paths for packages
- MSIX packages

Complete examples

File-based model example

Here's an example catalog with file-based models:

JSON

```
{
  "models": [
    {
      "id": "squeezenet-v1",
      "name": "squeezenet",
      "version": "1.0",
      "publisher": "WindowsAppSDK",
      "executionProviders": [
        {
          "name": "CPUExecutionProvider"
        }
      ],
      "modelSizeBytes": 13632917530,
      "license": "BSD",
      "licenseUri": "https://github.com/microsoft/WindowsAppSDK-Samples/raw/main/LICENSE",
      "licenseText": "BSD 3-Clause License",
      "uri": "https://github.com/microsoft/WindowsAppSDK-Samples/raw/main/models",
      "files": [
        {
          "name": "SqueezeNet.onnx",
          "uri": "https://github.com/microsoft/WindowsAppSDK-Samples/raw/main/models/SqueezeNet.onnx",
          "sha256": "d7f93e79ba1284a3ff2b4cea317d79f3e98e64acfce725ad5f4e8197864aef73"
        },
        {
          "name": "Labels.txt",
          "uri": "https://github.com/microsoft/WindowsAppSDK-Samples/raw/main/models/Labels.txt",
          "sha256": "dc1fd0d4747096d3aa690bd65ec2f51fdb3e5117535bfbce46fa91088a8f93a9"
        }
      ]
    }
  ]
}
```

```
    }
]
}
```

Package-based model example

Here's an example catalog with package-based models:

JSON

```
{
  "models": [
    {
      "id": "example-packaged-model-cpu",
      "name": "example-packaged-model",
      "version": "2.0.0",
      "publisher": "Microsoft",
      "executionProviders": [
        {
          "name": "CPUExecutionProvider"
        },
        {
          "name": "DmlExecutionProvider"
        }
      ],
      "license": "MIT",
      "licenseUri": "https://opensource.org/licenses/MIT",
      "licenseText": "MIT License - see URI for full text",
      "packages": [
        {
          "packageFamilyName": "Microsoft.ExampleAI_8wekyb3d8bbwe",
          "uri": "https://example.com/packages/ExampleAI.msix",
          "sha256": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
        }
      ]
    }
  ]
}
```

Schema validation

The Windows ML Model Catalog follows the JSON Schema specification (draft 2020-12). You can validate your catalog files against the official schema to ensure compatibility.

Key validation rules

1. **Required fields:** Every model must have `id`, `name`, `version`, `publisher`, `executionProviders`, and `license`
2. **Exclusive distribution:** Models must specify either `files` OR `packages`, but not both
3. **SHA256 format:** All SHA256 hashes must be exactly 64 hexadecimal characters
4. **Execution providers:** Must be objects with at least a `name` property
5. **URI format:** All URLs must be properly formatted according to RFC 3986
6. **HTTPS package integrity:** Package downloads via HTTPS must include SHA256 hashes for integrity verification

Common validation errors

- **Missing required fields:** Ensure all required properties are present
- **Invalid SHA256:** Check that hash values are exactly 64 hex characters
- **Mixed distribution:** Don't specify both `files` and `packages` for the same model
- **Invalid URLs:** Verify all URLs are properly formatted and accessible
- **Missing SHA256 for HTTPS packages:** HTTPS package downloads require integrity verification

Creating your catalog

Step 1: Choose distribution method

Use file-based distribution when:

- Your models are standard ONNX files with supporting assets
- You have web hosting for model files
- You want simple HTTPS downloads
- Models are relatively small (< 1GB per file)

Use package-based distribution when:

- Your models require system integration
- Models include execution providers or dependencies
- You need Windows package management features
- You want to distribute via MSIX packages

Step 2: Structure your models

JSON

```
{  
  "models": [  
    {  
      "id": "unique-identifier-cpu",  
      "name": "unique-identifier",  
      "version": "1.0.0",  
      "publisher": "YourOrganization",  
      "executionProviders": [  
        {  
          "name": "CPUExecutionProvider"  
        }  
      ],  
      "license": "MIT",  
      "files": [] // or "packages": []  
    }  
  ]  
}
```

Step 3: Add distribution details

For file-based models:

JSON

```
"uri": "https://yourserver.com/models/base-path",  
"files": [  
  {  
    "name": "model.onnx",  
    "sha256": "your-calculated-sha256-hash"  
  }  
]
```

For package-based models:

JSON

```
"packages": [  
  {  
    "packageFamilyName": "YourPublisher.YourApp_randomstring",  
    "uri": "https://yourserver.com/packages/YourApp.msix",  
    "sha256": "your-calculated-sha256-hash"  
  }  
]
```

Step 4: Test your catalog

1. Validate JSON syntax using a JSON validator
2. Verify all URIs are accessible and return correct content

3. Check SHA256 hashes match the actual file contents
4. Test model downloading using the Windows ML Model Catalog APIs

Best practices

1. **Use semantic versioning:** Follow semantic versioning (e.g., "1.2.3") for the `version` field
2. **Provide accurate SHA256 hashes:** Always include correct SHA256 hashes for integrity verification
3. **Consistent naming:** Use consistent naming conventions for IDs and names across model versions
4. **Clear descriptions:** Write helpful descriptions that explain model capabilities and use cases
5. **Proper licensing:** Always include complete license information (type, URI, and text)
6. **Test accessibility:** Ensure all URLs are accessible and return the expected content
7. **Execution provider compatibility:** Ensure execution providers match target hardware capabilities
8. **Logical grouping:** Use name field to group related model variants (different EP versions of the same base model)
9. **File organization:** Keep related files together and use descriptive file names
10. **Security:** Use HTTPS for all file downloads and include SHA256 verification

Hosting considerations

When hosting a model catalog:

1. **HTTPS required:** Always serve catalogs and models over HTTPS
2. **CORS headers:** Configure appropriate CORS headers for cross-origin access
3. **Content-Type:** Serve catalog JSON with `application/json` content type
4. **Cache headers:** Set appropriate cache headers for performance
5. **Authentication:** Support standard HTTP authentication if required

JSON Schema

The following is the JSON schema that can be used for validation of your JSON payload.

JSON

```
{  
  "$schema": "https://json-schema.org/draft/2020-12/schema",  
  "title": "WinML Model Catalog Schema",  
  "description": "JSON schema for WindowsML Model catalog configuration",  
  "type": "object",  
  "properties": {  
    "name": {  
      "type": "string",  
      "description": "Name of the catalog"  
    },  
    "models": {  
      "type": "array",  
      "description": "List of models in the catalog",  
      "items": {  
        "type": "object",  
        "properties": {  
          "name": {  
            "type": "string",  
            "description": "Name of the model"  
          },  
          "version": {  
            "type": "string",  
            "description": "Version of the model"  
          },  
          "url": {  
            "type": "string",  
            "description": "URL to download the model"  
          },  
          "sha256": {  
            "type": "string",  
            "description": "SHA256 hash of the model file"  
          }  
        }  
      }  
    }  
  }  
}
```

```
"type": "object",
"required": [ "models" ],
"properties": {
  "models": {
    "type": "array",
    "description": "Array of machine learning models in the catalog",
    "items": {
      "$ref": "#/$defs/Model"
    }
  }
},
"$defs": {
  "Model": {
    "type": "object",
    "required": [ "id", "name", "version", "publisher", "executionProviders",
"license" ],
    "properties": {
      "id": {
        "type": "string",
        "description": "Unique-in-the-catalog identifier for the model"
      },
      "name": {
        "type": "string",
        "description": "Common name of the model"
      },
      "version": {
        "type": "string",
        "description": "Version of the model"
      },
      "publisher": {
        "type": "string",
        "description": "Publisher or organization that created the model"
      },
      "executionProviders": {
        "type": "array",
        "description": "Array of execution providers supported by the model",
        "items": {
          "$ref": "#/$defs/ExecutionProvider"
        }
      },
      "modelSizeBytes": {
        "type": "integer",
        "minimum": 0,
        "description": "Size of the model in bytes"
      },
      "license": {
        "type": "string",
        "description": "License type (e.g., MIT, BSD, Apache)"
      },
      "licenseUri": {
        "type": "string",
        "format": "uri",
        "description": "URI to the license document"
      },
      "licenseText": {
        "type": "string",
        "format": "text"
      }
    }
  }
}
```

```
        "type": "string",
        "description": "Text content of the license"
    },
    "uri": {
        "type": "string",
        "format": "uri",
        "description": "URI where the model can be accessed"
    },
    "files": {
        "type": "array",
        "description": "Array of files associated with the model",
        "items": {
            "$ref": "#/$defs/File"
        }
    },
    "packages": {
        "type": "array",
        "description": "Array of packages required for the model",
        "items": {
            "$ref": "#/$defs/Package"
        }
    }
},
"oneOf": [
{
    "required": [ "files" ],
    "not": { "required": [ "packages" ] }
},
{
    "required": [ "packages" ],
    "not": { "required": [ "files" ] }
}
]
},
"File": {
    "type": "object",
    "required": [ "name", "sha256" ],
    "properties": {
        "name": {
            "type": "string",
            "description": "Name of the file"
        },
        "uri": {
            "type": "string",
            "format": "uri",
            "description": "URI where the file can be downloaded"
        },
        "sha256": {
            "type": "string",
            "pattern": "^[a-fA-F0-9]{64}$",
            "description": "SHA256 hash of the file for integrity verification"
        }
    }
},
"Package": {
```

```
"type": "object",
"required": [ "packageFamilyName", "uri" ],
"properties": {
  "packageFamilyName": {
    "type": "string",
    "description": "Windows package family name identifier"
  },
  "uri": {
    "type": "string",
    "format": "uri",
    "description": "URI where the package can be obtained (HTTPS or local file
path)"
  },
  "sha256": {
    "type": "string",
    "pattern": "^[a-fA-F0-9]{64}$",
    "description": "SHA256 hash of the package for integrity verification"
  }
},
"if": {
  "properties": {
    "uri": {
      "pattern": "^https://"
    }
  }
},
"then": {
  "required": [ "packageFamilyName", "uri", "sha256" ]
}
},
"ExecutionProvider": {
  "type": "object",
  "required": [ "name" ],
  "properties": {
    "name": {
      "type": "string",
      "description": "Name of the execution provider (e.g.,
CPUExecutionProvider)"
    }
  }
}
}
```

Next steps

- Learn about [Model Catalog](#) overview
 - Follow the [Get started guide](#)
 - Explore [Windows ML](#) documentation

Last updated on 01/06/2026

Deploying your app that uses Windows ML

When you're ready to distribute your C# or C++ app that uses Windows ML, you need to ensure that the Windows App SDK framework is properly deployed to your users' devices. The Windows ML runtime is distributed as part of the Windows App SDK.

Supported deployment options for Windows ML

Windows ML supports both the framework-dependent and the self-contained deployment options in Windows App SDK. See the [Windows App SDK deployment overview](#) for more details about the deployment options in Windows App SDK.

Framework-dependent: Supported

Your app depends on the Windows App SDK runtime and/or framework package being present on the target machine. Framework-dependent deployment is the default deployment mode of the Windows App SDK for its efficient use of machine resources and serviceability. See [Deployment architecture and overview for framework-dependent apps](#) for more details.

Self-contained: Supported

With the GA release of Windows ML, use of the self-contained deployment option in Windows App SDK is now supported when using Windows ML. See [Deployment guide for self-contained apps](#) for more details.

In self-contained mode, ONNX Runtime binaries are deployed alongside your application:

```
MyApp/
├── MyApp.exe
├── Microsoft.Windows.AI.MachineLearning.dll
├── onnxruntime.dll
├── onnxruntime_providers_shared.dll
└── DirectML.dll
```

Additional resources

For more detailed information on deploying Windows App SDK applications, refer to these resources:

- [Get started with the Windows App SDK](#)

- [Windows App SDK deployment guide](#)
 - [Windows ML samples ↗](#)
-

Last updated on 09/23/2025

Migrate from standalone ONNX Runtime to Windows ML's ONNX Runtime

This guide explains how to migrate from using the [standalone ONNX Runtime](#) (the Microsoft-maintained, cross-platform runtime available via NuGet or GitHub) to the ONNX Runtime included in Windows ML.

Why switch to Windows ML?

- **Smaller app download / install size** - Your app doesn't need to distribute large EPs and the ONNX Runtime
 - EPs are dynamically downloaded via Windows ML, so that you don't have to bundle them with your app
 - The ONNX Runtime in Windows ML is a shared system-wide copy, so your app doesn't have to bundle it with your app
- **Dynamically uses latest EPs** - Automatically downloads and manages the latest compatible hardware-specific execution providers, without requiring your app to update
- **Dynamically uses latest ONNX Runtime** - Automatically updates the ONNX Runtime without requiring your app to update. See the [ONNX versions](#) docs for more info

System requirements for Windows ML

- **OS:** Version of Windows that [Windows App SDK supports](#)
- **Architecture:** x64 or ARM64
- **Hardware:** Any PC configuration (CPUs, integrated/discrete GPUs, NPUs)

Step 1: Check ONNX version compatibility

See the [ONNX Runtime versions shipped in Windows ML](#) docs to ensure Windows ML has the version of the ONNX Runtime your app requires. Make any updates to your models or code as necessary.

Step 2: Check supported EPs

See the [supported execution providers in Windows ML](#) docs to ensure Windows ML supports the execution providers your app requires. Make any updates to your models or code as necessary.

Step 3: Check Windows App SDK requirements

Windows ML supports both the framework-dependent and the self-contained deployment options in Windows App SDK. See the [Windows App SDK deployment overview](#) for more details about the deployment options in Windows App SDK. Make any updates to your app as necessary.

Step 4: Switch to Windows ML's ONNX Runtime

Remove the copy of the ONNX Runtime your app is currently using.

Then, follow Step 1 of the [get started with Windows ML](#) docs to learn how to install the Windows App SDK (which contains Windows ML).

After installing Windows ML, C# and Python devs should be able to compile their app. The ONNX APIs in Windows ML are identical to the ONNX APIs in standalone ONNX Runtime. See [use ONNX APIs in Windows ML](#) for more info.

For C++ developers, there are two choices...

- Update your usage of the ONNX Runtime headers to use the Windows ML ONNX Runtime Headers, which are included in a `winml/` directory.
- OR, set the `WinMLEnableDefaultOrtHeaderIncludePath` property to true, so that the ONNX Runtime header paths will be the same as the standalone ONNX Runtime you were using before.

See [Use ONNX APIs](#) for more info on both options.

You also can incrementally migrate some of your models to Windows ML, while using your current copy of ONNX Runtime for other models by [running multiple versions of ONNX Runtime in your app](#).

Step 5: Initialize EPs via Windows ML

See the [initialize execution providers with Windows ML](#) docs to learn how to dynamically initialize (download and register) EPs using Windows ML.

Step 6: Run your app!

Your app should now be locally working with Windows ML!

Last updated on 01/09/2026

Windows ML walkthrough

This short tutorial walks through using Windows ML to run the ResNet-50 image classification model on Windows, detailing model acquisition and preprocessing steps. The implementation involves dynamically selecting execution providers for optimized inference performance.

The ResNet-50 model is a PyTorch model intended for image classification.

In this tutorial, you'll acquire the ResNet-50 model from Hugging Face, and convert it to QDQ ONNX format by using the AI Toolkit.

Then you'll load the model, prepare input tensors, and run inference using the Windows ML APIs, including post-processing steps to apply softmax, and retrieve the top predictions.

Acquiring the model, and preprocessing

You can acquire [ResNet-50](#) from Hugging Face (the platform where the ML community collaborates on models, datasets, and apps). You'll convert ResNet-50 to QDQ ONNX format by using the AI Toolkit (see [convert models to ONNX format](#) for more info).

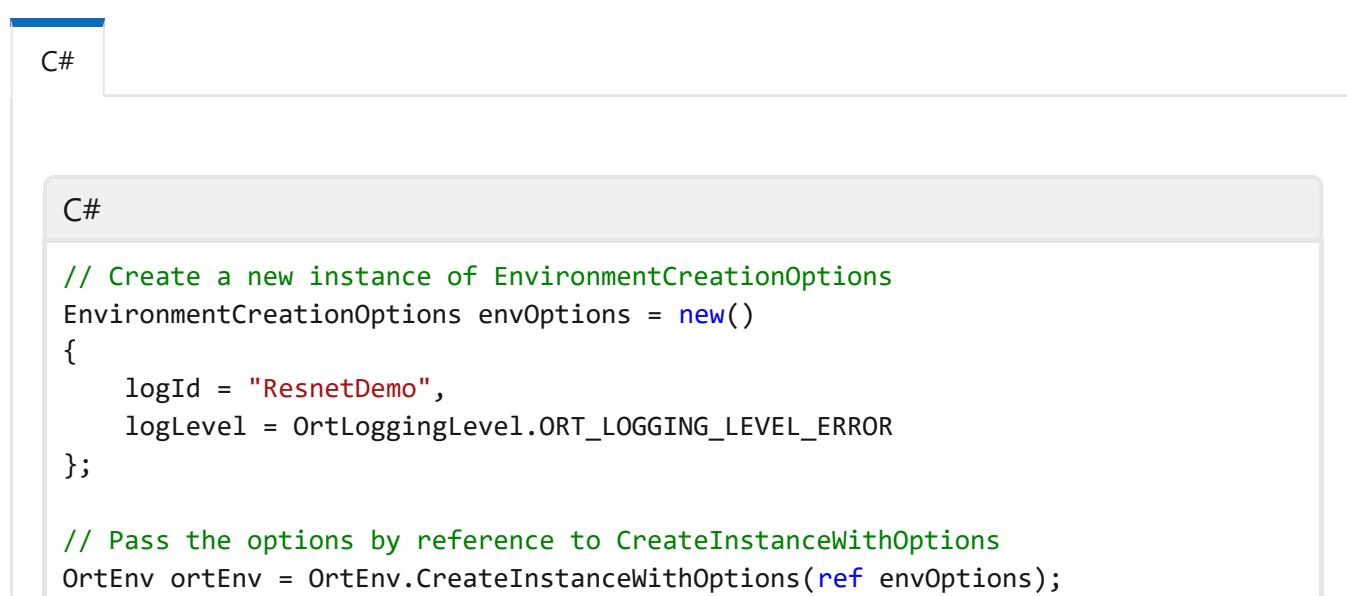
The goal of this example code is to leverage the Windows ML runtime to do the heavy lifting.

The Windows ML runtime will:

- Load the model.
- Dynamically select the preferred IHV-provided execution provider (EP) for the model and download its EP from the Microsoft Store, on demand.
- Run inference on the model using the EP.

For API reference, see [OrtSessionOptions](#) and

[Microsoft::Windows::AI::MachineLearning::ExecutionProviderCatalog class](#).



C#

```
// Create a new instance of EnvironmentCreationOptions
EnvironmentCreationOptions envOptions = new()
{
    logId = "ResnetDemo",
    LogLevel = OrtLoggingLevel.ORT_LOGGING_LEVEL_ERROR
};

// Pass the options by reference to CreateInstanceWithOptions
OrtEnv ortEnv = OrtEnv.CreateInstanceWithOptions(ref envOptions);
```

```

// Use Windows ML to download and register Execution Providers
var catalog =
Microsoft.Windows.AI.MachineLearning.ExecutionProviderCatalog.GetDefault();
Console.WriteLine("Ensuring and registering execution providers...");
await catalog.EnsureAndRegisterCertifiedAsync();

//Create Onnx session
Console.WriteLine("Creating session ...");
var sessionOptions = new SessionOptions();
// Set EP Selection Policy
sessionOptions.SetEpSelectionPolicy(ExecutionProviderDevicePolicy.MIN_OVERALL_POWER);

```

EP compilation

If your model isn't already compiled for the EP (which may vary depending on device), the model first needs to be compiled against that EP. This is a one-time process. The example code below handles it by compiling the model on the first run, and then storing it locally. Subsequent runs of the code pick up the compiled version, and run that; resulting in optimized fast inferences.

For API reference, see [Ort::ModelCompilationOptions struct](#), [Ort::Status struct](#), and [Ort::CompileModel](#).

C#

```

C#
// Prepare paths
string executableFolder =
Path.GetDirectoryName(Assembly.GetEntryAssembly()!.Location)!;
string labelsPath = Path.Combine(executableFolder, "ResNet50Labels.txt");
string imagePath = Path.Combine(executableFolder, "dog.jpg");

// TODO: Please use AITK Model Conversion tool to download and convert Resnet,
// and paste the converted path here
string modelPath = @"";
string compiledModelPath = @"";

// Compile the model if not already compiled
bool isCompiled = File.Exists(compiledModelPath);
if (!isCompiled)
{
    Console.WriteLine("No compiled model found. Compiling model ...");
    using (var compileOptions = new OrtModelCompilationOptions(sessionOptions))
    {

```

```

        compileOptions.SetInputModelPath(modelPath);
        compileOptions.SetOutputModelPath(compiledModelPath);
        compileOptions.CompileModel();
        isCompiled = File.Exists(compiledModelPath);
        if (isCompiled)
        {
            Console.WriteLine("Model compiled successfully!");
        }
        else
        {
            Console.WriteLine("Failed to compile the model. Will use original
model.");
        }
    }
else
{
    Console.WriteLine("Found precompiled model.");
}
var modelPathToUse = isCompiled ? compiledModelPath : modelPath;

```

Running the inference

The input image is converted to tensor data format, and then inference runs on it. While this is typical of all code that uses the ONNX Runtime, the difference in this case is that it's ONNX Runtime directly through Windows ML. The only requirement is adding `#include <winml/onnxruntime_cxx_api.h>` to the code.

Also see [Convert a model with AI Toolkit for VS Code](#)

For API reference, see [Ort::Session struct](#), [Ort::MemoryInfo struct](#), [Ort::Value struct](#), [Ort::AllocatorWithDefaultOptions struct](#), [Ort::RunOptions struct](#).

C#

```

C#
using var session = new InferenceSession(modelPathToUse, sessionOptions);

Console.WriteLine("Preparing input ...");
// Load and preprocess image
var input = await PreprocessImageAsync(await LoadImageFileAsync(imagePath));
// Prepare input tensor
var inputName = session.InputMetadata.First().Key;
var inputTensor = new DenseTensor<float>(
    input.ToArray(),           // Use the DenseTensor<float> directly
    new[] { 1, 3, 224, 224 }, // Shape of the tensor

```

```

        false                                // isReversedStride should be explicitly set to
    false
);

// Bind inputs and run inference
var inputs = new List<NamedOnnxValue>
{
    NamedOnnxValue.CreateFromTensor(inputName, inputTensor)
};

Console.WriteLine("Running inference ...");
var results = session.Run(inputs);
for (int i = 0; i < 40; i++)
{
    results = session.Run(inputs);
}

// Extract output tensor
var outputName = session.OutputMetadata.First().Key;
var resultTensor = results.First(r => r.Name == outputName).AsEnumerable<float>()
    .ToArray();

// Load labels and print results
var labels = LoadLabels(labelsPath);
PrintResults(labels, resultTensor);

```

Post-processing

The softmax function is applied to returned raw output, and label data is used to map and print the names with the five highest probabilities.

C#

```

C#
private static void PrintResults(IList<string> labels, IReadOnlyList<float>
results)
{
    // Apply softmax to the results
    float maxLogit = results.Max();
    var expScores = results.Select(r => MathF.Exp(r - maxLogit)).ToList(); // stability with maxLogit
    float sumExp = expScores.Sum();
    var softmaxResults = expScores.Select(e => e / sumExp).ToList();

    // Get top 5 results
    IEnumerable<(int Index, float Confidence)> topResults = softmaxResults
        .Select((value, index) => (Index: index, Confidence: value))
        .OrderByDescending(x => x.Confidence)

```

```
.Take(5);

// Display results
Console.WriteLine("Top Predictions:");
Console.WriteLine("-----");
Console.WriteLine("{0,-32} {1,10}", "Label", "Confidence");
Console.WriteLine("-----");

foreach (var result in topResults)
{
    Console.WriteLine("{0,-32} {1,10:P2}", labels[result.Index],
result.Confidence);
}

Console.WriteLine("-----");
}
```

Output

Here's an example of the kind of output to be expected.

Console

```
285, Egyptian cat with confidence of 0.904274
281, tabby with confidence of 0.0620204
282, tiger cat with confidence of 0.0223081
287, lynx with confidence of 0.00119624
761, remote control with confidence of 0.000487919
```

Full code samples

The full code samples are available in the WindowsAppSDK-Samples GitHub repository. See [WindowsML](#).

Last updated on 07/08/2025

Windows ML samples

Windows ML supports a wide variety of ONNX models for different AI scenarios. This page provides samples for using Windows ML from different frameworks and languages, as well as ready-to-use model samples for common AI tasks.

AI Dev Gallery

The [AI Dev Gallery](#) is the easiest way to explore Windows ML samples. It provides:

- **Interactive demos** for each model scenario
- **Full source code** for every sample
- **One-click export** to a Visual Studio project so you can use the model in your own app

💡 Tip

Each model sample in the AI Dev Gallery includes complete source code. Use the [Export to Visual Studio](#) feature to quickly create a working project with your chosen model.

Framework integration samples

The [Windows App SDK Samples repository on GitHub](#) contains sample applications that demonstrate how to integrate Windows ML into different languages and frameworks.

 Expand table

Language	Framework	Packaging type	Dependency type	Link
C#	Console	Unpackaged	Framework-dependent	Link ↗
C#	WPF	Unpackaged	Framework-dependent	Link ↗
C#	WinUI	MSIX	Framework-dependent	Link ↗
C#	WinForms	Unpackaged	Framework-dependent	Link ↗
C++	Console	DLL w/ separate EXE	Framework-dependent	Link ↗
C++	Console	Unpackaged	Self-contained	Link ↗
C++	Console	MSIX	Framework-dependent	Link ↗
Python	Console	Unpackaged	Framework-dependent	Link ↗

Model samples by scenario

The following sections list available models for each AI scenario, along with their size and hardware support. Click "Try it yourself" to open the sample in the AI Dev Gallery app.

Large Language Models (LLMs) - Text generation

Generate conversational text responses using state-of-the-art language models.

[Try it yourself in the AI Dev Gallery app ↗](#)

 [Expand table](#)

Model	Hardware	Size
Phi 4 Mini CPU	CPU	4.6 GB
Phi 3.5 Mini CPU ACC4	CPU	2.6 GB
Phi 3 Mini CPU	CPU	2.5 GB
Phi 3 Mini CPU ACC4	CPU	2.5 GB
Phi 3 Medium CPU ACC4	CPU	8.6 GB
Mistral 7B Instruct 0.2 CPU	CPU	4.6 GB
Mistral 7B Instruct 0.2 CPU ACC4	CPU	4.6 GB

Large Language Models (LLMs) - Multi-modal

Analyze images and generate text descriptions using vision-enabled language models.

[Try it yourself in the AI Dev Gallery app ↗](#)

 [Expand table](#)

Model	Hardware	Size
Phi 3 Vision CPU	CPU	3.0 GB
Phi 3.5 Vision CPU	CPU	3.0 GB

Text embedding (Semantic search)

Convert text into vector embeddings for semantic search and similarity matching.

[Try it yourself in the AI Dev Gallery app ↗](#)

[Expand table](#)

Model	Hardware	Size
all-MiniLM-L12-v2	CPU, GPU	127.2 MB
all-MiniLM-L6-v2	CPU, GPU	86.4 MB

Image generation

Create images from text prompts using diffusion models.

[Try it yourself in the AI Dev Gallery app ↗](#)

[Expand table](#)

Model	Hardware	Size
Stable Diffusion v1.4	CPU, GPU	5.1 GB

Image classification

Classify images into predefined categories.

[Try it yourself in the AI Dev Gallery app ↗](#)

[Expand table](#)

Model	Hardware	Size
MobileNet v2 1.0	CPU, GPU	13.3 MB
ResNet101 v1 7	CPU, GPU	170.6 MB
ResNet50 v1 7	CPU, GPU	97.8 MB
SqueezeNet 1.1	CPU, GPU, NPU	4.7 MB

Image object detection

Detect and locate objects within images.

[Try it yourself in the AI Dev Gallery app ↗](#)

[Expand table](#)

Model	Hardware	Size
Faster RCNN 10	CPU, GPU	159.6 MB
Faster RCNN 12	CPU, GPU	168.5 MB
YOLOv4	CPU, GPU	245.5 MB

Image segmentation (Background detection)

Separate foreground subjects from backgrounds in images.

[Try it yourself in the AI Dev Gallery app ↗](#)

[Expand table](#)

Model	Hardware	Size
SINet	CPU, GPU, NPU	404.7 KB

Human pose detection

Detect human body poses and key points in images.

[Try it yourself in the AI Dev Gallery app ↗](#)

[Expand table](#)

Model	Hardware	Size
HRNet Pose	CPU, GPU, NPU	108.9 MB

Street segmentation

Segment street photographs into detectable zones for autonomous driving and mapping applications.

[Try it yourself in the AI Dev Gallery app ↗](#)

[Expand table](#)

Model	Hardware	Size
FFNet 78s	CPU, GPU, NPU	104.9 MB
FFNet 54s	CPU, GPU, NPU	68.8 MB

Face detection

Detect and locate human faces in images.

[Try it yourself in the AI Dev Gallery app ↗](#)

 [Expand table](#)

Model	Hardware	Size
FaceDetLite	CPU, GPU, NPU	3.4 MB

Audio transcription (Voice-to-text)

Convert spoken audio into written text using speech recognition models.

[Try it yourself in the AI Dev Gallery app ↗](#)

 [Expand table](#)

Model	Hardware	Size
Whisper Tiny CPU	CPU	73.8 MB
Whisper Small CPU	CPU	422.5 MB
Whisper Medium CPU	CPU	1.3 GB

See also

- [AI Dev Gallery](#)
- [Get started with Windows ML](#)

Windows ML APIs

For conceptual guidance, see [Run ONNX models with Windows ML](#).

You can think of the APIs in the *Microsoft.WindowsAppSDK.ML* NuGet package as being the superset of these two sets:

- *Windows ML APIs*. Windows ML APIs in the [Microsoft.Windows.AI.MachineLearning](#) namespace, such as the [ExecutionProviderCatalog](#) class and its methods (which are Windows Runtime APIs).
- *ONNX Runtime APIs*. Windows ML implementations (in the *Microsoft.WindowsAppSDK.ML* NuGet package) of certain APIs from the ONNX Runtime (ORT). For documentation, see the [ONNX Runtime API docs](#). For example, the [OrtCompileApi struct](#). For code examples that use these APIs, and more links to documentation, see the [Use Windows ML to run the ResNet-50 model](#) tutorial.

The [Microsoft.WindowsAppSDK.ML](#) NuGet package

The Microsoft Windows ML runtime provides APIs for machine learning and AI operations in Windows applications. The *Microsoft.WindowsAppSDK.ML* NuGet package provides the Windows ML runtime `.winmd` files for use in both C# and C++ projects.

The [pywinrt](#) Python wheels

The Microsoft Windows ML runtime leverages the [pywinrt](#) project to provide Python access to the same Windows ML APIs. The package name is *winui3-Microsoft.Windows.AI.MachineLearning*. Additional packages are required to use Windows App SDK in python. For details, see the [Run ONNX models with Windows ML](#) topic.

Windows ML APIs

For API reference documentation, and code examples, see the [Microsoft.Windows.AI.MachineLearning](#) namespace.

Implementation notes

The Windows ML runtime is integrated with the Windows App SDK and relies on its deployment and bootstrapping mechanisms:

- Automatically discovers execution providers compatible with current hardware
- Manages package lifetime and updates
- Handles package registration and activation
- Supports different versions of execution providers

Framework-dependent deployment

Windows ML is delivered as a *framework-dependent* component. This means your app must either:

- Reference the main Windows App SDK NuGet package by adding a reference to `Microsoft.WindowsAppSDK` (recommended)
- Or, reference both `Microsoft.WindowsAppSDK.ML` and `Microsoft.WindowsAppSDK.Runtime`

For more information on deploying Windows App SDK applications, see the [Package and deploy Windows apps](#) documentation.

Using ONNX Runtime with Windows ML

For C++ applications, after registering execution providers, use the ONNX Runtime C API directly to create sessions and run inference.

For C# applications, use the ONNX Runtime directly for inference using the `Microsoft.ML.OnnxRuntime` namespace.

For Python applications, use the separate ONNX Runtime wheel (`onnxruntime`) for inference. For the experimental release, please use the `onnxruntime-winml==1.22.0.post2` package from index https://aiinfra.pkgs.visualstudio.com/PublicPackages/_packaging/ORT-Nightly/pypi/simple.

Python notes

Initialize Windows App SDK

All Windows ML calls should happen after the Windows App SDK is initialized. This can be done with the following code:

Python

```
from winui3.microsoft.windows.applicationmodel.dynamicdependency.bootstrap import (
    InitializeOptions,
    initialize
)
```

```
with initialize(options = InitializeOptions.ON_NO_MATCH_SHOW_UI):  
    # Your Windows ML code here
```

Registration happens out of Windows ML

The ONNX runtime is designed in a way where the Python and native environments are separate. And native registration calls in the same process will not work for the Python environment. Thus, the registration of execution providers should be done with the Python API directly.

Remove pywinrt's packed vcruntime

The pywinrt project includes a msdp140.dll in the winrt-runtime package. This may conflict with other packages. Please remove this dll to avoid this problem and install the missing vcruntime libraries with the [vc redistributable](#)

See also

- [Run ONNX models with Windows ML](#)
- [Windows App SDK Documentation](#)
- [Windows App SDK on GitHub ↗](#)
- [Windows ML Samples ↗](#)
- [ONNX Runtime API Documentation ↗](#)
- [Package and deploy Windows apps](#)

Last updated on 09/26/2025

What is the AI Dev Gallery?

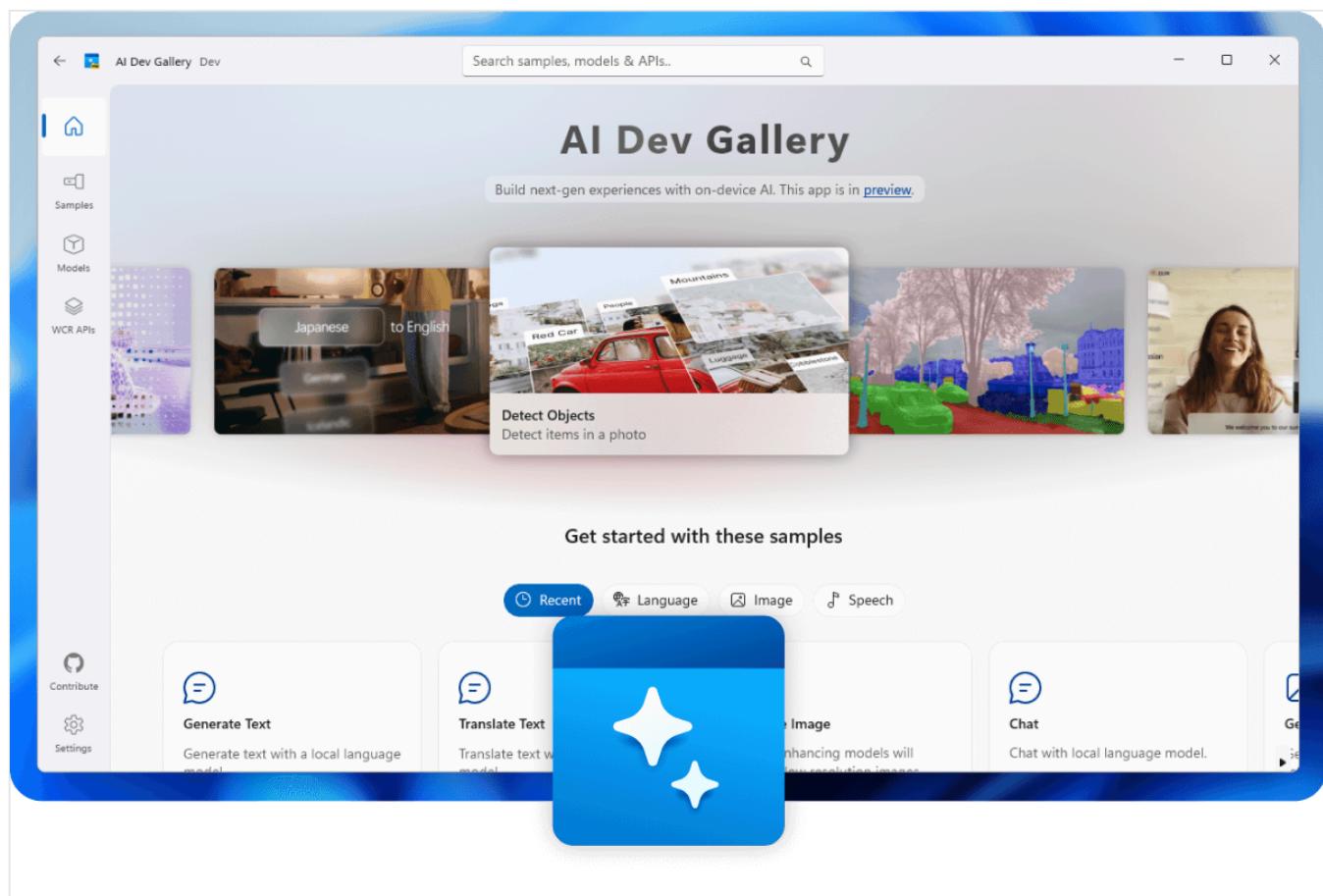
AI Dev Gallery is an [open-source app](#) for Windows developers aiming to integrate AI capabilities into their apps and projects.

The app contains:

- Over 25 interactive samples powered by local AI models, including samples for all the Windows AI APIs.
- A user-friendly interface to explore, download, and run models from Hugging Face and GitHub that can leverage your PC's NPU, CPU, or GPU depending on your device capabilities.
- The capability to view the C# source code and seamlessly export each sample to a standalone Visual Studio project.

[Install AI Dev Gallery](#)

Visit the [AI Dev Gallery repo on GitHub](#) for instructions on how to install without using the Microsoft Store.



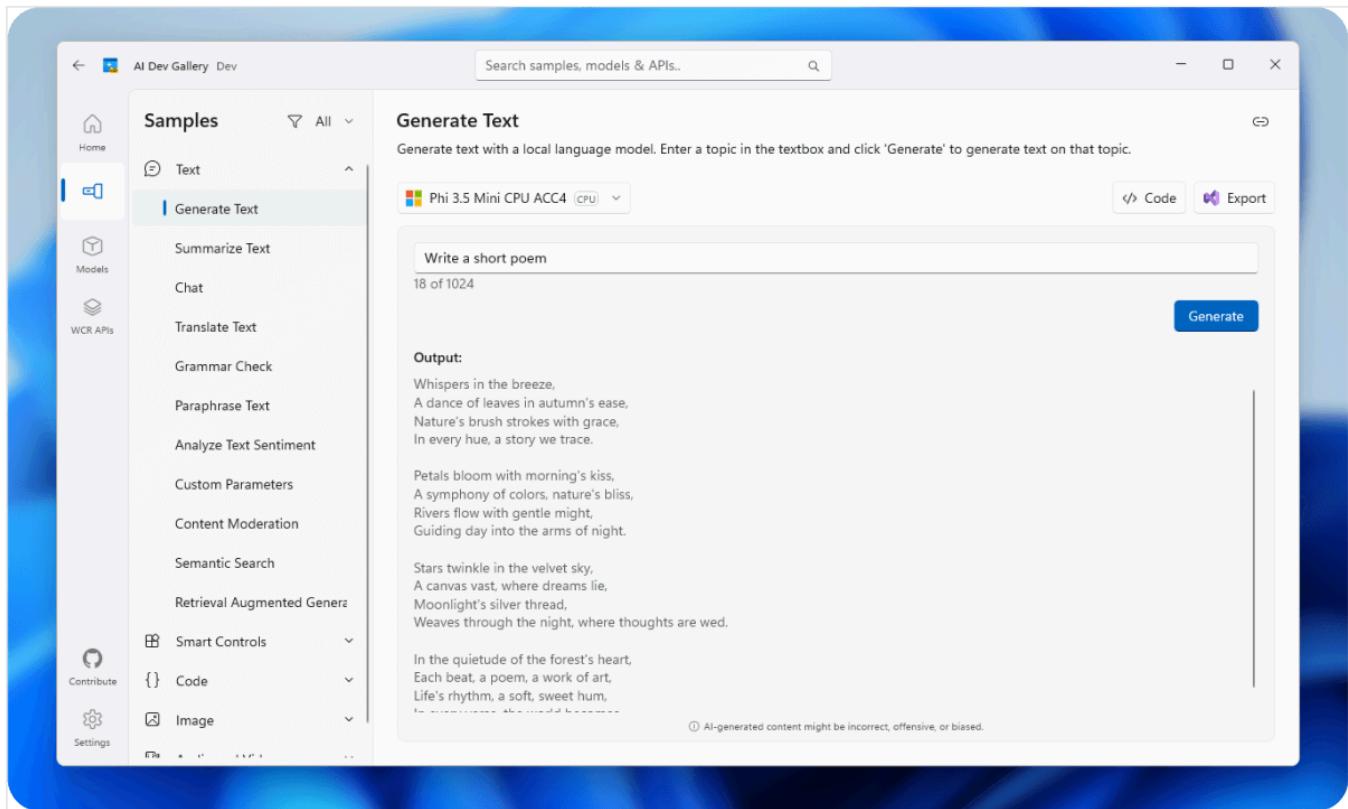
Prerequisites

- Windows 10 version 10.0.17763.0 or later

- [Visual Studio 2022 or later](#)
- [Windows Application Development workload](#) in Visual Studio
- Some AI models might require a minimum amount of dedicated GPU memory to function properly

Interact with samples powered by local AI models

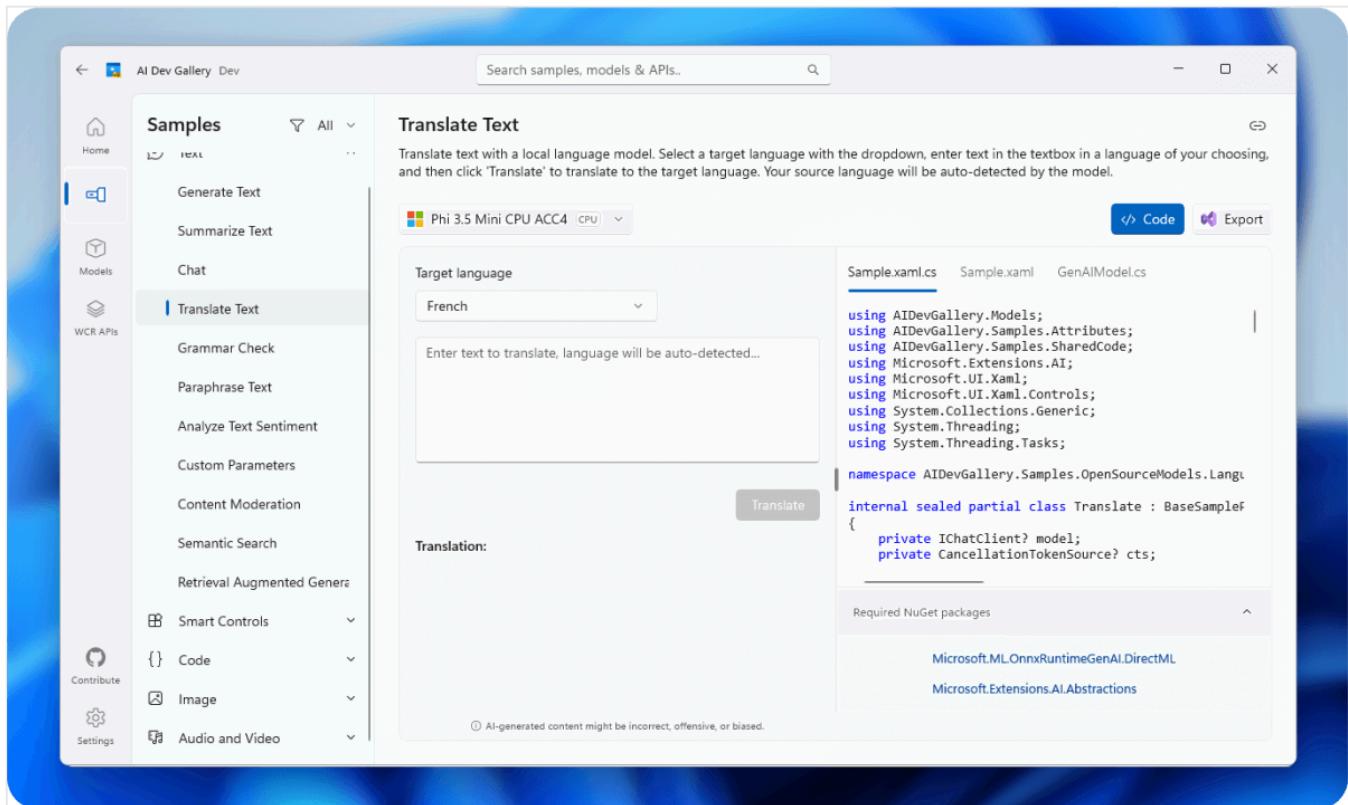
AI Dev Gallery includes a selection of interactive AI-powered samples that demonstrate a wide range of functionalities. These samples cover diverse use cases, from natural language processing to image recognition, enabling developers to explore AI's potential in a hands-on, offline environment.



View and export sample source code

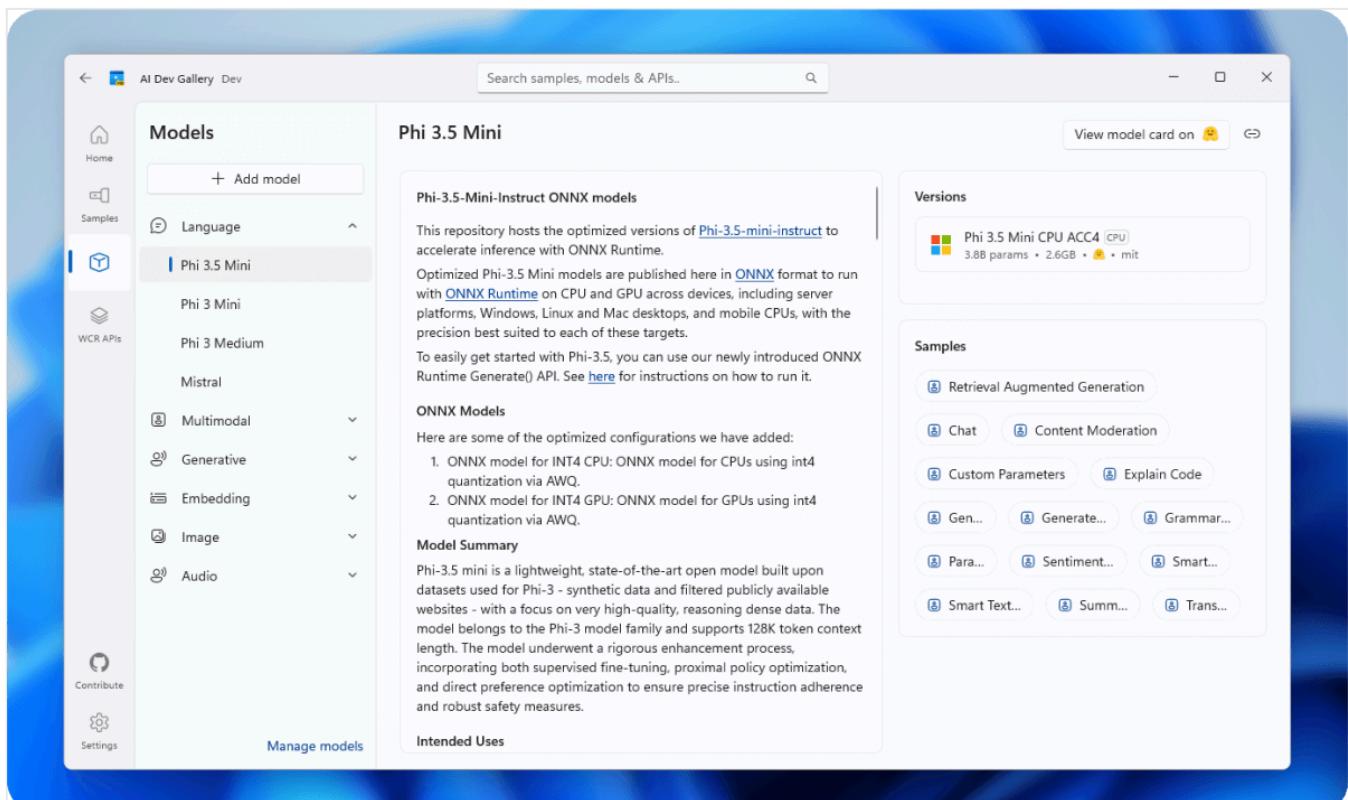
AI Dev Gallery allows developers to access the source code for every sample, making it quick to implement similar AI scenarios in your own project.

The app also enables exporting this code into a new standalone Visual Studio solution, containing only the specific code and model files required for the selected sample.



Experiment with different AI models

Developers can select from a curated list of AI models or download new ones from Hugging Face. A dedicated model section enables you to learn more about the capabilities of your favorite models.



Responsible AI

It is of utmost importance to adopt responsible AI practices when integrating AI into your app. The AI Dev Gallery enables users to access a variety of AI models from sources such as Hugging Face, but we cannot guarantee that these models comply with [Microsoft's Responsible AI standards](#).

Users of the AI Dev Gallery are strongly encouraged to take personal accountability for adopting Responsible AI principles, comprehend the limitations and ethical implications of the models, and ensure adherence to applicable licenses when integrating models into your own app.

Learn more: [Developing Responsible Generative AI Applications and Features on Windows](#)

Frequently Asked Questions

Is a Microsoft account necessary to use the app?

- No, the app does not require a Microsoft account for use.

Can I use the app without an internet connection?

- Yes, the app works offline since the AI models are downloaded locally. However, you will need to be online to download additional AI models from Hugging Face or GitHub.

What AI models are available in the app?

- The app features popular Hugging Face models and also includes models and APIs from Microsoft Foundry on Windows. When executing a sample, you can select which model you want to use. Before downloading or using an open-source model, we strongly recommend you read its model card to better understand it.

Is the app's source code accessible? Can I contribute new samples?

- Yes, the app is completely open-source, and its code is accessible on GitHub

Where can I provide feedback?

- Provide feedback or file an issue on the [AI Dev Gallery GitHub repository](#).

See also

- [AI on Windows Sample Gallery](#)
- [Windows AI API samples](#)

- [Frequently Asked Questions about using AI in Windows apps](#)
 - [Azure AI Content Safety ↗](#)
-

Last updated on 11/18/2025

Tutorial: Use any ONNX LLM in the AI Dev Gallery

09/25/2025

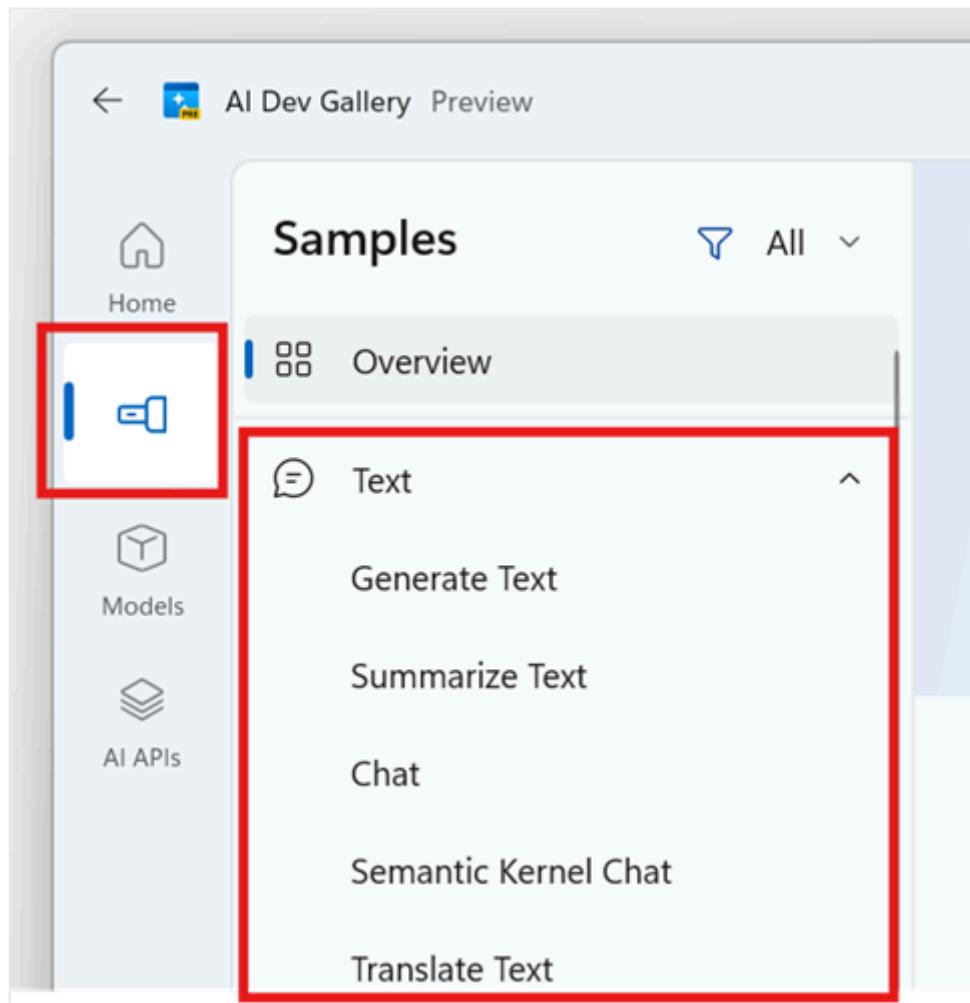
In this tutorial, you'll learn how to run any **ONNX-based large language model (LLM)** through the **AI Dev Gallery**—an open-source Windows application (available in the [Microsoft Store](#)) that showcases AI-powered samples.

These steps work for any LLM in the [ONNX Runtime GenAI format](#), including:

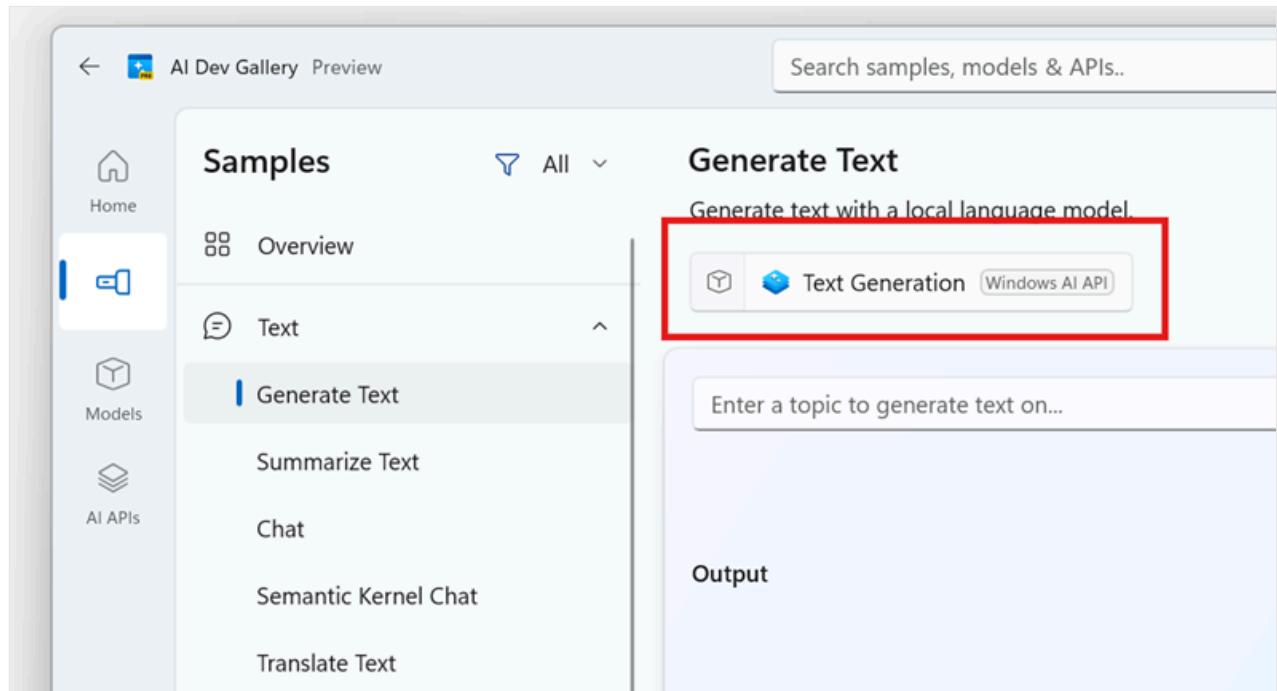
- Models downloaded from [Hugging Face](#)
 - Models converted from other frameworks using the [AI Toolkit for Visual Studio Code](#) conversion tool
-

Step 1: Select an interactive sample in the AI Dev Gallery

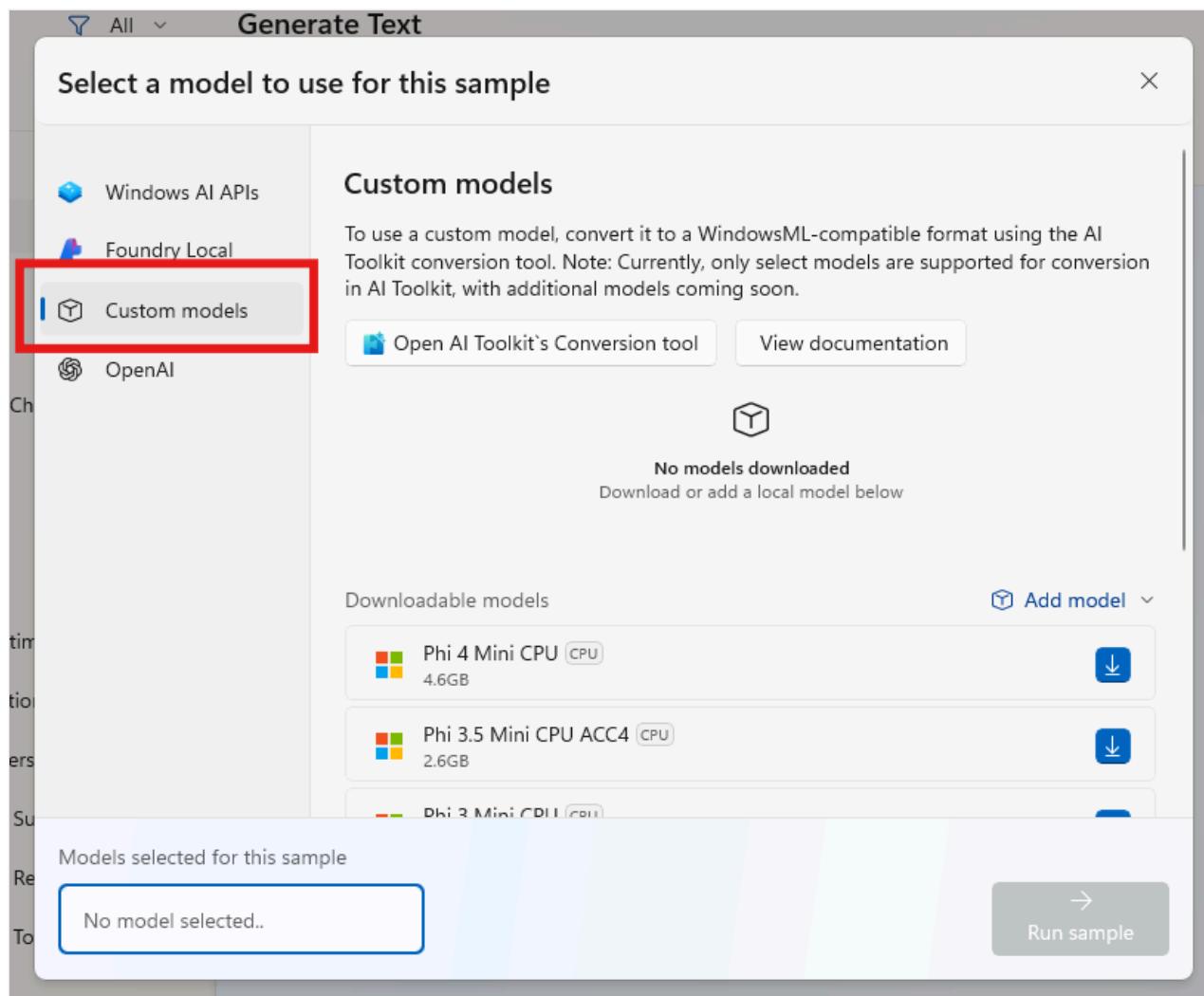
1. Open the **AI Dev Gallery** app.
2. Navigate to the **Samples** tab and choose a **Text** sample (for example, "Generate Text" or "Chat").



3. Click the Model Selector button to view available models for that sample.



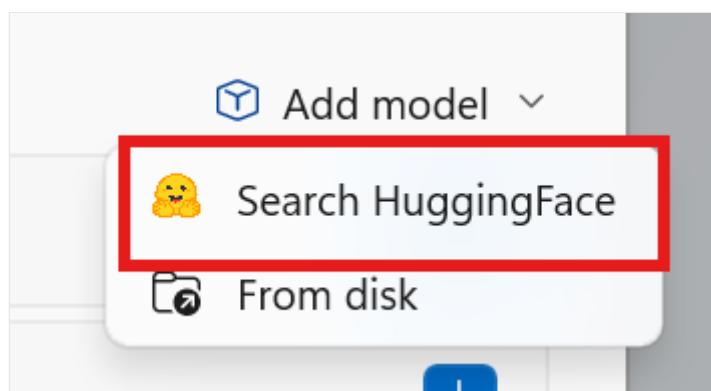
4. Select the Custom models tab to bring in your own ONNX LLM.



Step 2: Get or convert an ONNX LLM model

To use a model in the AI Dev Gallery, it must be in the **ONNX Runtime GenAI** format. You can:

- **Download a pre-converted model:**
 - Browse models on [Hugging Face ONNX Models](#), or
 - In AI Dev Gallery, go to **Add model** → **Search HuggingFace**



- Convert your own model:
 - Click **Open AI Toolkit's Conversion Tool** in the model selector, which launches the **AI Toolkit extension** in Visual Studio Code.
 - If you don't have it installed, search for "AI Toolkit" in the VS Code Extensions Marketplace.
 - Use the [AI Toolkit for Visual Studio Code](#) to convert a supported model to ONNX Runtime GenAI format.

Currently supported models for conversion:

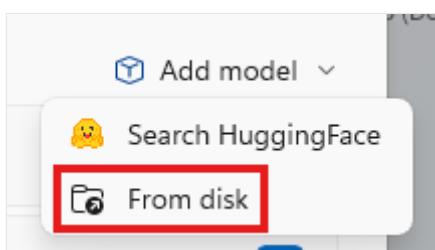
- DeepSeek R1 Distill Qwen 1.5B
- Phi 3.5 Mini Instruct
- Qwen 2.5-1.5B Instruct
- Llama 3.2 1B Instruct

! Note

AI Toolkit conversion is in Preview and currently supports only the models listed above.

Step 3: Use the ONNX model in the AI Dev Gallery

1. Once you have a model in the **ONNX Runtime GenAI format**, return to the AI Dev Gallery **model selector window**.
2. Click **Add model** → **From Disk** and provide the path to your ONNX model.

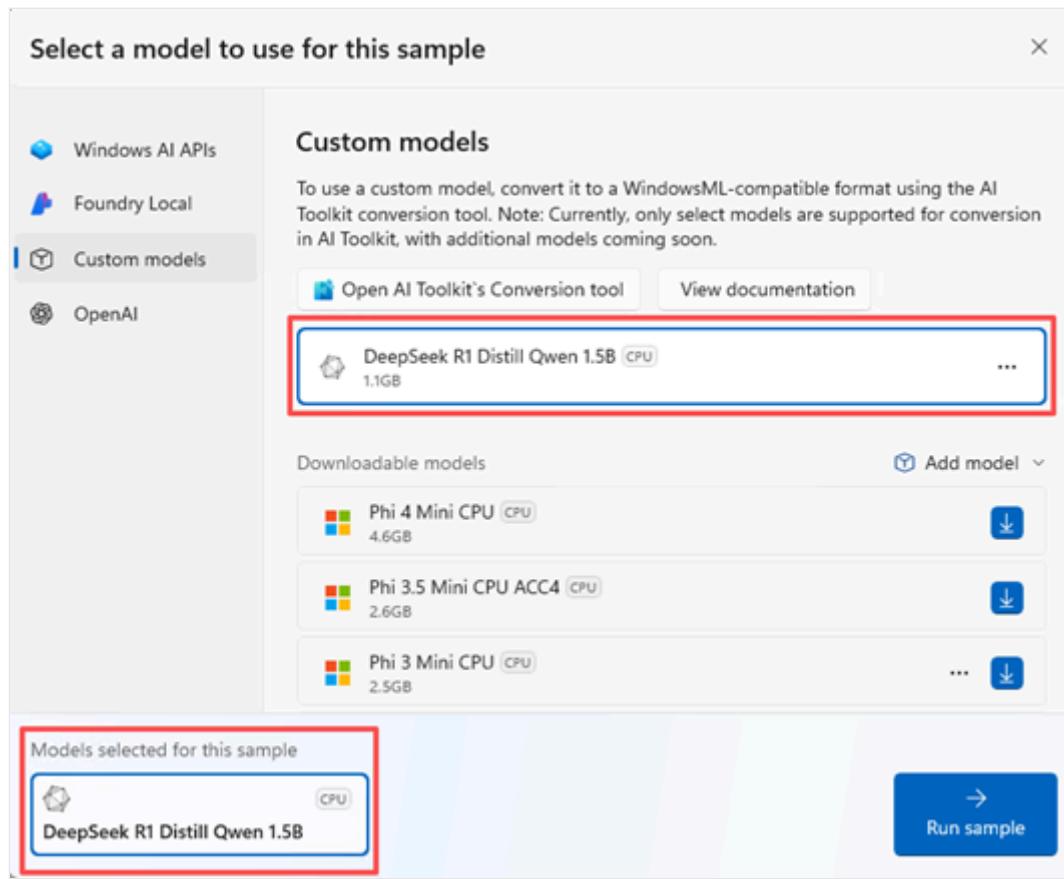


! Note

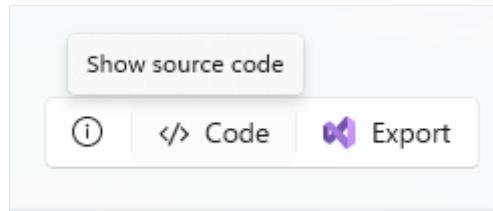
If you used AI Toolkit's conversion tool, the converted model path should follow this format:

```
c:/{{workspace}}/{{model_project}}/history/{{workflow}}/model/model.onnx
```

3. Once added, you can now select your model and use it with the interactive samples.



4. Optionally, click **Show source code** in the app to view the code that runs the model.



Supported samples in the AI Dev Gallery

These ONNX LLMs can be used with the following samples in the AI Dev Gallery:

Text

- Generate Text
- Summarize Text
- Chat
- Semantic Kernel Chat
- Grammar Check
- Paraphrase Text
- Analyze Text Sentiment
- Content Moderation

- Custom Parameters
- Retrieval Augmented Generation

Smart Controls

- Smart TextBox

Code

- Generate Code
 - Explain Code
-

Next steps

Now that you've tried your ONNX LLM in the AI Dev Gallery, you can bring the same approach into your own app.

How the sample works

- The AI Dev Gallery samples use `OnnxRuntimeGenAIChatClient` (from the [ONNX Runtime GenAI SDK](#)) to wrap your ONNX model.
- This client plugs into the `Microsoft.Extensions.AI` abstractions (`IChatClient`, `chatMessage`, etc.), so you can work with prompts and responses in a natural, high-level way.
- Inside the factory (`OnnxRuntimeGenAIChatClientFactory`), the app ensures [Windows ML \(WinML\)](#) execution providers are registered and runs the ONNX model with the best available hardware acceleration (CPU, GPU, or NPU).

Example from the sample:

```
C#  
  
// Register WinML execution providers (under the hood)  
var catalog =  
    Microsoft.Windows.AI.MachineLearning.ExecutionProviderCatalog.GetDefault();  
await catalog.EnsureAndRegisterCertifiedAsync();  
  
// Create a chat client for your ONNX model  
chatClient = await OnnxRuntimeGenAIChatClientFactory.CreateAsync(  
    @"C:\path\to\your\onnx\model",  
    new LlmPromptTemplate  
    {
```

```
System = "<|system|>\n{CONTENT}<|end|>\n",
User = "<|user|>\n{CONTENT}<|end|>\n",
Assistant = "<|assistant|>\n{CONTENT}<|end|>\n",
Stop = [ "<|system|>", "<|user|>", "<|assistant|>", "<|end|>" ]
});

// Stream responses into your UI
await foreach (var part in chatClient.GetStreamingResponseAsync(messages, null,
cts.Token))
{
    OutputTextBlock.Text += part;
}
```

For more details on integrating ONNX models into Windows applications, see:

- [Windows ML documentation](#)
 - [ONNX Runtime GenAI GitHub](#)
-

See also

- [Download AI Dev Gallery](#)
 - [ONNX models on Hugging Face](#)
 - [AI Toolkit for Visual Studio Code](#)
 - [AI Dev Gallery GitHub Repo](#)
-

AI on Windows code samples and tutorials

A collection of samples that demonstrate a variety of ways to enhance your Windows apps using local APIs and Machine Learning (ML) models, local hardware acceleration using DirectML, and cloud-based APIs.

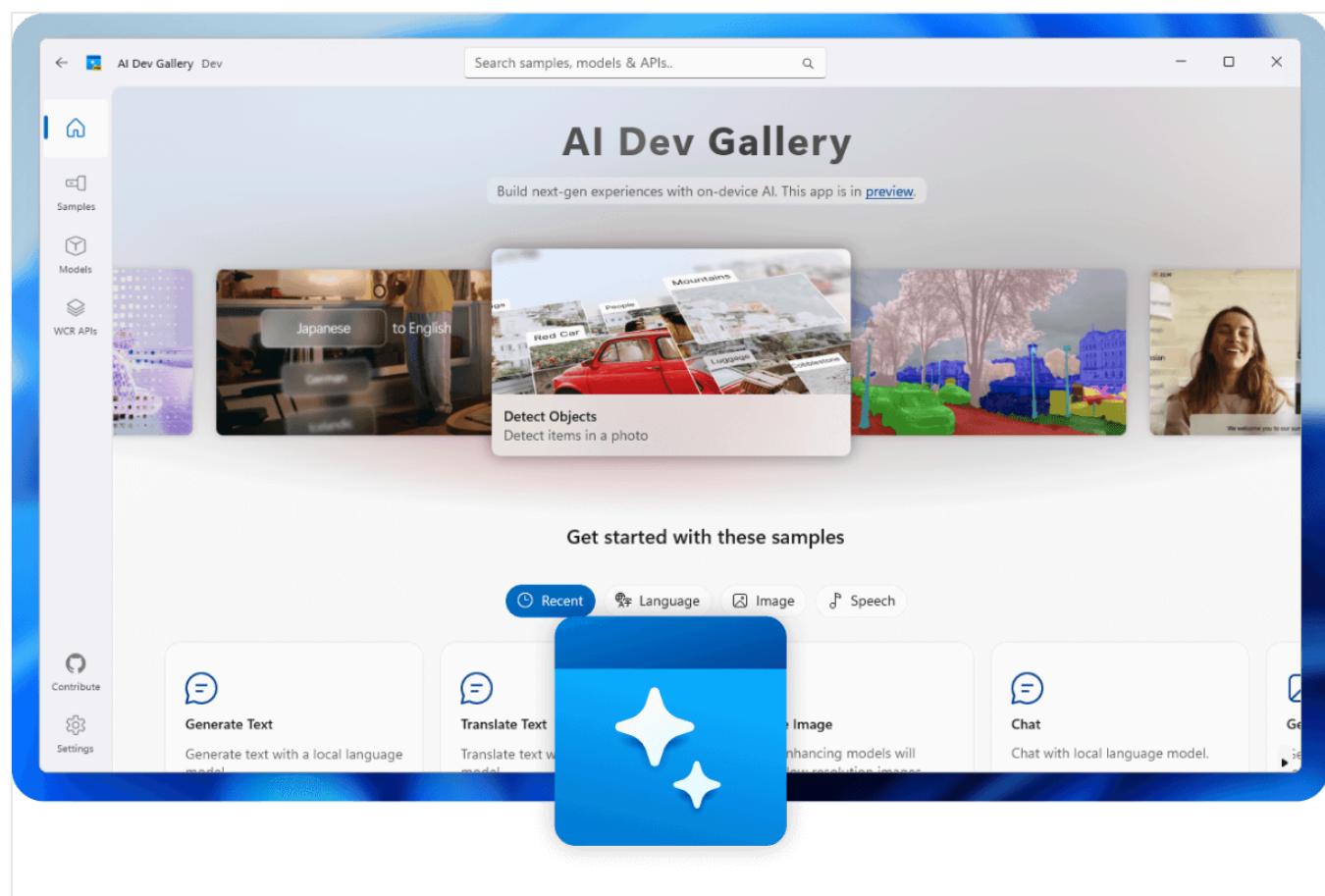
When adding support for new AI features to your Windows app, we recommend that you first [check for model availability](#).

When using AI features, we recommend that you review: [Developing Responsible Generative AI Applications and Features on Windows](#). Text Content Moderation is enabled across all Windows AI APIs to minimize potentially harmful content. Learn more: [Content Safety Moderation with Microsoft Foundry on Windows](#).

Enhance your Windows apps with AI using local APIs and ML models

These samples show how to enhance your Windows apps with AI using local APIs and Machine Learning models.

AI Dev Gallery



GitHub Repo: [AI Dev Gallery](#)

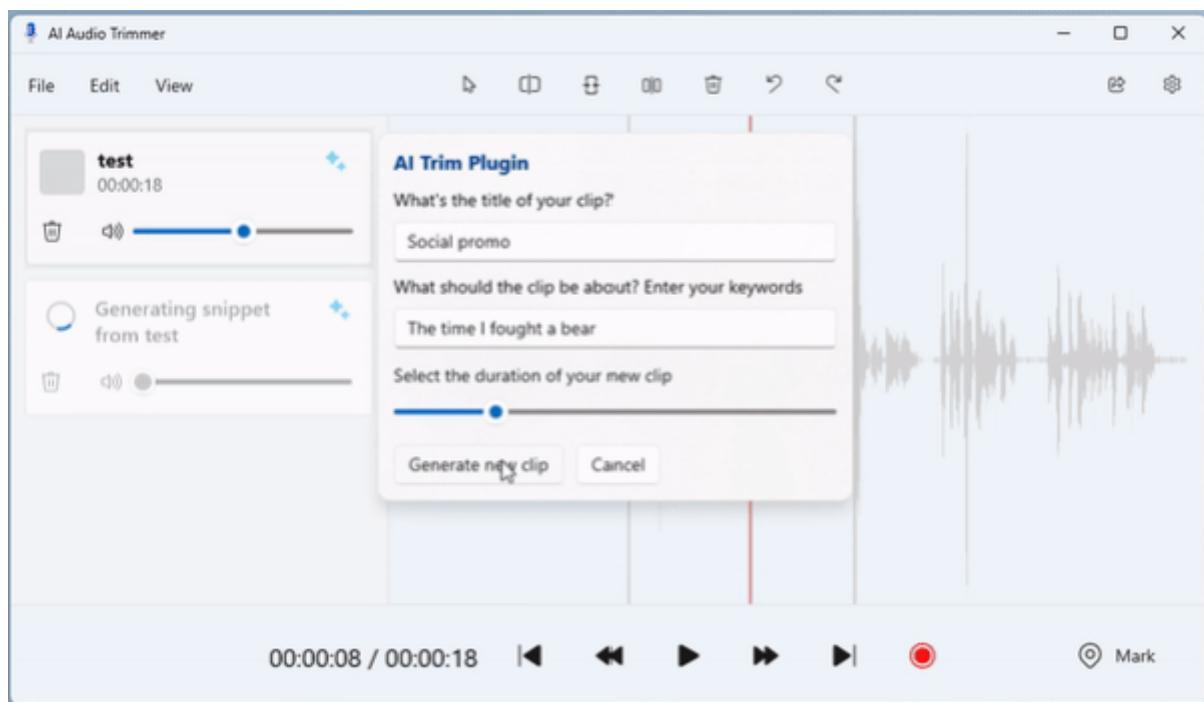
Description: AI Dev Gallery is an open-source app designed for Windows developers looking to integrate AI capabilities within their own apps and projects. It offers over 25 interactive samples powered by local AI models, including samples for all the Windows AI APIs. The app features a simple interface to explore, download, and run models from Hugging Face and GitHub, leveraging your PC's NPU, CPU, or GPU based on your device's capabilities. Additionally, it provides the ability to view the C# source code and export each sample to a standalone Visual Studio project.

Features: Interactive samples with easy-to-copy code, Local Model Inferencing, Showcase of the Windows AI APIs

App Type: [C#](#), WinUI 3

[Install AI Dev Gallery](#)

AI-powered Audio Editor



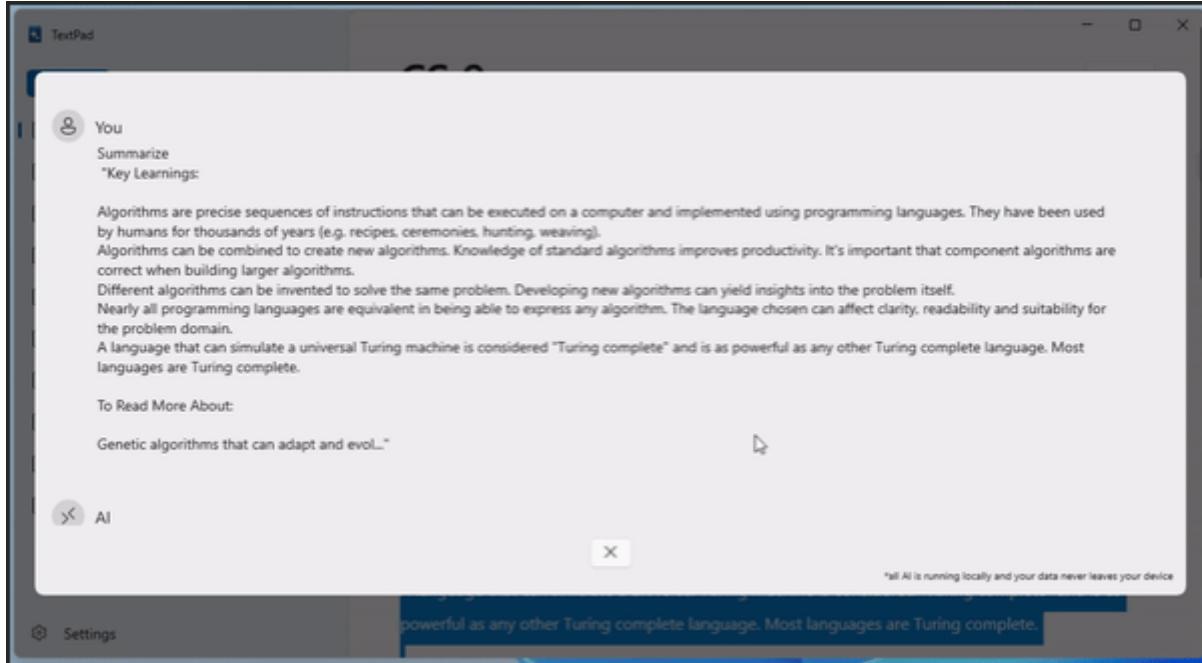
GitHub Repo: [AI Audio Editor Sample](#)

Description: The AI-powered Audio Editor demonstrates building a WinUI 3 audio editing app which utilizes AI to match snips of audio to a relevant query. An example use-case could be a podcast creator who wants to create short audio clips of their content to promote on Social Media. The sample uses local ML model inference to handle transcription and semantic search.

Features: Local Model Inferencing with ONNX Runtime, Whisper model, Embeddings model

App Type: C#, WinUI 3

AI-powered Notes App



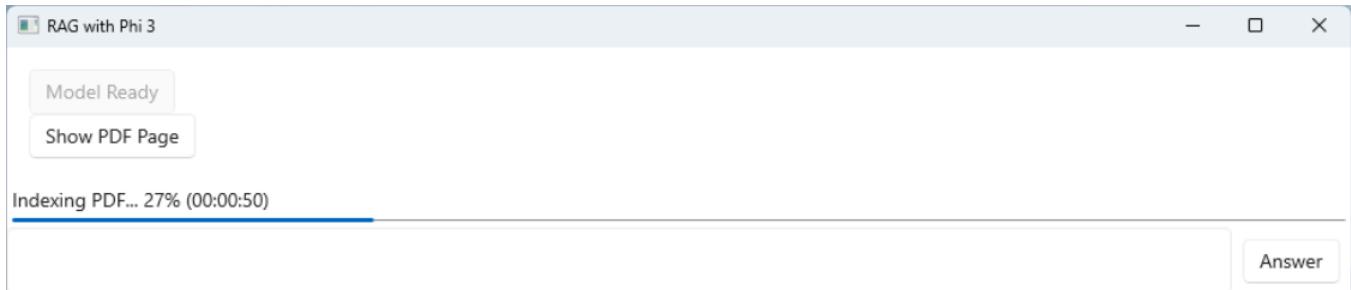
GitHub Repo: [AI-powered Notes Sample App](#)

Description: This AI-powered note taking application demonstrates the use of APIs including [OCR Text Recognition](#), Audio Transcription through local ML model, Semantic Search through a local embeddings model, local language model usage with Phi3 for summarization, autocomplete, and text reasoning, and Retrieval Augmented Generation (RAG) for grounding language models to real data.

Features: Semantic search with local model, Audio transcription with local model, Local Retrieval Augmented generation (RAG) with [Phi3](#), Local Text summarization and reasoning with Phi3, Text extraction from images with [OCR API](#)

App Type: C#, WinUI 3

Retrieval Augmented Generation (RAG) with PDFs and Phi3



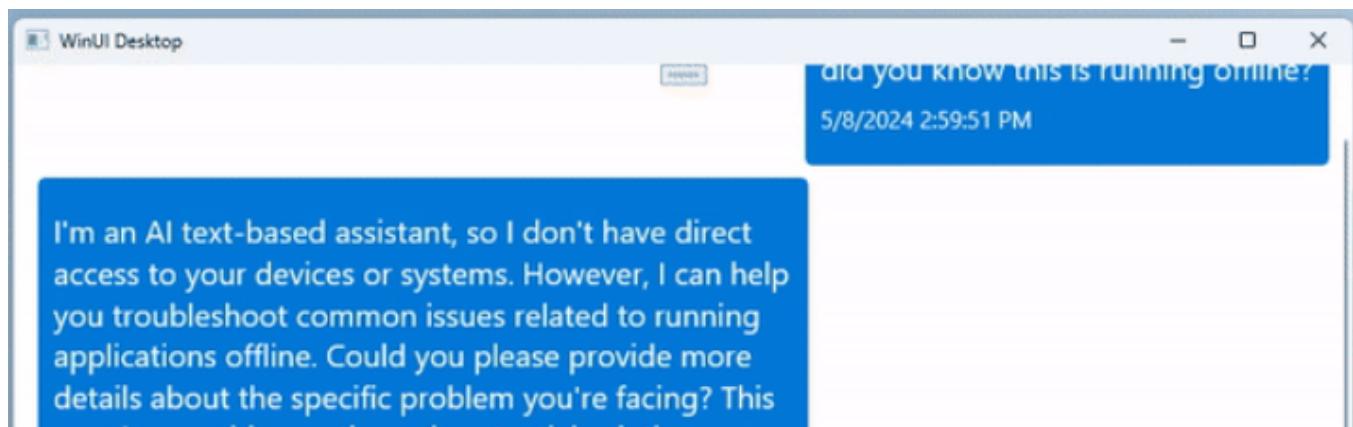
GitHub Repo: [RAG PDF Analyzer WPF Sample App](#)

Description: This WPF sample app demonstrates how to build an experience with a local language model (such as Phi3) to answer questions about content in a PDF document. The sample finds answers by referencing a knowledge base outside of the model's own training data before generating a response. This pattern, called Retrieval Augmented Generation (RAG), is an example of how to ground a language model to real-world authoritative data.

Features: Retrieval Augmented Generation (RAG), ONNX Runtime Generative AI, DirectML

App Type: [C#](#), [WPF](#)

Phi3 Generative AI Chat



GitHub Repo: [Phi3 Chat WinUI 3 Sample](#)

Description: This WinUI 3 app sample demonstrates how to use the ONNX Runtime Generative AI library to build a chat experience with a local language model, specifically the Phi3 Small Language Model (SLM).

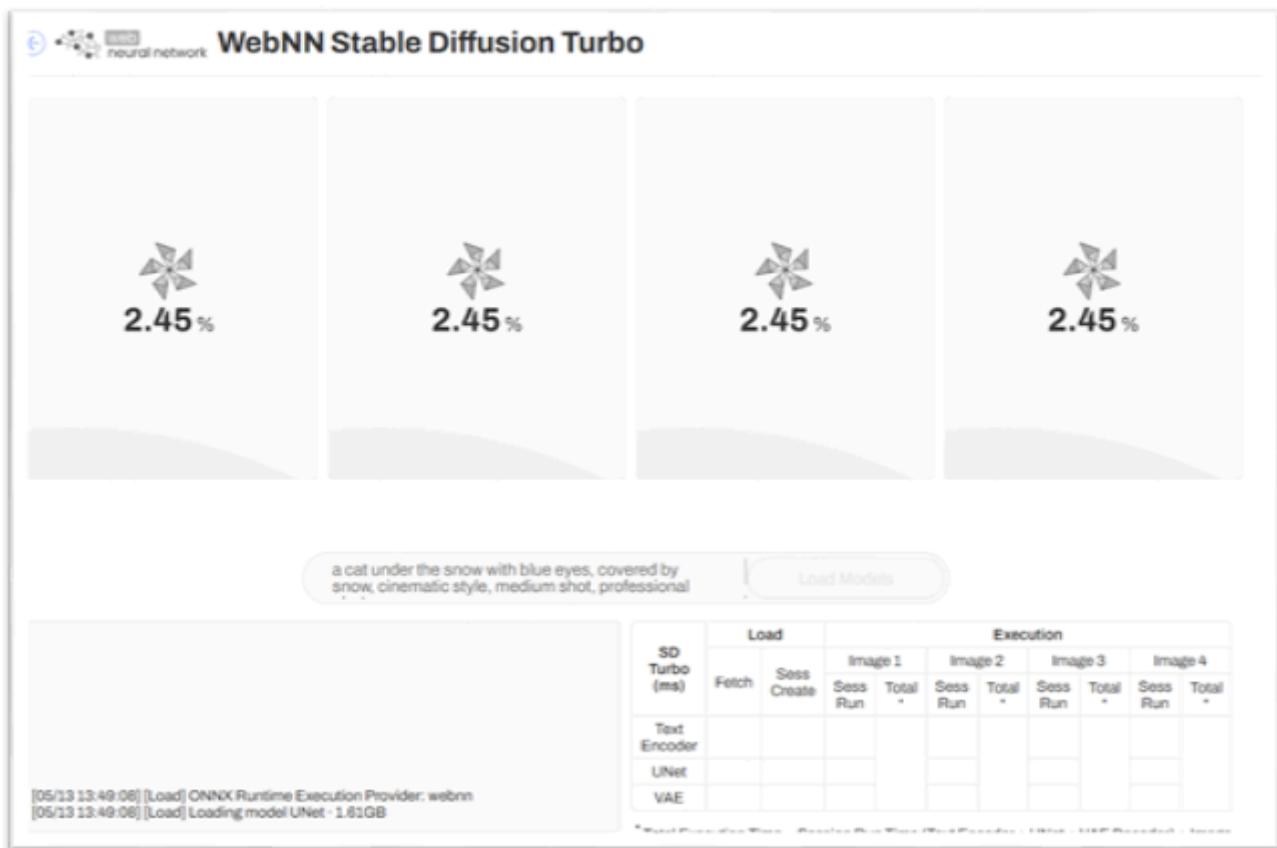
Features: [Phi3](#), [ONNX Runtime Generative AI](#), [DirectML](#)

App Type: [C#](#), [WinUI 3](#)

App Type: [C#](#), [WPF](#)

Local Hardware Acceleration through DirectML

Hardware accelerated Stable Diffusion on the web



GitHub Repo: [WebNN Stable Diffusion Turbo](#)

Description: This sample illustrates how to use WebNN with ONNX Runtime web to run Stable Diffusion locally on the GPU with DirectML. [SD-Turbo](#) is a fast generative text-to-image model that can synthesize photorealistic images from a text prompt in a single network evaluation. In the demo, you can generate an image in 2s on AI PC devices by leveraging WebNN API, a dedicated low-level API for neural network inference hardware acceleration.

Features: Local Image Generation, [WebNN](#), DirectML

App Type: [JavaScript](#), Web apps

Hardware accelerated Segment Anything on the web

GitHub Repo: [WebNN Segment Anything](#)

Description: This sample illustrates how to use WebNN with ONNX Runtime web to run Segment Anything locally on the GPU with DirectML. [Segment Anything](#) is a new AI model from Meta AI that can "cut out" any object. In the demo, you can segment any object from your uploaded images.

Features: Local Image Segmentation, [WebNN](#), DirectML

App Type: [JavaScript](#), Web apps

Hardware accelerated Whisper on the web

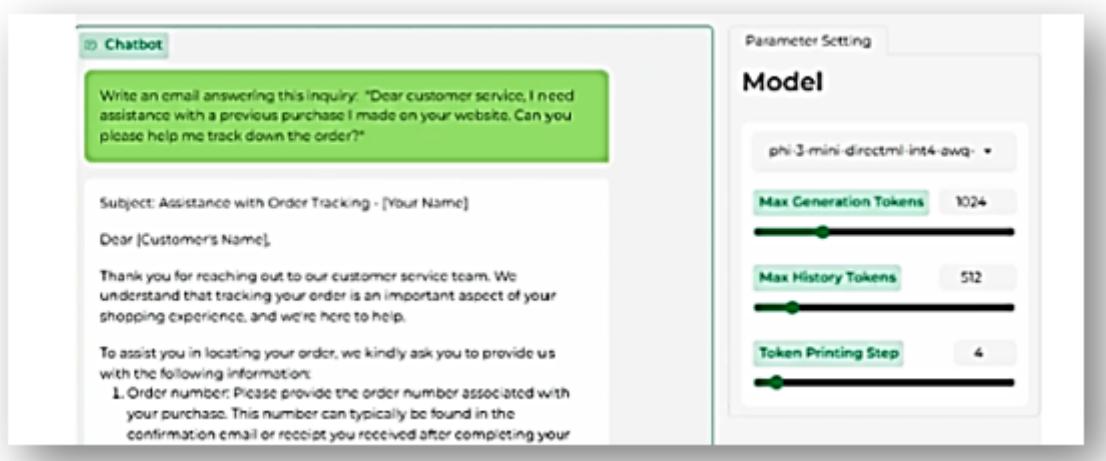
GitHub Repo: [WebNN Whisper Base ↗](#)

Description: This sample illustrates how to use WebNN with ONNX Runtime web to run the Whisper model's speech-to-text capabilities locally on the GPU or NPU with DirectML. [Whisper Base ↗](#) is a pre-trained model for automatic speech recognition (ASR) and speech translation. In the demo, you can experience the speech to text feature by using on-device inference powered by WebNN API and DirectML, especially the NPU acceleration.

Features: Local speech-to-text, [WebNN ↗](#), [DirectML](#)

App Type: [JavaScript](#), Web apps

Hardware accelerated and pre-optimized ONNX Runtime language models (Phi3, Llama3, etc) with DirectML



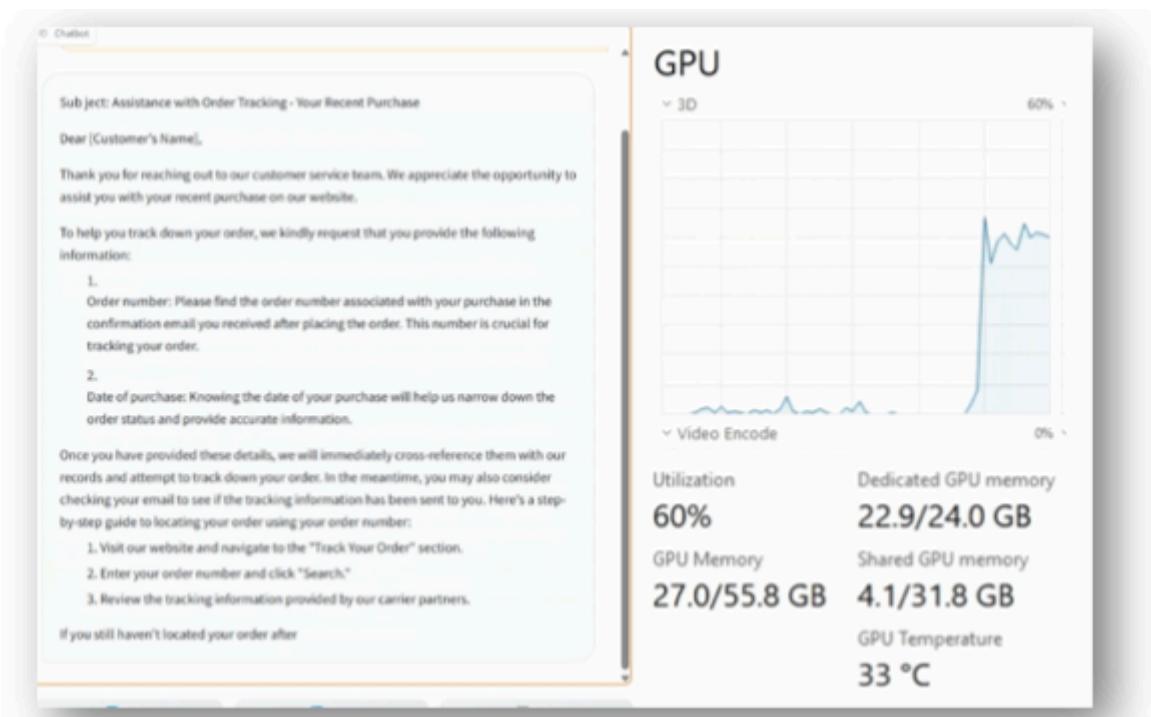
GitHub Repo: [DirectML examples in the Olive repo ↗](#)

Description: This sample illustrates how to run a pre-optimized ONNX Runtime (ORT) language model locally on the GPU with DirectML. The sample includes instructions on how to set up your environment, download the latest pre-trained language models using the ORT Generate API and run the model in a Gradio app.

Features: Hardware Acceleration, GenAI, [ONNX ↗](#), [ONNX Runtime ↗](#), [DirectML](#)

App Type: [Python](#), Gradio

Hardware accelerated PyTorch models (Phi3, Llama3, etc) with DirectML



GitHub Repo: [DirectML PyTorch samples](#)

Description: This sample illustrates how to run a PyTorch language model locally on the GPU with DirectML. The sample includes instructions on how to set up your environment, download the latest pre-trained language models and run the model in a Gradio app. This sample supports various open-source language models such as Llama models, Phi3-mini, Phi2 and Mistral-7B.

Features: Hardware Acceleration, [PyTorch](#), DirectML

App Type: Python, Gradio

Enhance your Windows apps with AI using cloud APIs

More cloud-based API samples can be found in the [Azure AI services documentation](#).

Add OpenAI chat completions to your WinUI 3 / Windows App SDK app

Tutorial: [Add OpenAI chat completions to your WinUI 3 / Windows App SDK app](#)

Description: Integrate the OpenAI chat completion capabilities into a WinUI 3 / Windows App SDK desktop app.

Features: OpenAI chat completion

App Type: [C#, WinUI 3](#)

Add DALL-E to your WinUI 3 / Windows App SDK desktop app

Tutorial: [Add DALL-E to your WinUI 3 / Windows App SDK desktop app](#)

Description: Integrate the OpenAI DALL-E image generation capabilities into a WinUI 3 / Windows App SDK desktop app.

Features: Image generation

App Type: [C#, WinUI 3](#)

Create a recommendation app with .NET MAUI and ChatGPT

Tutorial: [Create a recommendation app with .NET MAUI and ChatGPT](#)

Description: Integrate the OpenAI chat completion capabilities into a .NET MAUI desktop app.

Features: Image generation

App Type: [C#, .NET MAUI](#)

Add DALL-E to your .NET MAUI Windows desktop app

Tutorial: [Add DALL-E to your .NET MAUI Windows desktop app](#)

Description: Integrate the OpenAI DALL-E image generation capabilities into a .NET MAUI desktop app.

Features: Image generation

App Type: [C#, .NET MAUI](#)

Legacy WinML samples

GitHub Repo: [WinML samples on GitHub ↗](#)

Description: WinML continues to be supported, but these samples have not been updated to reflect modern AI use.

Last updated on 12/03/2025

AI Toolkit for Visual Studio Code overview

Important

This documentation has moved. For the latest cross-platform guidance, see [AI Toolkit for Visual Studio Code](#).

The AI Toolkit for Visual Studio Code is a VS Code extension that helps developers build generative AI applications locally. It provides access to AI models from the [Microsoft Foundry catalog](#), [Hugging Face](#), and additional catalogs. It also offers tools for downloading, fine-tuning, and deploying models on your local machine or NPU-enabled devices.

Tip

The **Deepseek R1 Distilled** model is optimized for the Neural Processing Unit (NPU) and available to download on Snapdragon powered Copilot+ PCs running Windows 11. For more information, see [Running Distilled DeepSeek R1 models locally on Copilot+ PCs, powered by Windows AI Foundry](#).

Related content

- [Developing Responsible Generative AI Applications and Features on Windows](#)
- [Model catalog and collections in Microsoft Foundry portal](#)
- [AI Dev Gallery](#)
- [Windows AI API samples](#)

Last updated on 12/19/2025

Develop AI applications for Copilot+ PCs

Copilot+ PCs are a new class of Windows 11 hardware powered by a high-performance Neural Processing Unit (NPU) — a specialized computer chip for AI-intensive processes like real-time translations and image generation—that can perform more than 40 trillion operations per second (TOPS). Copilot+ PCs provide all-day battery life and access to the most advanced AI features and models.

Learn more:

- [Empowering the future: The expanding Arm app ecosystem for Copilot+ PCs ↗](#)
- [Introducing Copilot+ PCs - The Official Microsoft Blog ↗](#).

The following Copilot+ PC Developer Guidance covers:

- Device Prerequisites
- What is the Arm-based Snapdragon Elite X+ chip?
- Unique AI features supported by Copilot+ PCs with an NPU processor
- How to access the NPU on a Copilot+ PC
- How to use ONNX Runtime to programmatically access the NPU on a Copilot+ PC
- How to measure performance of AI models running locally on the device NPU

Prerequisites

This guidance is specific to [Copilot+ PCs ↗](#).

Many of the new Windows AI features require an NPU with the ability to run at 40+ TOPS, including but not limited to:

- Microsoft Surface Laptop Copilot+ PC
- Microsoft Surface Pro Copilot + PC
- HP OmniBook X 14
- Dell Latitude 7455, XPS 13, and Inspiron 14
- Acer Swift 14 AI
- Lenovo Yoga Slim 7x and ThinkPad T14s
- Samsung Galaxy Book4 Edge
- ASUS Vivobook S 15 and ProArt PZ13
- [Copilot+ PCs with new AMD and Intel silicon ↗](#), including [AMD Ryzen AI 300 series ↗](#) and [Intel Core Ultra 200V series ↗](#).

Surface Copilot+ PCs for Business:

- Surface Laptop for Business with Intel Core Ultra processors (Series 2) [↗](#) (Available starting Feb. 18, 2025)
- Surface Pro for Business with Intel Core Ultra processors (Series 2) [↗](#) (Available starting Feb. 18, 2025)
- Introducing new Surface Copilot+ PCs for Business [↗](#)

What is the Arm-based Snapdragon Elite X chip?

The new Snapdragon X Elite Arm-based chip built by Qualcomm emphasizes AI integration through its industry-leading Neural Processing Unit (NPU). This NPU is able to process large amounts of data in parallel, performing trillions of operations per second, using energy on AI tasks more efficiently than a CPU or GPU resulting in longer device battery life. The NPU works in alignment with the CPU and GPU. Windows 11 assigns processing tasks to the most appropriate place in order to deliver fast and efficient performance. The NPU enables on-device AI intelligent experiences with Enterprise-grade security for enhanced protection from chip to cloud.

- Learn more about the [Qualcomm Snapdragon X Elite](#) [↗](#).
- Learn more about using and developing for [Windows on Arm](#).

Unique AI features supported by Copilot+ PCs with an NPU

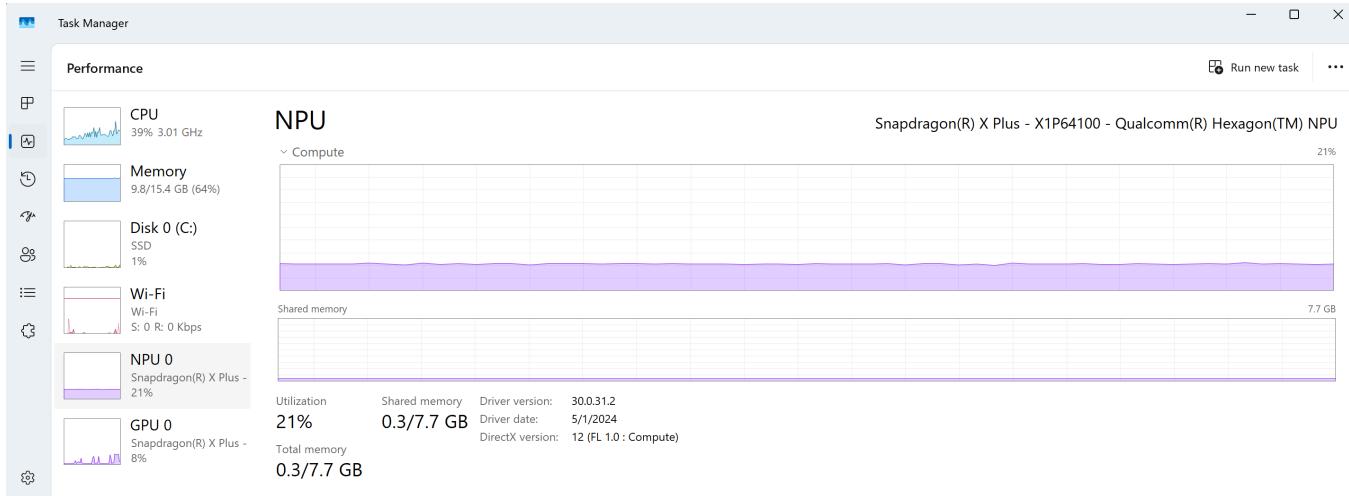
Copilot+ PCs offer unique AI experiences that ship with modern versions of Windows 11. These AI features, designed to run on the device NPU, ship in the latest releases of Windows and will be available via APIs in [Microsoft Foundry on Windows](#). Learn more about [Microsoft Foundry on Windows APIs](#) that are supported by models optimized to run (inference) on the NPU. These APIs will ship in an upcoming release of the [Windows App SDK](#).

How to access the NPU on a Copilot+ PC

The [Neural Processing Unit \(NPU\)](#) [↗](#) is a new hardware resource. Like other hardware resources on a PC, the NPU needs software to be specifically programmed to take advantage of the benefits it offers. NPUs are designed specifically to execute the deep learning math operations that make up AI models.

The Windows 11 Copilot+ AI features mentioned above have been specifically designed to take advantage of the NPU. Users will get improved battery life and faster inference execution time for AI models that target the NPU. Windows 11 support for NPUs will include Arm-based Qualcomm devices, as well as Intel and AMD devices (coming soon).

For devices with NPUs, the [Task Manager](#) can now be used to view NPU resource usage.



The recommended way to inference (run AI tasks) on the device NPU is to use [Windows ML](#).

How to programmatically access the NPU on a Copilot+ PC for AI acceleration

The recommended way to programmatically access the NPU (Neural Processing Unit) and GPU for AI acceleration has shifted from DirectML to Windows ML (WinML). This transition reflects a broader effort to simplify and optimize the developer experience for AI workloads on Windows devices. You can find updated guidance here: [Learn how Windows Machine Learning \(ML\) helps your Windows apps run AI models locally..](#)

- **Built-in EP discovery:** Previously, developers were required to know which Execution Providers (EPs) were compatible with their hardware and bundle those EPs with their applications. This often led to larger application sizes and increased complexity in managing dependencies. With Windows ML, the process is now automated and streamlined. Windows ML automatically detects the available hardware on the device and downloads the appropriate EPs as needed. This means that developers no longer need to bundle specific EPs with their applications, resulting in smaller application sizes and reduced complexity.
- **Integrated EP delivery:** The required EPs, such as Qualcomm's QNNExecutionProvider or Intel's OpenVINO EP, are now bundled with Windows or delivered via Windows Update, eliminating the need for manual downloads.
- **ORT under the hood:** Windows ML still uses ONNX Runtime as its inference engine, but abstracts away the complexity of EP management. [ONNX Runtime](#) is an open source inference and training engine for AI models using the ONNX format and enabling developers to build AI applications that can run efficiently across a wide range of devices.

- **Collaboration with hardware vendors:** Microsoft works directly with hardware vendors, such as Qualcomm and Intel, to ensure EP compatibility with early driver versions and new silicon (for example, Snapdragon X Elite, Intel Core Ultra, etc.).

When you deploy an AI model using Windows ML on a Copilot+ PC:

- Windows ML queries the system for available hardware accelerators.
- It selects the most performant EP (for example, QNN for Qualcomm NPUs, OpenVINO for Intel NPUs).
- The EP is loaded automatically, and inference begins.
- If the preferred EP fails or is unavailable, Windows ML gracefully falls back to another (for example, using the GPU or CPU).

This means developers can focus on building AI experiences without worrying about low-level hardware integration

Supported model formats

AI models are often trained and available in larger data formats, such as FP32. Many NPU devices, however, only support integer math in lower bit format, such as INT8, for increased performance and power efficiency. Therefore, AI models need to be converted (or "quantized") to run on the NPU. There are many models available that have already been converted into a ready-to-use format. You can also *bring your own model* (BYOM) to convert or optimize.

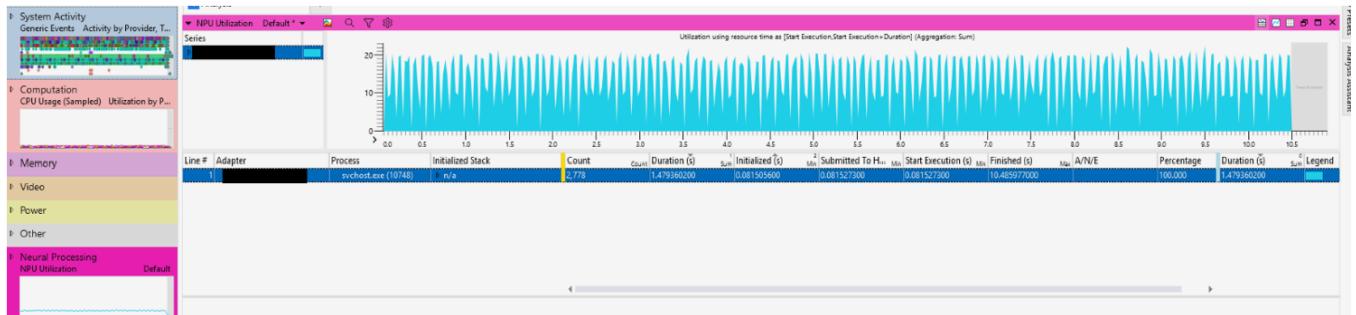
- [Qualcomm AI Hub \(Compute\)](#): Qualcomm provides AI models that have already been validated for use on Copilot+ PCs with Snapdragon X Elite with the available models specifically optimized to run efficiently on this NPU. Learn more: [Accelerate model deployment with Qualcomm AI Hub | Microsoft Build 2024](#).
- [ONNX Model Zoo](#): This open source repository offers a curated collection of pre-trained, state-of-the-art models in the ONNX format. These models are recommended for use with NPUs across all Copilot+ PCs, including Intel and AMD devices (coming soon).

For those who want to bring your own model, we recommend using the hardware-aware model optimization tool, [Olive](#). Olive can help with model compression, optimization, and compilation to work with ONNX Runtime as an NPU performance optimization solution. Learn more: [AI made easier: How the ONNX Runtime and Olive toolchain will help you Q&A | Build 2023](#).

How to measure performance of AI models running locally on the device NPU

To measure the performance of AI feature integration in your app and the associated AI model runtimes:

- **Record a trace:** Recording device activity over a period of time is known as system tracing. System tracing produces a "trace" file that can be used to generate a report and help you identify how to improve your app's performance. Learn more: [Capture a system trace to analyze memory usage](#).
- **View NPU usage:** Examine which processes are using the NPU and the callstacks submitting work.
- **View work and callstacks on the CPU:** Examine the results of the pre-work feeding AI models and post-work processing AI models.
- **Load and Runtime:** Examine the length of time to load an AI model and create an ONNX Runtime session.
- **Runtime parameters:** Examine ONNX Runtime configuration and Execution Provider (EP) parameters that affect model runtime performance and optimization.
- **Individual inference times:** Track per-inference times and sub-details from the NPU.
- **Profiler:** Profile AI model operations to see how long each operator took to contribute to the total inference time.
- **NPU-specific:** Examine NPU sub-details such as sub-HW metrics, memory bandwidth, and more.



To perform these measurements, we recommend the following diagnostic and tracing tools:

- **Task Manager:** Enables a user to view the performance of the Windows Operating System installed on their device, including Processes, Performance, App history, Startup apps, Users, Details, and Services. Real-time performance data will be shown for your device CPU, Memory, Storage Disk, Wi-Fi, GPU... and now NPU. Data includes the percentage of utilization, available memory, shared memory, driver version, physical location, and more.
- **Windows Performance Recorder (WPR):** WPR now ships with a Neural Processing profile to record NPU activity. This records [Microsoft Compute Driver Model \(MCDM\)](#)

interactions with the NPU. Developers can now see NPU usage, which processes are using the NPU, and the callstacks submitting work.

- **Windows Performance Analyzer (WPA)**: WPA creates graphs and data tables of Event Tracing for Windows (ETW) events that are recorded by Windows Performance Recorder (WPR), Xperf, or an assessment that is run in the Assessment Platform. It provides convenient access points for analyzing the CPU, Disk, Network, ONNX Runtime Events... and a new table for [NPU analysis](#), all in a single timeline. WPA can now view the work and callstacks on the CPU related to pre-work feeding AI models and post-work processing AI model results. [Download Windows Performance Analyzer from the Microsoft Store](#).
- **GPUView**: GPUView is a development tool that reads logged video and kernel events from an event trace log (.etl) file and presents the data graphically to the user. This tool now includes both GPU and NPU operations, as well as support for viewing DirectX events for [MCDM](#) devices like the NPU.
- **ONNX Runtime events in Windows Performance Analyzer**: Starting with ONNXRuntime 1.17 (and enhanced in 1.18.1) the following use cases are available with events emitted in the runtime:
 - See how long it took to load an AI model and create an ONNX Runtime session.
 - See ONNX Runtime configuration and Execution Provider (EP) parameters that affect model runtime performance and optimization.
 - Track per inference times and sub-details from the NPU (QNN).
 - Profile AI Model operations to see how long each operator took to contribute to the total inference time.
 - Learn more about [ONNX Runtime Execution Provider \(EP\) Profiling](#).

Note

WPR UI (the user interface available to support the command-line based WPR included in Windows), WPA, and GPUView are all part of Windows Performance Toolkit (WPT), version May 2024+. To use the WPT, you will need to: [Download the Windows ADK Toolkit](#).

For a quickstart on viewing ONNX Runtime events with the Windows Performance Analyzer (WPA), follow these steps:

1. Download [ort.wprp](#) and [etw_provider.wprp](#).

2. Open your command line and enter:

PowerShell

```
wpr -start ort.wprp -start etw_provider.wprp -start NeuralProcessing -start CPU
echo Repro the issue allowing ONNX to run
wpr -stop onnx_NPU.etl -compress
```

3. Combine the Windows Performance Recorder (WPR) profiles with other [Built-in Recording Profiles](#) such as CPU, Disk, etc.
4. [Download Windows Performance Analyzer \(WPA\) from the Microsoft Store](#).
5. Open the `onnx_NPU.etl` file in WPA. Double-Click to open these graphs:

- "Neural Processing -> NPU Utilization
- Generic Events for ONNX events

Additional performance measurement tools to consider using with the Microsoft Windows tools listed above, include:

- [Qualcomm Snapdragon Profiler](#) (qprof): A GUI and system-wide performance profiling tool designed to visualize system performance, as well as identify optimization and application scaling improvement opportunities across Qualcomm SoC CPUs, GPUs, DSPs and other IP blocks. The Snapdragon Profiler allows viewing NPU sub-details, such as sub-HW metrics, memory bandwidth and more.

Additional Resources

- [Microsoft Foundry on Windows overview](#)
- [Windows app performance and fundamentals overview](#)
- [Windows on Arm overview- Empowering the future: The expanding Arm app ecosystem for Copilot+ PCs](#)
- [What is Windows ML](#)
- [All about neural processing units \(NPUs\)](#)

Last updated on 11/17/2025

Developing Responsible Generative AI Applications and Features on Windows

This document provides an overview of recommended responsible development practices to use as you create applications and features on Windows with generative artificial intelligence.

[Microsoft Foundry on Windows](#) on-device generative AI models can help you to enforce local content safety features, such as on-device classification engines for harmful content and a default blocklist. Microsoft prioritizes supporting developers to build safe, trustworthy AI experiences with local models on Windows.

Guidelines for responsible development of generative AI apps and features on Windows

Every team at Microsoft follows [core principles and practices](#) to responsibly build and ship AI, including Windows. You can read more about Microsoft's approach to responsible development in the [Microsoft Responsible AI Transparency Report](#). Windows follows foundational pillars of RAI development — govern, map, measure, and manage — that are aligned to the National Institute for Standards and Technology (NIST) AI Risk Management Framework.

Govern - Policies, practices, and processes

Standards are the foundation of governance and compliance processes. Microsoft has developed our own [Responsible AI Standard](#), including [six principles](#) that you can use as a starting point to develop your guidelines for responsible AI. We recommend you build AI principles into your development lifecycle end to end, as well as into your processes and workflows for compliance with laws and regulations across privacy, security, and responsible AI. This spans from early assessment of each AI feature, using tools like the [AI Fairness Checklist](#) and [Guidelines for Human-AI Interaction - Microsoft Research](#), to monitoring and review of AI benchmarks, testing and processes using tools like a [Responsible AI scorecard](#), to public documentation into your AI features' capabilities and limitations and user disclosure and controls -- notice, consent, data collection and processing information, etc. -- in keeping with applicable privacy laws, regulatory requirements, and policies.

Map - Identify risk

Recommended practices for identifying risks include:

End-to-end testing

End-to-end testing evaluates the entire AI system from start to finish to ensure that it operates as intended and adheres to established standards. This comprehensive approach may include:

Red teaming

The term [red teaming](#) has historically described systematic adversarial attacks for testing security vulnerabilities. More recently, the term has extended beyond traditional cybersecurity and evolved in common usage to describe many kinds of probing, testing, and attacking of AI systems.

With both large language models (LLMs) and small language models (SLMs), both benign and adversarial usage may produce potentially harmful outputs that can take many forms, including hate speech, incitement or glorification of violence, or sexual content. Thorough red teaming allows you to stress-test your system and refine your content strategy to decrease the possibility that your system causes harm.

All AI systems should undergo red team testing, depending on function and purpose, for both high-risk systems that employ generative AI and lower-risk systems that use non-generative AI:

- **Formal red teaming:** Independent red teaming should be completed for all high-risk systems that employ generative AI using large language models (LLMs). Formal red teaming includes recruiting professionals outside of your organization to participate in red teaming activities.
- **Internal red teaming:** At a minimum, plan internal red teaming for all lower-risk, non-generative AI systems. This can be done by people inside your organization.

Learn more about red teaming and how to assess your system's red teaming needs: [Microsoft AI Red Team](#)

Model evaluation

As a part of end-to-end testing, it is important to evaluate the model itself.

- **Model Card:** For publicly available models, such as those on HuggingFace, you can check each model's Model Card as a handy reference to understand if a model is the right one for your use case. [Read more about Model Cards ↗](#).
- **Manual testing:** Humans performing step-by-step tests without scripts is an important component of model evaluation that supports...

- Measuring progress on a small set of priority issues. When mitigating specific harms, it's often most productive to keep manually checking progress against a small dataset until the harm is no longer observed before moving to automated measurement.
 - Defining and reporting metrics until automated measurement is reliable enough to use alone.
 - Spot-checking periodically to measure the quality of automatic measurement.
- **Automated testing:** Automatically executed testing is also an important component of model evaluation that supports...
 - Measuring at a large scale with increased coverage to provide more comprehensive results.
 - Ongoing measurement to monitor for any regression as the system, usage, and mitigations evolve.
 - **Model selection:** Select a model that is suited for your purpose and educate yourself to understand its capabilities, limitations, and potential safety challenges. When testing your model, make sure that it produces results appropriate for your use. To get you started, destinations for Microsoft (and non-Microsoft/open source) model sources include:
 - [Hugging Face](#) ↗
 - [ONNX Model Zoo](#) ↗
 - [Qualcomm AI Hub](#) ↗
 - [AI Toolkit for VS Code](#)
 - [PyTorch Hub](#) ↗
 - [Tensorflow Hub](#) ↗

Measure - Assess risks and mitigation

Recommended practices include:

- **Assign a Content Moderator:** Content Moderator checks text, image, and video content for material that is potentially offensive, risky, or otherwise undesirable in content. Learn more: [Introduction to Content Moderator \(Microsoft Learn Training\)](#).
- **Use content safety filters:** This ensemble of multi-class classification models detects four categories of harmful content (violence, hate, sexual, and self-harm) at various

severity levels (low, medium, and high). Learn more: [How to configure content filters with Azure OpenAI Service](#).

- **Apply a meta-prompt:** A meta-prompt is a system message included at the beginning of the prompt and is used to prime the model with context, instructions, or other information relevant to your use case. These instructions are used to guide the model's behavior. Learn more: [Creating effective security guardrails with metaprompt / system message engineering ↗](#).
- **Utilize blocklists:** This blocks the use of certain terms or patterns in a prompt. Learn more: [Use a blocklist in Azure OpenAI](#).
- **Get familiar with the provenance of the model:** Provenance is the history of ownership of a model, or the who-what-where-when, and is very important to understand. Who collected the data in a model? Who does the data pertain to? What kind of data is used? Where was the data collected? When was the data collected? Knowing where model data came from can help you assess its quality, reliability, and avoid any unethical, unfair, biased, or inaccurate data use.
- **Use a standard pipeline:** Use one content moderation pipeline rather than pulling together parts piecemeal. Learn more: [What are Azure Machine Learning pipelines?](#).
- **Apply UI mitigations:** These provide important clarity to your user about capabilities and limitations of an AI-based feature. To help users and provide transparency about your feature, you can:
 - Encourage users to edit outputs before accepting them
 - Highlight potential inaccuracies in AI outputs
 - Disclose AI's role in the interaction
 - Cite references and sources
 - Limit length of input and output where appropriate
 - Provide structure out input or output – prompts must follow a standard format
 - Prepare pre-determined responses for controversial prompts.
- **Implement customer feedback loops:** Encourage your users to actively engage in feedback loops:
 - Ask for feedback directly in your app / product using a simple feedback mechanism that is available in context as part of the user experience.

- Apply social listening techniques on the channels your customers use for early conversations about feature issues, concerns, and possible harm.

Manage - Mitigate AI risks

Recommendations for mitigating AI risks include:

- **Abuse monitoring:** This methodology detects and mitigates instances of recurring content and/or behaviors that suggest a service has been used in a manner that may violate the Code of Conduct or other applicable product terms. Learn more: [Abuse Monitoring](#).
- **Phased delivery:** Roll out your AI solution slowly to handle incoming reports and concerns.
- **Incident response plan:** For every high-priority risk, evaluate what will happen and how long it will take to respond to an incident, and what the response process will look like.
- **Ability to turn feature or system off:** Provide functionality to turn the feature off if an incident is about to or has occurred that requires pausing the functionality to avoid further harm.
- **User access controls/blocking:** Develop a way to block users who are misusing a system.
- **User feedback:** Utilize mechanisms to detect issues from the user's side.
 - Ask for feedback directly in your product, with a simple feedback mechanism that is available in the context of a typical workflow.
 - Apply social listening techniques on the channels your customers use for early conversations about feature issues, concerns, and possible harm.
- **Responsible deployment of telemetry data:** Identify, collect, and monitor signals that indicate user satisfaction or their ability to use the system as intended, ensuring you follow applicable privacy laws, policies, and commitments. Use telemetry data to identify gaps and improve the system.

Tools and resources

- **Microsoft Foundry on Windows:** A unified, reliable and secure platform supporting the AI developer lifecycle from model selection, finetuning, optimizing and deployment across CPU, GPU, NPU and cloud.

- **Responsible AI Toolbox** [↗](#): Responsible AI is an approach to assessing, developing, and deploying AI systems in a safe, trustworthy and ethical manner. The Responsible AI toolbox is a suite of tools providing a collection of model and data exploration and assessment user interfaces and libraries that enable a better understanding of AI systems. These interfaces and libraries empower developers and stakeholders of AI systems to develop and monitor AI more responsibly and take better data-driven actions.
- **Responsible AI Dashboard Model Debugging** [↗](#): This dashboard can help you to Identify, Diagnose, and Mitigate issues, using data to take informed actions. This customizable experience can be taken in a multitude of directions, from analyzing the model or data holistically, to conducting a deep dive or comparison on cohorts of interest, to explaining and perturbing model predictions for individual instances, and to informing users on business decisions and actions. [Take the Responsible AI Decision Making Quiz](#) [↗](#).
- Review the Azure Machine Learning summary of [What is Responsible AI?](#)
- Read the [Approach to Responsible AI for Copilot in Bing](#) [↗](#).
- Read Brad Smith's article on [Combating abusive AI-generated content: a comprehensive approach](#) [↗](#) from Feb 13, 2024.
- Read the [Microsoft Security Blog](#) [↗](#).
- [Overview of Responsible AI practices for Azure OpenAI models - Azure AI services](#)
- [How to use content filters \(preview\) with Azure OpenAI Service](#)
- [How to use blocklists with Azure OpenAI Service](#)
- [Planning red teaming for large language models \(LLMs\) and their applications](#)
- [Azure OpenAI Service abuse monitoring](#)
- [Threat modeling AI/ML systems and dependencies](#)
- [AI/ML pivots to the security. A development lifecycle bug bar](#)
- [Failure modes in machine learning](#)
- [Tools for Managing and Ideating Responsible AI Mitigations - Microsoft Research](#) [↗](#)
- [Planning for natural language failures with the AI Playbook](#) [↗](#)
- [Software engineering for ML: A case study](#) [↗](#)
- [Security and machine learning in the real world](#) [↗](#)

- Overreliance on AI: Literature review ↗
 - Error Analysis ↗ and Build Responsible AI using Error Analysis toolkit (youtube.com) ↗
 - InterpretML ↗ and How to Explain Models with InterpretML Deep Dive (youtube.com) ↗
 - Black-Box and Glass-Box Explanation in Machine Learning (youtube.com) ↗
-

Last updated on 11/17/2025

Choose between cloud-based and local AI models

For app developers seeking to integrate AI features, Microsoft Windows offers a comprehensive and flexible platform that supports both local, on-device processing and scalable, cloud-based solutions.

Choosing between cloud-based and local AI models depends on your specific needs and priorities. Factors to consider include:

- Data privacy, compliance, and security
- Resource availability
- Accessibility and collaboration
- Cost
- Maintenance and updates
- Performance & latency
- Scalability
- Connectivity requirements
- Model size and complexity
- Tooling and the associated ecosystem
- Customization and control

Key decision factors for app developers

- **Data privacy, compliance, and security**
 - ◦ **Local, on-premises:** Since data remains on the device, running a model locally can offer benefits regarding security and privacy, with the responsibility of data security resting on the user. The developer holds responsibility for managing updates, ensuring compatibility, and monitoring security vulnerabilities.
 - ◦ **Cloud:** Cloud providers offer robust security measures, but data needs to be transferred to the cloud, which might raise data privacy concerns for the business or app service maintainer in some cases. Sending data to the cloud also must comply with data protection regulations, such as GDPR or HIPAA, depending on the nature of the data and the region in which the app operates. Cloud providers typically handle security updates and maintenance, but users must ensure that they are using secure APIs and following best practices for data handling.
- **Resource availability**
 - **Local, on-premises:** Running a model depends on the resources available on the device being used, including the CPU, GPU, NPU, memory, and storage capacity. This

can be limiting if the device does not have high computational power or sufficient storage. Small Language Models (SLMs), like [Phi](#), are more suitable for local use on a device. [Copilot+ PCs](#) offer built-in models with ready-to-use AI features supported by [Microsoft Foundry on Windows](#).

- ○ **Cloud:** Cloud platforms, such as [Azure AI Services](#), offer scalable resources. You can use as much computational power or storage as you need and only pay for what you use. Large Language Models (LLMs), like the [OpenAI language models](#), require more resources, but are also more powerful.

- **Accessibility and collaboration**

- ○ **Local, on-premises:** The model and data are accessible only on the device unless shared manually. This has the potential to make collaboration on model data more challenging.
- ○ **Cloud:** The model and data can be accessed from anywhere with internet connectivity. This may be better for collaboration scenarios.

- **Cost**

- **Local, on-premises:** There is no additional cost beyond the initial investment in the device hardware.
- **Cloud:** While cloud platforms operate on a pay-as-you-go model, costs can accumulate based on the resources used and the duration of usage.

- **Maintenance and Updates**

- **Local, on-premises:** The user is responsible for maintaining the system and installing updates.
- **Cloud:** Maintenance, system updates, and new feature updates are handled by the cloud service provider, reducing maintenance overhead for the user.

- **Performance & Latency**

- **Local, on-premises:** Running a model locally can reduce latency since data does not need to be sent over the network. However, performance is limited by the device's hardware capabilities.
- **Cloud:** Cloud-based models can leverage powerful hardware, but they may introduce latency due to network communication. The performance can vary based on the user's internet connection and the cloud service's response time.

- **Scalability**

- **Local, on-premises:** Scaling a model on a local device may require significant hardware upgrades or the addition of more devices, which can be costly and time-consuming.
 - **Cloud:** Cloud platforms offer easy scalability, allowing you to quickly adjust resources based on demand without the need for physical hardware changes.
- **Connectivity Requirements**
 - **Local, on-premises:** A local device does not require an internet connection to run a model, which can be beneficial in environments with limited connectivity.
 - **Cloud:** Cloud-based models require a stable internet connection for access and may be affected by network issues.
 - **Model Size and Complexity**
 - **Local, on-premises:** Local devices may have limitations on the size and complexity of models that can be run due to hardware constraints. Smaller models, such as [Phi](#), are more suitable for local execution.
 - **Cloud:** Cloud platforms can handle larger and more complex models, such as those provided by OpenAI, due to their scalable infrastructure.
 - **Tooling and the associated ecosystem**
 - **Local, on-premises:** Local AI solutions, such as Microsoft Foundry on Windows, Windows ML, and Foundry Local, integrate with Windows App SDK and ONNX Runtime, allowing developers to embed models directly into desktop or edge apps with minimal external dependencies.
 - **Cloud:** Cloud AI solutions, such as Microsoft Foundry, Azure AI Services, and Azure OpenAI Service, provide a comprehensive set of APIs and SDKs for building AI applications. These services are designed to integrate seamlessly with Azure DevOps, GitHub Copilot, Semantic Kernel, and other Azure services, enabling end-to-end orchestration, model deployment, and monitoring at scale.
 - **Customization and Control**
 - **Local, on-premises:** Local models can be used out-of-the-box, without requiring a high level of expertise. Microsoft Foundry on Windows offers models like [Phi Silica](#) that are ready to use. Alternatively, [Windows ML](#) allows developers to run custom models, such as those trained with ONNX Runtime, directly on Windows devices. This provides a high level of control over the model and its behavior, allowing for fine-tuning and optimization based on specific use cases. [Foundry Local](#) also allows developers to run

models locally on Windows devices, providing a high level of control over the model and its behavior.

- **Cloud:** Cloud-based models also offer both ready-to-use and customizable options, allowing developers to leverage pre-trained capabilities while still tailoring the model to their specific needs. [Microsoft Foundry](#) is a unified Azure platform-as-a-service offering for enterprise AI operations, model builders, and application development. This foundation combines production-grade infrastructure with friendly interfaces, enabling developers to focus on building applications rather than managing infrastructure.

Cloud AI Samples

If a cloud-based solution works better for your Windows app scenario, you may be interested in some of the tutorials below.

Many APIs are available for accessing cloud-based models to power AI features in your Windows app, whether those models are customized or ready-to-use. Using a cloud-based model can allow your app to remain streamlined by delegating resource-intensive tasks to the cloud. A few resources to help you add cloud-based AI-backed APIs offered by Microsoft or OpenAI include:

- [Add OpenAI chat completions to your WinUI 3 / Windows App SDK desktop app](#): A tutorial on how to integrate the cloud-based OpenAI ChatGPT completion capabilities into a WinUI 3 / Windows App SDK desktop app.
- [Add DALL-E to your WinUI 3 / Windows App SDK desktop app](#): A tutorial on how to integrate the cloud-based OpenAI DALL-E image generation capabilities into a WinUI 3 / Windows App SDK desktop app.
- [Create a recommendation app with .NET MAUI and ChatGPT](#): A tutorial on how to create a sample Recommendation app that integrates the cloud-based OpenAI ChatGPT completion capabilities into a .NET MAUI app.
- [Add DALL-E to your .NET MAUI Windows desktop app](#): A tutorial on how to integrate the cloud-based OpenAI DALL-E image generation capabilities into a .NET MAUI app.
- [Azure OpenAI Service](#): If you want your Windows app to access OpenAI models, such as GPT-4, GPT-4 Turbo with Vision, GPT-3.5-Turbo, DALLE-3 or the Embeddings model series, with the added security and enterprise capabilities of Azure, you can find guidance in this Azure OpenAI documentation.

- **Azure AI Services:** Azure offers an entire suite of AI services available through REST APIs and client library SDKs in popular development languages. For more information, see each service's documentation. These cloud-based services help developers and organizations rapidly create intelligent, cutting-edge, market-ready, and responsible applications with out-of-the-box and prebuilt and customizable APIs and models. Example applications include natural language processing for conversations, search, monitoring, translation, speech, vision, and decision-making.
-

Last updated on 11/17/2025

Frequently Asked Questions about using AI in Windows apps

General

How can I integrate AI into my Windows client app?

Integrating AI into your Windows application can be achieved through two primary methods: a local model or a cloud-based model. For the local model option, you have the ability to utilize a pre-existing model or train your own using platforms like TensorFlow or PyTorch, and then incorporate it into your application via [OnnxRuntime](#). [Microsoft Foundry on Windows](#) offers APIs for various functions, including OCR or utilizing the Phi Silica model. On the other hand, hosting your model on the cloud and accessing it through a REST API allows your application to remain streamlined by delegating resource-intensive tasks to the cloud. See [Use Machine Learning models in your Windows app](#) for more information.

Do I need the latest version of Windows 11 and a Copilot+ PC with an NPU to use AI features?

There are many ways to run AI workloads, both by installing and running models locally on your Windows device or by running cloud-based models (see [Get started with AI on Windows](#)), however, the AI features supported by [Windows AI APIs](#) currently require a [Copilot+ PC](#) with an NPU.

What programming languages are best for developing AI in Windows client apps?

You can use any programming language you prefer. For instance, C# is widely used for creating Windows client apps. If you require more control over low-level details, C++ is an excellent option. Alternatively, you might consider using [Python](#). You can also use the [Windows Subsystem for Linux](#) (WSL) to run Linux-based AI tools on Windows.

What are the best AI frameworks for Windows client apps?

We recommend using [OnnxRuntime](#).

How should I handle data privacy and security when using AI in Windows client apps?

Respecting the privacy and security of user data is essential when developing AI-powered apps. You should follow best practices for data handling, such as encrypting sensitive data, using secure connections, and obtaining user consent before collecting data. You should also be transparent about how you are using data and give users control over their data. Make sure to read [Developing Responsible Generative AI Applications and Features on Windows](#) too.

What are the system requirements for running AI in Windows client apps?

System requirements for Windows apps that use AI depend on the complexity of the AI model and the hardware acceleration used. For simple models, a modern CPU may be sufficient, but for more complex models, a GPU or NPU may be required. You should also consider the memory and storage requirements of your app, as well as the network bandwidth required for cloud-based AI services.

How to optimize AI performance in Windows client apps?

To optimize AI performance in Windows apps, you should consider using hardware acceleration, such as GPUs or NPUs, to speed up model inference. Windows Copilot+ laptops are optimized for AI workloads and can provide a significant performance boost for AI tasks. See also [AI Toolkit for Visual Studio Code overview](#).

Can I use pre-trained AI models in my Windows client app?

Yes, you can use pre-trained AI models in your Windows app. You can download pre-trained models from the internet or use a cloud-based AI service to access pre-trained models. You can then integrate these models into your app using a framework like OnnxRuntime.

What is DirectML?

DirectML is a low-level API for machine learning that provides GPU acceleration for common machine learning tasks across a broad range of supported hardware and drivers, including all DirectX 12-capable GPUs from vendors such as AMD, Intel, NVIDIA, and Qualcomm.

What is ONNX?

Open Neural Network Exchange, or ONNX, is an open standard format for representing ML models. Popular ML model frameworks, such as PyTorch, TensorFlow, SciKit-Learn, Keras, Chainer, MATLAB, etc., can be exported or converted to the standard ONNX format. Once in ONNX format, the model can run on a variety of platforms and devices. ONNX is good for using an ML model in a different format than it was trained on.

What is ORT?

OnnxRuntime, or ORT, is a unified runtime tool for executing models in different frameworks (PyTorch, TensorFlow, etc) that supports hardware accelerators (device CPUs, GPUs, or NPUs).

How does ONNX differ from other ML frameworks, like PyTorch or TensorFlow?

PyTorch and TensorFlow are used for developing, training, and running deep learning models used in AI applications. PyTorch is often used for research, TensorFlow is often used for industry deployment, and ONNX is a standardized *model exchange format* that bridges the gap, allowing you to switch between frameworks as needed and compatible across platforms.

What is an NPU? How is different from a CPU or GPU?

A Neural Processing Unit, or NPU, is a dedicated AI chip designed specifically to perform AI tasks. The focus of an NPU differs from that of a CPU or GPU. A Central Processing Unit, or CPU, is the primary processor in a computer, responsible for executing instructions and general-purpose computations. A Graphics Processing Unit, or GPU, is a specialized processor designed for rendering graphics and optimized for parallel processing. It is capable of rendering complex imagery for video editing and gaming tasks.

NPUs are designed to accelerate deep learning algorithms and can remove some of the work from a computer's CPU or GPU, so the device can work more efficiently. NPUs are purpose-built for accelerating neural network tasks. They excel in processing large amounts of data in parallel, making them ideal for common AI tasks like image recognition or natural language processing. As an example, during an image recognition task, the NPU may be responsible for object detection or image acceleration, while the GPU takes responsibility for image rendering.

How can I find out what sort of CPU, GPU, or NPU my device has?

To check the type of CPU, GPU, or NPU on your Windows device and how it's performing, open Task Manager (Ctrl + Shift + Esc), then select the *Performance* tab and you will be able to see your machine's CPU, Memory, Wi-Fi, GPU, and/or NPU listed, along with information about its speed, utilization rate, and other data.

What is Windows ML?

Windows ML (Machine Learning) enables your app to use a shared system-wide copy of the ONNX Runtime (ORT, see above), and adds support to dynamically download vendor-specific **execution providers** (EPs) so that your model inference can be optimized across the wide variety of CPUs, GPUs, and NPUs in the Windows ecosystem without requiring your app to carry heavy runtimes or EPs itself.

Helpful AI concepts

What is a Large Language Model (LLM)?

An LLM is a type of Machine Learning (ML) model known for the ability to achieve general-purpose language generation and understanding. LLMs are artificial neural networks that acquire capabilities by learning statistical relationships from vast amounts of text documents during a computationally intensive self-supervised and semi-supervised training process. LLMs are often used for Text Generation, a form of generative AI that, given some input text, generates words (or "tokens") that are most likely to create coherent and contextually relevant sentences in return. There are also Small Language Models (SLMs) that have fewer parameters and more limited capacity, but may be more efficient (requiring less computational resources), cost-effective, and ideal for specific domains.

What is ML model training?

In Machine Learning, model training involves feeding a dataset into a model (an LLM or SLM), allowing it to learn from the data so that the model can make predictions or decisions based on that data, recognizing patterns. It may also involve adjusting the model parameters iteratively to optimize its performance.

What is Inferencing?

The process of using a trained machine learning model to make predictions or classifications on new, unseen data is called “Inferencing”. Once a language model has been trained on a dataset, learning its underlying patterns and relationships, it's ready to apply this knowledge to real-world scenarios. Inference is an AI model's moment of truth, a test of how well it can apply information learned during training to make a prediction or solve a task. The process of using an existing model for inference is different from the training phase, which requires the use of training and validation data to develop the model and fine-tune its parameters.

What is ML model fine-tuning?

Fine-tuning is a crucial step in machine learning where a pre-trained model is adapted to perform a specific task. Instead of training a model from scratch, fine-tuning starts with an existing model (usually trained on a large dataset) and adjusts its parameters using a smaller, task-specific dataset. By fine-tuning, the model learns task-specific features while retaining the general knowledge acquired during pre-training, resulting in improved performance for specific applications.

What is prompt engineering?

Prompt engineering is a strategic approach used with generative AI to shape the behavior and responses of a language model. It involves thoughtfully crafting input prompts or queries to achieve the desired result from a language model (like GPT-3 or GPT-4). By designing an effective prompt, you can guide an ML model to produce the type of response you want. Techniques include adjusting the wording, specifying context, or using control codes to influence model output.

What is hardware acceleration (in regard to ML model training)?

Hardware acceleration refers to the use of specialized computer hardware designed to speed up AI applications beyond what is achievable with general-purpose CPUs. Hardware acceleration enhances the speed, energy efficiency, and overall performance of machine learning tasks, such as training models, making predictions, or offloading computation to dedicated hardware components that excel at parallel processing for deep learning workloads. GPUs and NPUs are both examples of hardware accelerators.

What are the differences between a Data Scientist, ML Engineer, and App Developer who wants apply AI features in their app?

The process of creating and using ML models involves three main roles: **Data Scientists**: Responsible for defining the problem, collecting and analyzing the data, choosing and training the ML algorithm, and evaluating and interpreting the results. They use tools such as Python, R, Jupyter Notebook, TensorFlow, PyTorch, and scikit-learn to perform these tasks. **ML Engineers**: Responsible for deploying, monitoring, and maintaining the ML models in production environments. They use tools such as Docker, Kubernetes, Azure ML, AWS SageMaker, and Google Cloud AI Platform to ensure the scalability, reliability, and security of the ML models. **App Developers**: Responsible for integrating the ML models into the app logic, UI, and UX. They use tools such as Microsoft Foundry on Windows, OnnxRuntime, or REST APIs and process the user input and model output. Each role involves different responsibilities and skills, but collaboration and communication between these roles is required to achieve the best results. Depending on the size and complexity of the project, these roles can be performed by the same person or by different teams.

Model Context Protocol (MCP) on Windows

ⓘ Note

Some information relates to prereleased product that might change substantially before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

MCP on Windows provides the Windows On-device Agent Registry (ODR), a secure, manageable interface to discover and use agent connectors from local apps and remote servers using [Model Context Protocol \(MCP\)](#). MCP is an open protocol designed to standardize integrations between AI apps and external tools and data sources. By using MCP on Windows to discover and interact with MCP servers, app developers can enhance the capabilities of AI models, enabling them to produce more accurate, relevant, context-aware responses, and to expand the tasks they can complete for users.

About the ODR

The Windows ODR enables apps and agents to securely discover and access MCP servers on Windows. Benefits of using the registry include:

- **Broad, standardized MCP support:** the registry supports MCP servers from local apps and remote servers
- **Discoverability:** agents can easily find and connect to all available MCP servers
- **Security:** MCP servers are [contained in a separate environment](#) by default and can only access approved resources, limiting vulnerability to threats like cross-prompt injection attacks
- **User and admin control:** users and IT admins can control access to MCP servers for each agent by using Windows Settings and management tools like Microsoft Intune
- **Logging and auditability:** users and administrators can log and audit interactions between MCP clients and servers
- **Windows connectors:** Windows includes default connectors for agents to use, including a [File Explorer MCP server](#)

For information about registering MCP servers with Windows ODR, see [Register an MCP server on Windows](#).

The Windows ODR also provides a command line tool, `odr.exe`, that enables users and developers to view and manage MCP servers. For more information, see: [The Windows on-](#)

device agent registry command-line tool (odr.exe).

MCP agent connectors on Windows

An agent connector is a packaged integration point that allows an AI agent to connect to external capabilities through the Model Context Protocol (MCP). It acts as the bridge between the agent and an MCP server, enabling the agent to discover and use tools, resources, and prompts exposed by that server. Windows File Explorer implements an MCP connector that provides a set of tools to access and modify files using a Model Context Protocol (MCP) server. For more information, see [Agent Launchers on Windows overview](#).

Several agents that support using MCP servers on Windows are currently available, including:

- [Windows Settings connector](#)
- [Visual Studio GitHub Copilot agent mode](#)
- [Visual Studio Code GitHub Copilot agent mode ↗](#)

There are frameworks that enable you to build your own agents that use the Windows ODR to access MCP servers, including:

- [Microsoft Agent Framework](#) - a development kit for building AI agents and agent workflows

The Windows File Explorer MCP connector integrates MCP server tools into the context menus for working with files and folders in File Explorer. For more information, see [Windows File Explorer MCP connector](#).

For more information and examples of creating a host app that uses MCP to list, connect to, and interact with the MCP servers registered on Windows, see: [Quickstart: MCP host on Windows](#).

Next steps

- [Create an MCP server on Windows](#)
- [Test your MCP server](#)
- [Create an MCP host on Windows](#)

MCP servers on Windows overview

⚠ 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.

This section provides information about MCP server registration on Windows. Registering your MCP server enables agents on Windows to easily discover and connect to your server, provides greater security for your users and services, and enables additional management when your server is used in enterprise environments.

For information about building an MCP server, look at one of the available SDKs including:

- [MCP C# SDK](#) - an SDK for building MCP clients and servers for .NET apps and libraries
- [MCP TypeScript SDK](#) - an SDK for building MCP clients and servers using TypeScript

Information on additional SDKs is available here: <https://modelcontextprotocol.io/docs/sdk>.

Register an MCP server with Windows

Once you've built an MCP server, you can register it with Windows in several ways. The method you choose depends on how your application is packaged and deployed.

Apps with package identity

Applications with package identity can register with Windows by including required metadata in your app package. The OS will automatically register and unregister your server when the app package is installed and uninstalled. For more information, see [Register an MCP server from an app with package identity](#).

Apps are granted package identity when they are packaged using the MSIX package format. Unpackaged apps can be referenced in an MSIX package, granting them package identity, by using packaging with external location. For more information, see:

- [An overview of Package Identity in Windows apps](#)
- [What is MSIX?](#)
- [Grant package identity by packaging with external location](#)

Apps without identity

If you have an server without package identity, such as a .exe file, a file packaged using MSI, or a standalone MCP bundle, and you don't want to use MSIX with external location to grant package identity, you can install an MCP bundle directly with your installer. Directly installed MCP bundles can't run in the securely contained agent process and will not be accessible from the Windows on-device agent registry unless users explicitly enable the option to Reduce protections for agent connectors in Windows Settings. For information on installing registering an MCP server distributed as an MCP bundle, see [Register an MCP server with an MCP bundle](#).

Manual registration

If you want to register a remote MCP servers or if you need fine-grained control when registering a local MCP server, you can manually register your server using the Windows on-device agent registry command-line tool. For more information, see [Manually register remote and local MCP servers](#).

MCP server containment

By default, MCP servers accessed through the Windows on-device agent registry (ODR) are run in an agent session, securely contained in a separate environment and can only allowed access to approved resources, limiting vulnerability to threats like cross-prompt injection attacks. For information about the restrictions and requirements of MCP server containment, see [Securely containing MCP servers on Windows](#).

Testing your MCP server on Windows

MCP on Windows provides a few different ways to test your MCP registration, to validate that it is being recognized by the ODR, and to test the functionality of your MCP server on Windows. For more information, see [Testing MCP servers on Windows](#).

Last updated on 11/18/2025

Register an MCP server from an app with package identity

This article describes how an app with package identity can register an MCP server by walking through a sample from the MCP on Windows samples repo, github.com/microsoft/mcp-on-windows-samples, to illustrate how an app with package identity registers an MCP server. When you add the required metadata to your packaged app, the OS will automatically register the server when the app package is installed. Apps that use the MSIX package format have package identity. Unpackaged apps can be granted package identity by building and registering a package with external location with your app. For more information, see [Grant package identity by packaging with external location](#).

If you have an app that doesn't have package identity, such as one that is installed using MSI or just a standalone .exe, and you don't want to use MSIX packaging to grant it package identity, you can install an MCP server using an MCP Bundle. For more information, see [Register an MCP server with an MCP bundle](#).

! 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.

Prerequisites

- Windows build 26220.7262 or higher
- Ensure you have the latest [SignTool.exe](#), version 10.0.26100.4188 or greater. SignTool.exe ships with the [Windows SDK](#). You can get the latest Windows SDK using WinGet.
`winget install Microsoft.WindowsSDK.10.0.26100`
- For production releases, you will need a certificate that is part of the [Microsoft Trusted Root Program](#).
- An MCP server. For more information, see [MCP server overview](#)
- An app with package identity and an AppxManifest.xml file
- NodeJS installed. To install NodeJS using WinGet, use the following command:
 - `winget install OpenJS.NodeJS`

Overview

Adding an MCP server to an app with package identity includes the following steps:

- Add an MCP Bundle `manifest.json` file that describes your MCP server to your project. For more information on the format of this file, see the MCP bundle github repo, github.com/anthropics/mcpb/.
- Add an `uap3:AppExtension` entry to your `AppxManifest.xml` file.

Clone the sample

Clone the MCP on Windows samples repo.

PowerShell

```
git clone https://github.com/microsoft/mcp-on-windows-samples.git
```

The solution file for the MSIX app that registers the MCP server is in the `mcp-on-windows-samples\msix-app-with-server\` directory.

Set up and build the sample

Open the `.sln` file from the sample in Visual Studio, select the correct architecture for your device from the **Solution Platforms** drop-down.

The app requires a certificate to build successfully. To add a test certificate:

1. In **Solution Explorer**, under the `mcp-server-msix` project, double-click the file `Package.appxmanifest` to open the package manifest editor.
2. On the **Packaging** tab, click **Choose Certificate**.
3. In the **Choose a Certificate** dialog, click **Create...**
4. Leave the default values in all of the fields and click **OK**. Click **OK** again to complete the certificate creation process.

Build the sample. In **Solution Explorer**, right-click the `mcp-server-msix` project and select **Deploy**. Now run the app.

Next, launch the MCP server by opening a PowerShell prompt to that folder and running the following command. You may need to change the path based on your machine's architecture.

PowerShell

```
npx @modelcontextprotocol/inspector .\McpServer\bin\x64\Debug\net8.0\McpServer.exe
```

This command will start the MCP server. It will also launch the MCP inspector, a utility for interacting with MCP servers, in a browser Window. Switch back to the sample app to use the

app's UI to interact with the MCP server.

The rest of this walkthrough will call out the components of this sample app that cause it to be registered upon installation.

The MCP bundle (MCPB) config JSON file

The MCP bundle (MCPB) config JSON file describes your server and how to interact with it. The MCP bundle config JSON file for the example project is found in the file `Assets/manifest.json`. The code listing is shown below. Note that it provides metadata, such as a name, description, and author for the server. Next it defines three different tools and defines the expected input types. For a detailed description of the MCPB config file format, see [MCPB Manifest.json Spec](#) on the [MCPB github repo](#).

Note that the values in the `_meta` section of the MCPB config file must match what your server returns at runtime in order for your server to run in default mode. For more information, see [The _meta section of the MCPB config JSON file](#).

JSON

```
{  
  "manifest_version": "0.1",  
  "name": "MCP-Server-Sample-App",  
  "version": "1.0.0",  
  "description": "A Sample App MCP Server",  
  "author": {  
    "name": "Microsoft"  
  },  
  "server": {  
    "type": "binary",  
    "entry_point": "SampleMCPServer.McpServer.exe",  
    "mcp_config": {  
      "command": "SampleMCPServer.McpServer.exe",  
      "args": []  
    }  
  },  
  "tools": [  
    {  
      "name": "get_random_fact",  
      "description": "Gets a random interesting fact from an online API."  
    },  
    {  
      "name": "get_random_quote",  
      "description": "Gets a random inspirational quote from an online API."  
    },  
    {  
      "name": "get_weather",  
      "description": "Gets current weather information for a specified city."  
    }  
  ]}
```

```
],
  "tools_generated": false,
  "license": "MIT",
  "_meta": {
    "com.microsoft.windows": {
      "static_responses": {
        "initialize": {
          "protocolVersion": "2025-06-18",
          "capabilities": {
            "logging": {},
            "tools": {
              "listChanged": true
            }
          },
          "serverInfo": {
            "name": "McpServer",
            "version": "1.0.0.0"
          }
        },
        "tools/list": {
          "tools": [
            {
              "name": "get_random_quote",
              "description": "Gets a random inspirational quote from an online API.",
              "inputSchema": {
                "type": "object",
                "properties": {}
              }
            },
            {
              "name": "get_random_fact",
              "description": "Gets a random interesting fact from an online API.",
              "inputSchema": {
                "type": "object",
                "properties": {}
              }
            },
            {
              "name": "get_weather",
              "description": "Gets current weather information for a specified city.",
              "inputSchema": {
                "type": "object",
                "properties": {
                  "city": {
                    "type": "string",
                    "default": "London"
                  }
                }
              }
            }
          ]
        }
      }
    }
  }
}
```

```
}
```

The MCP app extension entry in the package manifest file

You register your MCP server with the system by adding a `uap3:AppExtension` element to your app's package manifest file, `AppxManifest.xml`. The name of the extension is `com.microsoft.windows.ai.mcpServer`. This extension allows Windows to register and unregister your MCP server when the app is installed or uninstalled. The `uap5:AppExecutionAlias` tells the system the executable for your MCP server.

XML

```
<Extensions>
    <uap3:Extension Category="windows.appExtension">
        <uap3:AppExtension
            Name="com.microsoft.windows.ai.mcpServer"
            Id="WindowsMCPServerSampleApp"
            DisplayName="Windows Sample MCP Server"
            PublicFolder="Assets">
            <uap3:Properties>
                <Registration>mcpServerConfig.json</Registration>
            </uap3:Properties>
        </uap3:AppExtension>
    </uap3:Extension>

    <uap5:Extension Category="windows.appExecutionAlias">
        <uap5:AppExecutionAlias>
            <uap5:ExecutionAlias Alias="WindowsMCPServerSampleApp.exe" />
        </uap5:AppExecutionAlias>
    </uap5:Extension>
</Extensions>
```

You can view the code of the sample project by right-clicking the `Package.appxmanifest` file and selecting **View Code**.

Request capabilities for your server

An MCP server registered with a package app will run in a contained environment, which means that by default it won't have access to protected resources like the files of the current user. For more information, see [Securely containing MCP servers on Windows](#).

You can request access to sets of resources by declaring capabilities in your app manifest file. When your server is accessed, the system will prompt the user allow access to the requested resources. Most capabilities can be set directly in the manifest editor UI. Some common resources that can be accessed via capabilities include:

- documentsLibrary – Grants access to the user's Documents folder.
- downloadsFolder – Grants access to the user's Downloads folder.
- picturesLibrary – Grants access to the user's Pictures folder.
- musicLibrary – Grants access to the user's Music folder.
- videosLibrary – Grants access to the user's Videos folder.

For more information about capabilities, see [App Capability Declarations](#).

Test your MCP server

For information on testing the registration and functionality of an MCP server, see [Testing MCP servers on Windows](#).

Last updated on 12/16/2025

Register an MCP server with an MCP bundle

This article will walk through the process of registering an MCP server using an MCP bundle, a package format that standardize deployment for MCP servers. For more information on this package format, see the MCP bundle github repo, github.com/anthropics/mcpb/ You can include an MCPB into your existing Windows app, or you can distribute the MCPB on its own.

Apps that are packaged using the MSIX package format can include metadata in their package that will automatically register the MCP server when the package is installed. For more information see, [Register an MCP server from an app with package identity](#).

⚠ Warning

MCP servers run in an agent session that runs under its own user account. MCP bundles are not supported in the agent session, and are therefore not supported by default. Servers registered using MCP bundles will not be accessible from the Windows on-device agent registry. For testing purposes, you can enable MCP bundles for agent sessions. For more information including important security caveats, see [Reducing protections for agent connectors](#).

ⓘ 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.

Prerequisites

- Windows build 26220.7262 or higher
- Developer mode enabled. For more information, see [Developer Mode features and debugging](#).
- The .NET SDK. For installation information, see [Install .NET on Windows](#).

Install the MCP bundle tooling package

Open the command line and install the NPM package with this command:

PowerShell

```
dotnet tool install -g mcpb.cli
```

Build your MCP Bundle

You can initialize and build your bundle by running `mcpb init` and `mcpb pack`. For more information, see the [MCPB .NET CLI github repo](#).

You can also use the [MCP server C# sample](#) to generate a MCP bundle to test with using the following commands:

PowerShell

```
git clone https://github.com/microsoft/mcp-on-windows-samples.git
cd mcp-on-windows-samples
cd mcp-server-csharp
.\build-mcpb.ps1
```

These commands will create a file named `mcp-dotnet-mcpb-server.mcpb` in the `mcp-server-csharp` folder.

The `_meta` section of the MCPB config JSON file

MCP on Windows and the on-device agent registry (ODR) require the inclusion of the `_meta` section of the MCP bundle manifest schema, which is intended to hold optional extensions to the schema. ODR uses the following properties that are declared in the `_meta` section of the manifest, as children of the `com.microsoft.windows` object.

[] Expand table

Property	Description
package_family_name	The server's package family name. The value of this property is always generated by ODR. If a server provides them while registering, they are ignored.
static_responses	The MCP server being registered must provide this section which lists all capabilities and tools provided by the server. This list must match the runtime responses list exactly. This allows ODR to validate that the only expected tools and capabilities are provided at runtime and enables tool discovery for MCP hosts without launching the MCP server, which can generate consent prompts and the associated performance costs.

(i) Important

If you want your MCP server to run in default mode, the values returned by your server implementation must match the values declared in your MCPB file.

Some examples of values in the MCPB file that must match those returned by your server implementation include the following:

- The top-level `name` and `version` properties must match the `name` and `version` property in `serverInfo` inside the `initialize` section.
- The top-level `tools` must be present in the `tools/list` richer schema.
- The server must not dynamically change the set of tools, their descriptions, or input or output schemas.
- The server's `initialize` and `tools/list` at runtime must match what is in the manifest.

If you update any of these values in your server, you must update the MCPB config file to match the new values or your MCP server will no longer run in default mode. The .NET CLI MCPB can be used to produce and validate valid `_meta` fields for your server.

The following shows an example `_meta` section from an MCPB config file.

JSON

```
{  
  "_meta": {  
    "com.microsoft.windows": {  
      "static_responses": {  
        "initialize": {  
          "protocolVersion": "2025-06-18",  
          "capabilities": {  
            "logging": {},  
            "tools": {  
              "listChanged": true  
            }  
          },  
          "serverInfo": {  
            "name": "ProductStockSystem.McpServer",  
            "version": "1.0.0.0"  
          }  
        },  
        "tools/list": {  
          "tools": [  
            {  
              "name": "list_products",  
              "description": "Get a list of all active products in the stock  
system",  
              "inputSchema": {  
                "type": "object",  
                "properties": {}  
              },  
              "outputSchema": {  
                "type": "array",  
                "items": {  
                  "type": "object",  
                  "properties": {}  
                }  
              }  
            }  
          ]  
        }  
      }  
    }  
  }  
}
```

```
"type": "object",
"properties": {
  "products": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer"
        },
        "name": {
          "type": "string"
        },
        "description": {
          "type": "string"
        },
        "sku": {
          "type": "string"
        },
        "price": {
          "type": "number"
        },
        "stockQuantity": {
          "type": "integer"
        },
        "reorderLevel": {
          "type": "integer"
        },
        "isLowStock": {
          "type": "boolean"
        }
      },
      "required": [
        "id",
        "name",
        "description",
        "sku",
        "price",
        "stockQuantity",
        "reorderLevel",
        "isLowStock"
      ]
    }
  },
  "count": {
    "type": "integer"
  },
  "success": {
    "type": "boolean"
  },
  "error": {
    "type": [
      "string",
      "null"
    ]
  }
}
```

```
        }
    },
    "required": [
        "products",
        "count",
        "success"
    ]
}
}
}
}
}
```

Distribute your `.mcpb` file or include it as part of your app

You can distribute the `.mcpb` file directly, as-is, to ship a standalone MCP server or you can integrate it as part of your application by executing the `.mcpb` bundle file during install of your application.

Test your MCP server

You can now test that your MCP server shows up correctly as part of regular app install by test installing your app and then using the [testing guide](#) to interact with it. For more information, see [Testing MCP servers on Windows](#).

Next steps

- Learn about the features of the agent session for MCP for Windows and understand why you might want to move your app to use MSIX and package identity in order to use the agent session. For more information, see [Securely containing MCP servers on Windows](#).

Manually register remote and local MCP servers

This article describes how to register and unregister MCP servers by interacting directly with the Windows On-Device Registry (ODR). Manual registration is required to register remote MCP servers. Manual registration also enables fine-grained control when registering a local MCP server. For typical scenarios we recommend registering local MCP servers using the automatic registration features of packaged apps or, for non-packaged apps, including an MCP bundle in your app installed. For more information, see [Register an MCP server from an app with package identity](#) or [Register an MCP server with an MCP bundle](#) guides instead where possible.

ⓘ 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.

Prerequisites

- Windows build 26220.7262 or higher

The `odr.exe` command line interface

`odr.exe` is the CLI that enables MCP servers to be listed, registered, and interacted with on Windows. For more information about this tool, see the [Windows odr.exe on-device agent registry tool](#).

Register a server

If you have a remote MCP server you can register with the following command:

PowerShell

```
odr.exe mcp add --uri <url-to-your-server>
```

If you have a `mcpb` manifest JSON file you can register a local server with the following command:

PowerShell

```
odr.exe mcp add <path-to-mcpb-manifest>-json
```

For more information on the format of the `manifest.json` file, see the MCP bundle github repo, github.com/anthropics/mcpb/.

List installed servers

The following command outputs a list of all MCP servers that are registered on the machine.

PowerShell

```
odr.exe list
```

Unregister a server

The following command unregisters an MPC server.

PowerShell

```
odr.exe mcp remove <mcp_server_name>
```

This unregisters a specific server.

Last updated on 11/18/2025

Securely containing MCP servers on Windows

ⓘ 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.

By default, MCP servers accessed through the Windows on-device agent registry (ODR) are run in an agent session, securely contained in a separate environment and can only allowed access to approved resources, limiting vulnerability to threats like cross-prompt injection attacks. This article describes security requirements for MCP servers to run in an agent session, the access restrictions for agent sessions, and how to grant agents access to additional resources.

Overview

MCP servers that are contained run in a separate Windows session using a separate agent user account. This includes all servers provided by Windows and all MCP servers that are registered with the Windows on-device agent registry using MSIX packages or packages with external locations. For information about server registration, see [Register an MCP server on Windows](#).

Servers which are packaged with MCP bundles currently cannot run contained in an agent session. For more information about registering MCP servers using MCP bundles, see [Register an MCP server with an MCP bundle](#).

Requirements

To run contained, MCP servers must meet the following requirements:

- The server must be implemented as a binary (.exe) server
- The server must have package identity and be registered via an MSIX package extension
- The server must expose a valid manifest.json registration, including at minimum the following fields:
 - manifest_version
 - name
 - version
 - description
 - author

- o server
- o `_meta` with `com.microsoft.windows` definition (including `static_responses` and `tools/list`)

For information about packaging an MCP server to meet these requirements, see [Register an MCP server from an app with package identity](#).

What a server can and can't do when contained

A contained MCP server has its own identity and runs in a separate Windows session using a separate agent user account. Therefore, a server does not have direct access to the user's session, including:

- user files (unless the user grants permission to the agent)
- user settings, registry, and credentials
- apps and windows the user is using
- running executables that modify the user session

However, a server running in the agent session can:

- Access the internet
- Read and write to the agent's files and registry
- Run executables in the agent user environment
- Run executables in the agent session
- Access certain user files with the user's consent

Requesting access to user files

While running in the agent session, an MCP server will not have access to the user files by default. However, an MCP server can request access to user files by declaring the specific known folder capabilities in their package appxmanifest.xml. A host app is an app that the user interacts with directly and acts as a proxy between the user and one or more MCP servers. If an MCP server that requests a particular capability is used by a host, the user will be prompted to grant access to the associated resource. Once the user grants access, all MCP servers used by that host app will have access to the resource. For example, if one MCP server used by the host requests user file access and the access is granted, any MCP server used by the same host app will be able to access the user files.

 **Note**

Permissions to user files are granted for the host and not per server. When the user grants access to their user files, any MCP server used in that session will have access to the user's files.

Reducing protections for agent connectors

Any packaged apps with identity will always run in a contained session in this preview.

Unpackaged applications, including MCP bundles (.mcpb files) will not be able to run in containment. For testing purposes, you can enable this setting in Windows Settings to enable MCP servers accessed through the Windows on-device agent registry to run with reduced protections:

Important

This setting enables MCP servers to run with more access and privileges, and may expose your device to additional security threats.

`Settings > System > Advanced > AI components > Reduce protections for agent connectors`

To make your MCP server run in the user session, currently in the public preview you would need to deploy it as an unpackaged application.

Last updated on 11/18/2025

Test MCP servers on Windows

This article describes several ways that you can test and validate the registration and functionality of your MCP server.

! 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.

Test that the MCP functions work correctly

After building your MCP server into a binary executable, you can test out its functionality using the MCP inspector, a browser-based interface for interacting with an MCP server. This tool requires that NodeJS is installed on your device. To install NodeJS using WinGet, use the following command: - `winget install OpenJS.NodeJS`

To run the tool, open PowerShell and run this command:

PowerShell

```
npx @modelcontextprotocol/inspector <path to your .exe>
```

This will open a page in your web browser that allows you to manually test the functionality of your server.

Test that the MCP server has been registered on Windows correctly

Once your MCP server is part of a Windows app and is registered on Windows, you can test that it was registered successfully using this command in PowerShell:

PowerShell

```
odr.exe list
```

If your server shows up correctly in that list, then it has been registered successfully.

Quickstart: MCP host on Windows

This article shows how an MCP host app can list, connect to, and interact with the MCP servers registered on Windows using the [Windows odr.exe on-device agent registry tool](#) `odr.exe`. This walkthrough will use a sample host app from the MCP on Windows samples repo, github.com/microsoft/mcp-on-windows-samples.

! 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.

Prerequisites

- Windows build 26220.7262 or higher
- An MCP host app with package identity. For more information on package identity, see [An overview of Package Identity in Windows apps](#). Package identity is granted to apps that are packaged using the MSIX package format. For more information, see [What is MSIX?](#).
 - **Note** This requirement is not enforced in the public preview release but it will be in the stable release.

Clone the sample

Clone the [MCP on Windows host sample](#) to your device and navigate to it:

PowerShell

```
git clone https://github.com/microsoft/mcp-on-windows-samples.git
cd mcp-on-windows-samples/mcp-client-js
```

Set up and build the sample

Run these commands:

PowerShell

```
npm install
npm run start
```

The tool will present you with a command line UI that lets you interact with the MCP servers registered on your device. The following sections will show the Javascript code used by the tool to implement various features of an MCP host app.

List available MCP servers

List the available MCP servers executing the command line call `odr.exe list`. This command returns the list of servers in JSON format, which is stored and used in subsequent examples:

JavaScript

```
const { stdout, stderr } = await execFileAsync('odr.exe', ['list']);

if (stderr) {
  console.error('Warning:', stderr);
}

const servers = JSON.parse(stdout);
```

Connect to an MCP server

Connect to one of the available MCP servers by getting the command and arguments from the JSON returned in the previous step. Create a `StdioClientTransport`, passing in the command and arguments. Create a new `Client` object. Call `connect` to connect to the MCP server.

JavaScript

```
const command = server.manifest?.server?.mcp_config?.command;
const args = server.manifest?.server?.mcp_config?.args || [];

if (!command) {
  throw new Error('Server configuration missing command.');
}

// Create MCP client with stdio transport
// Set stderr to 'ignore' to silence server info logs
const transport = new StdioClientTransport({
  command: command,
  args: args,
  stderr: 'ignore'
});

const client = new Client({
  name: 'mcp-client',
  version: '1.0.0'
}, {
  capabilities: {}
```

```
});  
  
// Connect to the server  
await client.connect(transport);
```

List tools from a server

Call `listTools` to list the tools that are registered by the MCP server.

JavaScript

```
// List available tools  
const toolsResponse = await client.listTools();  
const tools = toolsResponse.tools || [];
```

Calling a tool

Each MCP tool has a name and an optional set of parameters. The `gatherToolParameters` function in the sample will help gather input parameters, and then you can call the tool directly:

JavaScript

```
const parameters = await gatherToolParameters(tool); // This function is from the  
sample code  
  
const result = await client.callTool({  
    name: tool.name,  
    arguments: parameters  
});
```

Next Steps

- Learn how to build and register an MCP server that can be discovered and used by a host app. For more information, see [Registering an MCP server](#).

Windows File Explorer MCP connector

(!) 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 Windows File Explorer MCP connector provides a set of tools to access and modify files using a Model Context Protocol (MCP) server. This connector provides access to common user folders including Documents, Desktop, Downloads, Music, Videos and Pictures, as well as public folders on the device. A feature of MCP that makes it so powerful is that agents can dynamically discover the tools that an MCP server provides, as well as the input and output parameters for each tool. So there is no need to hard code calls to a particular tool into a connector app.

The File Explorer MPC connect tools and parameters are shown below to illustrate the kinds of tools an MCP can provide.

Permissions

(i) Important

File access requires explicit user permission. Users and administrators may deny access to files.

File tools

[] Expand table

Tool Name	Input	Output	Notes
get_file_details	<code>path</code> (string)	<code>extension</code> (string/null), <code>size</code> (integer), <code>creationTime</code> , <code>lastAccessTime</code> , <code>lastWriteTime</code>	Input is a single file path; output provides metadata only, no file content. Read-only operation.
create_directory	<code>path</code> (string), <code>directoryName</code> (string)	<code>result</code> (string)	Creates a new directory; destructive operation

Tool Name	Input	Output	Notes
			because it changes the file system.
<code>move_file</code>	<code>oldFullPath</code> (string), <code>newFullPath</code> (string)	<code>result</code> (string)	Moves or renames a file; destructive because it alters file location.
<code>create_text_file</code>	<code>path</code> (string), <code>filename</code> (string), <code>content</code> (string)	<code>result</code> (string)	Creates a text file with optional content; destructive operation.
<code>read_file</code>	<code>filePath</code> (string)	<code>contents</code> (array), <code>_meta</code> (object/null)	Reads file content as a resource; read-only operation.
<code>get_directories</code>	<code>path</code> (string)	<code>result</code> (array of objects with <code>path</code>)	Lists directories within a given path; read-only operation.
<code>unzip_folder</code>	<code>zipPath</code> (string), <code>extractPath</code> (string)	<code>path</code> (string/null)	Decompresses a zip file into a folder; destructive because it writes files.
<code>zip_folder</code>	<code>folderPath</code> (string), <code>zipName</code> (string)	<code>path</code> (string/null)	Compresses a folder into a zip file; destructive because it creates a new file.
<code>read_text_file</code>	<code>filePath</code> (string)	<code>result</code> (string)	Reads text content from a file; supports Office and PDF formats; read-only.
<code>edit_text_file</code>	<code>filePath</code> (string), <code>oldText</code> (string), <code>newText</code> (string)	<code>result</code> (string)	Edits text in a file by replacing old text with new; destructive operation.
<code>search_files</code>	<code>startingDir</code> (string), <code>searchPatterns</code> (string), optional filters like <code>specifiedFileTypes</code> , <code>startDate</code> , <code>endDate</code>	<code>result</code> (array of objects with <code>name</code> , <code>path</code> , <code>dateCreated</code> , <code>snippet</code>)	Searches files by name, extension, date range; can return snippets for text-based files. On Copilot+ PCs, this can include semantic file search using natural language.

The Windows on-device agent registry command-line tool (odr.exe)

! 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 Windows on-device agent registry command-line tool `odr.exe` provides discovery and management tools for on-device agents in Windows, including registered [MCP servers](#).

Syntax

Windows Command Prompt

```
odr [command] [options]
```

Parameters

 [Expand table](#)

Command	Description
mcp	commands related to Model Context Protocol servers

 [Expand table](#)

Option	Description
-?, -h, --help	Show help and usage information
--version	Show version information
--verbose	Enable verbose logging

MCP Commands

Windows Command Prompt

```
odr mcp [command] [options]
```

[Expand table](#)

Command	Description
run	Run MCP server
list	List registered MCP servers
add <manifest file path>	Register an MCP server
remove <server id>	Unregister an MCP server
configure <server id>	Configure an MCP server

Last updated on 11/18/2025

App Actions on Windows Overview

App Actions on Windows are individual units of behavior that a Windows app can implement and register so that they can be accessed from other apps and experiences, seamlessly integrating into user workflows.

What is an App Action?

An App Action is an atomic unit of functionality. Apps build and register actions, and then Windows or other apps can recommend registered actions to the user at contextually relevant times and locations within the user workflow.

App Action implementation

Actions can be implemented by handling URI launch activation, or by through COM activation by implementing the **IActionProvider** interface. For a walkthrough of implementing a simple app action provider using URI activation, see [Get started with App Actions on Windows](#).

Apps must have package identity in order to register an app action. The MSIX package manifest provides metadata about the actions that are supported by the provider app. For more information on the app package manifest syntax for App Action registration, see [actions-provider-manifest.md](#).

Actions are defined using a JSON format that provides metadata about one or more actions, which includes information like the unique identifier and description for the action as well as the list of inputs and outputs that the action operates on. The JSON action definition file is packaged with the provider app as content. The path to the file within the package is specified in the app package manifest so that the system can find and ingest the action definitions. For more information on the JSON format for declaring actions, see [Action definition JSON schema for Windows App Action providers](#).

An *entity* is an object that an App Action operates on. Actions take entities as inputs and can return entities as outputs. Entities are divided into subtypes to represent different types of content that an action can operate on, such Document, Photo, and Text. Each entity type has a set of properties that provide information related to each content type, such as the path or file extension of a file. Entities are expressed as JSON in the action definition JSON file to declare the inputs and outputs of an app action. A set of WinRT APIs representing entities are also available for working with entities in code. For more information, see the [Windows.AI.Actions Namespace](#).

Responsible AI Notes

When building AI backed actions, it is your responsibility as the Action author to perform content moderation and abuse monitoring when it comes to entities returned to the user. For more information about Microsoft Responsible AI policies for more information, see [Microsoft Responsible AI: Principles and approach](#)

 **Note**

Consider if children should have access to the action using the 'contentAgeRating' property in the action definition JSON.

Recommended scenarios for App Actions

App Actions are intended to provide atomic units of functionality that are applicable to scenarios and workflows outside of the provider app. For example, an action might translate a piece of text, or process an image. For scenarios that are entirely specific to the Windows app that implements the behavior, the recommended path is to implement a custom extensibility point with an app extension. For more information, see [Create and host an app extension](#).

The following list describes some kinds of functionality that can make good candidates for implementing as an action.

- The functionality broadly applicable and reusable. The functionality is intended for discovery and reuse across multiple apps or contexts (e.g., file operations, printing).
- Other apps can compose and extend the functionality.
- The functionality is context-dependent and should be dynamically discovered at runtime (e.g., displaying context-specific commands in a UI).
- The functionality integrates with existing system tools or other app ecosystems.
- The functionality simplifies user interaction by encapsulating complex tasks into a single, higher-level action (e.g., user-driven automation).
- The functionality can operate independently of the app's internal control and doesn't need to follow strict app-specific protocols.
- The functionality expected to be discoverable and invoked in a uniform way across various parts of the system or other apps (e.g., an API to manipulate files or share content).

Get started with App Actions on Windows

This article describes the steps for creating app actions and describes the components of an App Action provider app. App actions are individual units of behavior that a Windows app can implement and register so that they can be accessed from other apps and experiences, seamlessly integrating into user workflows. For more information about App Actions on Windows, see [App Actions on Windows Overview](#)

The [IActionProvider](#) interface is the primary interface that app action providers use to communicate with the Windows actions framework. However, Microsoft provides the [Microsoft.AI.Actions NuGet Package](#) which automatically generates the [IActionProvider](#) implementation based on .NET attributes in your code, allowing you to make strongly typed classes to represent your actions. This is the recommended way of implementing an app action provider app and is the technique described in this article. For some edge-case scenarios, developers may want to implement [IActionProvider](#) directly. For more information, see [Manually implement IActionProvider](#).

You can also implement an app action provider using URI launch activation rather than COM activation, although some more advanced scenarios, like streaming text responses from an action, aren't supported. For more information, see [Implement URI launch for App Actions on Windows](#).

Automated dependency installation (recommended)

1. Run the command below in Terminal (whether you are a C# or C++ developer). This runs a [WinGet Configuration file](#) that performs the following tasks (dependencies already installed will be skipped):

- Enables Developer Mode.
- Installs Visual Studio Community Edition
- Include Windows App development workload and either C++ or .NET/C# Workloads
- Include MSIX Packaging tools

Console

```
winget configure https://raw.githubusercontent.com/microsoft/winget-dsc/refs/heads/main/samples/Configuration%20files/Learn%20tutorials/Windows%20AI/app_actions_cs.winget
```

Create a new Windows app project in Visual Studio

The App Actions on Windows feature is supported for multiple app frameworks, but apps must have package identity to be able to register with the system. This walkthrough will implement a Windows App Action provider in a packaged C# WinUI 3 desktop app.

1. In Visual Studio, create a new project.
2. In the **Create a new project** dialog, set the language filter to "C#" and the platform filter to "WinUI", then select the "Blank App, Packaged (WinUI 3 in Desktop)" project template.
3. Name the new project "ExampleAppActionProvider".
4. When the project loads, in **Solution Explorer** right-click the project name and select **Properties**. On the **General** page, scroll down to **Target OS** and select "Windows". For **Target OS version** and **Supported OS version**, select version 10.0.26100.0 or greater.
5. To update your project to support the Action Provider APIs, in **Solution Explorer** right-click the project name and select **Edit Project File**. Inside of **PropertyGroup**, add the following **WindowsSdkPackageVersion** element.

XML

```
<WindowsSdkPackageVersion>10.0.26100.75</WindowsSdkPackageVersion>
```

Add a reference to the Microsoft.AI.Actions NuGet package

The example in this article uses the code generation features of the Microsoft.AI.Actions NuGet package.

1. In **Solution Explorer**, right-click the project name and select **Manage NuGet Packages...**
2. Make sure you are on the **Browse** tab and search for Microsoft.AI.Actions.
3. Select Microsoft.AI.Actions and click **Install**.

Add an ActionProvider class to handle action operations

The following section shows how to implement a custom action provider class that uses .NET attributes from the [Microsoft.AI.Actions.Annotations](#) namespace to identify the components of an action. The Microsoft.AI.Actions NuGet package uses these attributes to automatically generate an underlying implementation of the [IActionProvider](#) interface. This allows you to

create strongly typed classes for actions without having to interact directly with the low-level action provider APIs.

1. In Visual Studio, right-click the `ExampleAppActionProvider` project in **Solution Explorer** and select **Add->Class**.
2. In the **Add class** dialog, name the class "MyActionProvider" and click **Add**.
3. Replace the contents of `MyActionProvider.cs` with the following code.

C#

```
using Microsoft.AI.Actions.Annotations;
using System.Threading.Tasks;
using Windows.AI.Actions;

namespace ExampleAppActionProvider
{
    [ActionProvider]
    public sealed class MyActionsProvider
    {
        [WindowsAction(
            Description = "Send a message to a contact",
            Icon = "ms-resource://Files/Assets/StoreLogo.png",
            FeedbackHandler = nameof(SendMessageFeedback),
            UsesGenerativeAI = false
        )]
        [WindowsActionInputCombination(
            Inputs = ["Contact"],
            Description = "Send message to '${Contact.Text}'"
        )]
        [WindowsActionInputCombination(
            Inputs = ["Contact", "Message"],
            Description = "Send '${Message.Text}' to '${Contact.Text}'"
        )]

        public async Task<SendMessageResult> SendMessage(
            [Entity(Name = "Contact")] string contact,
            [Entity(Name = "Message")] string? message,
            InvocationContext context)
        {
            // Your action logic here
            string result = await ProcessMessageAsync(contact, message);

            return new SendMessageResult
            {
                Text = context.EntityFactory.CreateTextEntity(result)
            };
        }

        public Task SendMessageFeedback(ActionFeedback feedback, InvocationContext context)
        {
            // Handle user feedback for the action
        }
    }
}
```

```

        return Task.CompletedTask;
    }

    public record SendMessageResult
    {
        public required TextActionEntity Text { get; init; }
    }

    public async Task<string> ProcessMessageAsync(string contact, string?
message)
    {
        if (message != null)
        {
            return await Task.Run(() => $"Processed {contact}, {message}");
        }
        else
        {
            return await Task.Run(() => $"Processed {contact}");
        }
    }
}
}

```

The code generation features of the Microsoft.AI.Actions Nuget package uses .NET attributes in your code to determine the details of the actions your app provides. This example uses the following attributes:

[] [Expand table](#)

Attribute	Description
ActionProviderAttribute	This attribute identifies a class that implements one or more actions.
WindowsActionAttribute	This attribute provides metadata about an action, such as the human-readable description of the app and an icon file that consumers of your actions can display to users.
WindowsActionInputCombinationAttribute	This attribute declares a set of input entities that an action can accept as input. A single action can support multiple combinations of inputs.
EntityAttribute	Indicates that a class represents an ActionEntity

Most of the supported attributes map directly to fields in the action definition JSON file that the system uses to discover actions. In fact, as will be shown later in this article, the Microsoft.AI.Actions code generation feature uses these attributes to automatically generate the action definition JSON file at build time. As you update your action provider class, adding or modifying these attributes, the Nuget package will regenerate the action definition file to

reflect your changes. For more information on the action definition JSON file, see [Action definition JSON schema for App Actions on Windows](#).

For a list of the supported attributes see the readme file for the [Microsoft.AI.Actions](#) Nuget package.

Update the app package manifest file

The Package.appmanifest file provides the details of the MSIX package for an app. To be registered by the system as a Windows App Action provider, the app must include a `uap3:Extension` element with the **Category** set to "windows.appExtension". This element is used to specify the location of the App Action JSON file that defines the app's actions. You must also specify `com.microsoft.windows.ai.actions` as the **Name** for the `uap3:AppExtension` element.

For more information on the action provider app package manifest format, see [Windows App Action provider package manifest XML format](#).

The example in this walkthrough uses COM activation to launch the app action provider. To enable COM activation, use the `com2:Extension` element in the app package manifest. The `invocation.clsid` value specified in the Action definition JSON file must match the class ID specified in the `com:Class` element in the app package manifest.

1. Right-click the Package.appxmanifest file and select **View Code**
2. Add the following namespaces to the **Package** element at the root of the file.

XML

```
xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
xmlns:com2="http://schemas.microsoft.com/appx/manifest/com/windows10/2"
xmlns:com3="http://schemas.microsoft.com/appx/manifest/com/windows10/3"
```

3. Add the following **Extensions** element inside the **Application** element and after the **VisualElements** element.

XML

```
<Extensions>
    <com2:Extension Category="windows.comServer">
        <com2:ComServer>
            <com3:ExeServer Executable="ExampleAppActionProvider.exe"
Displayname="ExampleAppActionProvider">
                <com:Class Id="00001111-aaaa-2222-bbbb-3333cccc4444"
Displayname="ExampleAppActionProvider" />
            </com3:ExeServer>
        </com2:ComServer>
    </com2:Extension>
```

```
<uap3:Extension Category="windows.appExtension">
    <uap3:AppExtension Name="com.microsoft.windows.ai.actions"
DisplayName="Example App Action Provider" Id="appactionprovider"
PublicFolder="Assets">
        <uap3:Properties>
            <Registration>registration.json</Registration>
        </uap3:Properties>
    </uap3:AppExtension>
</uap3:Extension>
</Extensions>
```

Implement a custom Main method

In the default project template, the **Main** method entry point is autogenerated by the compiler. This example will disable the autogenerated **Main** so that the necessary activation code can be run at startup.

1. In **Solution Explorer**, right-click the project icon and select **Edit Project File**.
2. In the **PropertyGroup** element, add the following child element to disable the auto-generated main function.

XML

```
<DefineConstants>$(DefineConstants);DISABLE_XAML_GENERATED_MAIN</DefineConstants>
```

Next, in **Solution Explorer**, right-click the project icon and select **Add->New Item**. Select **Code File**. Change the file name to "Program.cs" and click **Add**.

In the Program.cs file, we will add a line of code that will cause the actions nuget package to auto-generate the COM server activation that allows the system to invoke the action provider.

C#

```
ComServerRegisterActions.RegisterActions();
```

The rest of the code in the **Main** method in this example is just the boilerplate code to launch a WinUI app. Replace the contents of Program.cs with the following code.

C#

```
namespace ExampleAppActionProvider;

public static class Program
{
    [global::System.STAThreadAttribute]
    static void Main(string[] args)
```

```

{
    global::WinRT.ComWrappersSupport.InitializeComWrappers();
    ComServerRegisterActions.RegisterActions();
    global::Microsoft.UI.Xaml.Application.Start((p) =>
    {
        var context = new
global::Microsoft.UI.Dispatching.DispatcherQueueSynchronizationContext(global::Micro
soft.UI.Dispatching.DispatcherQueue.GetForCurrentThread());

global::System.Threading.SynchronizationContext.SetSynchronizationContext(context);
        new App();
    });
}
}

```

Add Microsoft.AI.Actions configuration properties to the project file

The code generation feature of the Microsoft.AI.Actions Nuget package uses property values defined in the project file to configure its behavior at build time. Add the following properties inside the first **PropertyGroup** element in your .csproj file.

XML

```

<GenerateActionRegistrationManifest>true</GenerateActionRegistrationManifest>
<ActionRegistrationManifest>Assets\registration.json</ActionRegistrationManifest>
<GenerateActionsWinRTComServer>true</GenerateActionsWinRTComServer>
<RootNamespace>ExampleAppActionProvider</RootNamespace>

```

The following table describes these properties.

[] Expand table

Property	Description
GenerateActionRegistrationManifest	When set to true the actions package will auto-generate an action definition JSON file based on the .NET attributes in your action provider class definition. Note that manual changes you make to the generated action definition file will be overwritten whenever you build the project. So, if you need to preserve manual changes you have made, you can set this value to false .
ActionRegistrationManifest	The package-relative path to the auto-generated action definition JSON file. Note that the system will look in the folder specified in the PublicFolder attribute of the uap3:AppExtension element in the app package manifest file. So make sure that path for this property and the public folder declared in the manifest file match.

Property	Description
GenerateActionsWinRTComServer	Set this to true to enable autogeneration of COM Server activation code from the ComServerRegisterActions.RegisterActions call in Program.cs shown previously in this article. If this value is set to false you will need to implement your own COM Server activation.
RootNamespace	Sets the root namespace of the auto-generated code so that you can access it from your own code.

Make updates based on the generated registration.json file

After building your project you can view the generated `registration.json` file in the **Assets** folder in **Solution Explorer**.

JSON

```
{
  "version": 2,
  "actions": [
    {
      "id": "ExampleAppActionProvider.MyActionsProvider.SendMessage",
      "description": "Send a message to a contact",
      "icon": "ms-resource://Files/Assets/StoreLogo.png",
      "usesGenerativeAI": false,
      "hasFeedbackHandler": true,
      "inputs": [
        {
          "name": "Contact",
          "kind": "Text"
        },
        {
          "name": "Message",
          "kind": "Text"
        }
      ],
      "inputCombinations": [
        {
          "inputs": [
            "Contact"
          ],
          "description": "Send message to '${Contact.Text}'"
        },
        {
          "inputs": [
            "Contact",
            "Message"
          ],
          "description": "Send '${Message.Text}' to '${Contact.Text}'"
        }
      ]
    }
  ]
}
```

```
        }
    ],
    "outputs": [
        {
            "name": "Text",
            "kind": "Text"
        }
    ],
    "invocation": {
        "type": "COM",
        "clsid": "11112222-bbbb-3333-cccc-4444dddd5555"
    }
}
]
}
```

Update the CLSID in the app package manifest file

The first time you build your action provider app, you will get the warning: `warning WASDK0012: The Action Provider type ExampleAppActionProvider.MyActionsProvider is not registering a ComServer with Class Id '00000000-0000-0000-0000-00000000'`. This is because the auto-generated `registration.json` file declares the `clsid` of the COM server for the action with a unique GUID. After building your project, open the `registration.json` file and note that the file declares that the action uses COM activation and specifies a `clsid` value. Replace the value of the `Id` attribute in the `com:Class` element in your app package manifest file to use the generated GUID.

For example, if the `clsid` value in the generated `registration.json` file is `11112222-bbbb-3333-cccc-4444dddd5555`, the updated `com:Class` element would look like the following:

```
<com:Class Id="11112222-bbbb-3333-cccc-4444dddd5555"
DisplayName="ExampleAppActionProvider" />
```

Allowed app invokers

A new field that has been added to the action definition JSON schema is `allowedAppInvokers` which specifies a list of Application User Model IDs (AppUserModelIDs) that can discover the action through a call to [GetActionsForInputs](#) or [GetAllActions](#). Wildcards are supported. "*" will match all AppUserModelIDs. This is recommended for most actions, unless there is a specific reason to limit the callers that can invoke an action. If `allowedAppInvokers` is omitted or is an empty list, no apps will be able to discover the action. For more information on AppUserModelIDs, see [Application User Model IDs](#).

The following example shows the recommended practice of setting `allowedAppInvokers` to allow all apps to discover the associated action.

JSON

```
"actions": [
  {
    "id": "ExampleAppActionProvider.MyActionsProvider.SendMessage",
    "description": "Send a message to a contact",
    "icon": "ms-resource://Files/Assets/StoreLogo.png",
    "usesGenerativeAI": false,
    "hasFeedbackHandler": true,
    "allowedAppInvokers" : ["*"],
  ...
}
```

ⓘ Important

In the current **Microsoft.AI.Actions** release, `allowedAppInvokers` will be overwritten whenever the action definition file is regenerated. After adding `allowedAppInvokers` to your action definition JSON file, you should set `GenerateActionRegistrationManifest` to `false` in your project file. If you modify your code and need to enable the JSON file generation again, be sure to add `allowedAppInvokers` back to the file and disable JSON file generation again.

Test the Windows App Action

The App Actions Testing Playground app allows you to validate the registration and functionality of your Windows App Action provider app. For more information on using this tool, see [App Actions Testing Playground](#).

Last updated on 09/05/2025

Implement URI launch for App Actions on Windows

This article describes the steps for creating app actions and describes the components of an App Action provider app. App actions are individual units of behavior that a Windows app can implement and register so that they can be accessed from other apps and experiences, seamlessly integrating into user workflows. For more information about App Actions on Windows, see [App Actions on Windows Overview](#).

Action provider apps can be implemented to use COM activation or URI launch activation. URI launch actions don't support some advanced action features such as displaying UI in context or streaming text results, but the implementation is very simple and may be the best choice for actions that only consist of a single request and response.

Provider apps that use COM activation implement the [IActionProvider](#) interface to handle action invocation. This method enables advanced action features such as support for streaming text, but requires more code than using the Windows App Actions VSIX Extension. For information about using COM activation in an app provider, see [Get started with App Actions on Windows](#).

Automated dependency installation (recommended)

1. Run one of the commands below in Terminal (whether you are a C# or C++ developer). This runs a [WinGet Configuration file](#) that performs the following tasks (dependencies already installed will be skipped):

- Enables Developer Mode.
- Installs Visual Studio Community Edition
- Include Windows App development workload and either C++ or .NET/C# Workloads
- Include MSIX Packaging tools

For C# developers:

Console

```
winget configure https://raw.githubusercontent.com/microsoft/winget-dsc/refs/heads/main/samples/Configuration%20files/Learn%20tutorials/Windows%20AI/app_actions_cs.winget
```

For C++ developers:

Console

```
winget configure https://raw.githubusercontent.com/microsoft/winget-dsc/refs/heads/main/samples/Configuration%20files/Learn%20tutorials/Windows%20AI/app_actions_cpp.winget
```

Create a new Windows app project in Visual Studio

The App Actions feature is supported for multiple app frameworks and languages, but apps must have package identity to be able to register with the system. This walkthrough will implement a Windows App Action provider in a packaged C# WinUI 3 desktop app.

1. In Visual Studio, create a new project.
2. In the **Create a new project** dialog, set the language filter to "C#" and the platform filter to "WinUI", then select the "WinUI Blank App (Packaged)" project template.
3. Name the new project "ExampleAppActionProvider".
4. When the project loads, in **Solution Explorer** right-click the project name and select **Properties**. On the **General** page, scroll down to **Target OS** and select "Windows". For **Target OS version** and **Supported OS version**, select version 10.0.26100.0 or greater.
5. To update the project to support the Action Provider APIs, in **Solution Explorer** right-click the project name and select **Edit Project File**. Inside of **PropertyGroup**, add the following **WindowsSdkPackageVersion** element.

XML

```
<WindowsSdkPackageVersion>10.0.26100.75</WindowsSdkPackageVersion>
```

Add a reference to the Microsoft.AI.Actions nuget package

The Microsoft.AI.Actions nuget package enables you to initialize the app actions runtime, which provides APIs for creating the entity objects that are passed as inputs to and outputs from app actions.

1. In **Solution Explorer**, right-click the project name and select **Manage NuGet Packages...**
2. Make sure you are on the **Browse** tab and search for Microsoft.AI.Actions.
3. Select Microsoft.AI.Actions and click **Install**.

Add an action definition JSON file

Action provider apps must provide an action definition file that defines the actions the app implements. This file provides information such as the unique identifier and description for each action as well as the names and types of inputs and outputs each action supports. For more information about the App Action JSON file format, see [Action definition JSON schema for Windows App Action providers](#).

This example will define one action called **SendMessage**, that takes a single **Text** entity as input, and returns a single **TextEntity** as output. In addition to defining actions, the JSON file also specifies whether the action provider app should be launched using COM activation or via URI launch. This example will use URI activation. The URI scheme `urilaunchaction-protocol` will be registered in a later step in this walkthrough.

1. In **Solution Explorer**, right-click the **Assets** folder and select **Add->New Item....**
2. In the **Add New Item** dialog, select **Text File**. Name the new file "registration.json", and click OK.
3. In **Solution Explorer**, right-click the registration.json file and select **Properties**. In the **Properties** pane, set **Build Action** to "Content" and set **Copy to Output Directory** to "Copy if Newer".
4. Add the following JSON action definition to the registration.json file.

JSON

```
{  
  "version": 3,  
  "actions": [  
    {  
      "id": "ExampleActionProvider.SendMessage",  
      "description": "Send a message (URI Launch)",  
      "icon": "ms-resource://Files/Assets/LockScreenLogo.png",  
      "usesGenerativeAI": false,  
      "allowedAppInvokers": ["*"],  
      "inputs": [  
        {  
          "name": "message",  
          "kind": "Text"  
        }  
      ],  
      "inputCombinations": [  
        {  
          "inputs": [  
            "message"  
          ],  
          "description": "Send message '${message.Text}'"  
        }  
      ],  
      "outputs": [  
        {  
          "name": "entity",  
          "kind": "TextEntity"  
        }  
      ]  
    }  
  ]  
}
```

```

    {
      "name": "response",
      "kind": "Text"
    }
  ],
  "invocation": {
    "type": "Uri",
    "uri": "urilaunchaction-protocol://",
    "inputData": {
      "message": "${message.Text}"
    }
  }
}
]
}

```

Update the app package manifest file

The Package.appmanifest file provides the details of the MSIX package for an app. To be registered by the system as a Windows App Action provider, the app must include a [uap3:Extension](#) element with the **Category** set to "windows.appExtension". This element is used to specify the location of the App Action JSON file that defines the app's actions. For more information on the action provider app package manifest format, see [App Actions on Windows package manifest XML format](#).

In order for an app action provider to be launched via URI, it must register a protocol with the system. This registration is made by providing the [com2:Extension](#) element in the app package manifest. The *Name* attribute of the **Protocol** element must match the **invocation.uri** value specified in the Action definition JSON file, which for this example is `urilaunchaction-protocol`. For more information on URI launch activation, see [Launch an app for results](#).

1. Right-click the Package.appxmanifest file and select [View Code](#)
2. Add the following namespace to the **Package** element at the root of the file.

XML

```
xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
```

3. Add the following **Extensions** element inside the **Application** element and after the **VisualElements** element.

XML

```
<Extensions>
  <uap:Extension Category="windows.protocol">
    <uap:Protocol Name="urilaunchaction-protocol" ReturnResults="always">
```

```
<!-- app-defined protocol name -->
<uap:DisplayName>URI Launch Action Scheme</uap:DisplayName>
</uap:Protocol>
</uap:Extension>
<uap3:Extension Category="windows.appExtension">
  <uap3:AppExtension Name="com.microsoft.windows.ai.actions"
  DisplayName="URILaunchAction" Id="UriLaunchActionId" PublicFolder="Assets">
    <uap3:Properties>
      <Registration xmlns="">registration.json</Registration>
    </uap3:Properties>
  </uap3:AppExtension>
</uap3:Extension>
</Extensions>
```

Handle app activation

When the app action provider is launched from its registered URI schema, the inputs for the action can be accessed through the [AppActivationArguments](#) object that is obtained by calling the [Applnstance.GetActivatedEventArgs](#). To make sure that the activation was from the registered protocol, you must first check to make sure that the value of the [Kind](#) property is [ExtendedActivationKind.ProtocolForResults](#). If so, you can cast the arguments object to a [ProtocolForResultsActivatedEventArgs](#) object.

Note

This example uses the [ProtocolForResultsActivatedEventArgs](#) object to verify that the action was invoked by Windows, and exits without completing if the action was launched by another caller. For more information see, [Detect and filter callers for App Actions on Windows](#).

The inputs for the action are accessed through the [Data](#) property of the event args, which is a [ValueSet](#) that contains key/value pairs for each input entity. This example gets the [message](#) input entity as defined in the registration.json file. The value of this input is a text string.

In order to return results for the action, the app action provider must instantiate one or more output entities. This example returns a single [TextActionEntity](#) containing the response to the input message.

To instantiate a new output entity, get an instance of the [ActionEntityFactory](#) class from the [EntityFactory](#) property of the [ActionRuntime](#) object. The factory object provides methods for instantiating all of the different types of action entities. After creating the text entity containing our response message, we make an entry in a new [ValueSet](#) where the key is the output name specified in the registration.json file, "response", and the value is the [Id](#) of the [TextActionEntity](#). Note that when you create the entity by calling [CreateTextEntity](#), the entity is

registered with the action runtime. This is why you add the **Id** of the entity as the value in the **ValueSet**. Consumers of your action will retrieve the entity object from the action runtime using this ID.

The final step in handling URI activation is to create a new [ProtocolForResultsOperation](#) and call the [ReportCompleted](#) method, passing in the **ValueSet** that contains the output entity reference.

In App.xaml.cs, replace the default implementation of **OnLaunched** with the following code.

C#

```
ProtocolForResultsOperation? _operation;
protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs
args)
{
    var eventargs =
Microsoft.Windows.AppLifecycle.AppInstance.GetCurrent().GetActivatedEventArgs();

    if ((eventargs != null) && (eventargs.Kind ==
ExtendedActivationKind.ProtocolForResults))
    {
        ProtocolForResultsActivatedEventArgs? protocolForResultsArgs =
eventargs.Data as ProtocolForResultsActivatedEventArgs;

        if
(protocolForResultsArgs.CallerPackageFamilyName.EndsWith("_cw5n1h2txyewy"))
        {
            using (ActionRuntime runtime =
ActionRuntimeFactory.CreateActionRuntime())
            {
                if (protocolForResultsArgs != null)
                {
                    ValueSet inputData = protocolForResultsArgs.Data;
                    var message = inputData["message"];

                    Windows.AI.Actions.ActionEntityFactory source =
runtime.EntityFactory;
                    Windows.AI.Actions.ActionEntity textEntity =
source.CreateTextEntity("Message response.");

                    ValueSet result = new ValueSet();
                    result["response"] = textEntity.Id;

                    _operation =
protocolForResultsArgs.ProtocolForResultsOperation;
                    _operation.ReportCompleted(result);
                }
            }
        }
    }
}
```

```
_window = new MainWindow();
_window.Activate();
}
```

Test your Windows App Action

The App Actions Testing Playground app allows you to validate the registration and functionality of your Windows App Action provider app. For more information on using this tool, see [App Actions Testing Playground app](#).

Additional App Actions on Windows features

The following articles provide information about additional features of App Actions on Windows.

- [Toggle availability of an App Action for Windows](#)

Related content

Last updated on 10/21/2025

Manually implement IActionProvider

09/05/2025

This article describes the steps for creating a provider app for App Actions on Windows by manually implementing the [IActionProvider](#) interface. Microsoft provides the [Microsoft.AI.Actions NuGet Package](#) which automatically generates the [IActionProvider](#) implementation based on .NET attributes in your code, allowing you to make strongly typed classes to represent your actions. This is the recommended way of implementing an app action provider app. The guidance in this article is provided to enable edge-case scenarios where developers want to implement [IActionProvider](#) directly. In this scenario, the Microsoft.AI.Actions Nuget package is still used to auto-generate the COM server activation code required for app action providers.

You can also implement an app action provider using URI launch activation rather than COM activation, although some more advanced scenarios, like streaming text responses from an action, aren't supported. For more information, see [Implement URI launch for App Actions on Windows](#).

App actions are individual units of behavior that a Windows app can implement and register so that they can be accessed from other apps and experiences, seamlessly integrating into user workflows. For more information about App Actions on Windows, see [App Actions on Windows Overview](#)

Automated dependency installation (recommended)

1. Run one of the commands below in Terminal (whether you are a C# or C++ developer). This runs a [WinGet Configuration file](#) that performs the following tasks (dependencies already installed will be skipped):
 - Enables Developer Mode.
 - Installs Visual Studio Community Edition
 - Include Windows App development workload and either C++ or .NET/C# Workloads
 - Include MSIX Packaging tools

For C# developers:

Console

```
winget configure https://raw.githubusercontent.com/microsoft/winget-dsc/refs/main/samples/Configuration%20files/Learn%20tutorials/Windows%20
```

```
AI/app_actions_cs.winget
```

For C++ developers:

Console

```
winget configure https://raw.githubusercontent.com/microsoft/winget-dsc/refs/heads/main/samples/Configuration%20files/Learn%20tutorials/Windows%20AI/app_actions_cpp.winget
```

Create a new Windows app project in Visual Studio

The App Actions on Windows feature is supported for multiple app frameworks, but apps must have package identity to be able to register with the system. This walkthrough will implement a Windows App Action provider in a packaged C# WinUI 3 desktop app.

1. In Visual Studio, create a new project.
2. In the **Create a new project** dialog, set the language filter to "C#" and the platform filter to "WinUI", then select the "Blank App, Packaged (WinUI 3 in Desktop)" project template.
3. Name the new project "ExampleAppActionProvider".
4. When the project loads, in **Solution Explorer** right-click the project name and select **Properties**. On the **General** page, scroll down to **Target OS** and select "Windows". For **Target OS version** and **Supported OS version**, select version 10.0.26100.0 or greater.
5. To update your project to support the Action Provider APIs, in **Solution Explorer** right-click the project name and select **Edit Project File**. Inside of **PropertyGroup**, add the following **WindowsSdkPackageVersion** element.

XML

```
<WindowsSdkPackageVersion>10.0.26100.75</WindowsSdkPackageVersion>
```

Add an action definition JSON file

Action provider apps must provide an action definition file that defines the actions the app implements. This file provides information about the inputs and outputs of your actions and metadata such as a unique identifier and a description for your actions. For more information about the App Action JSON file format, see [Action definition JSON schema for Windows App Action providers](#).

This example will define one action called **SendMessage**, that takes a single **Text** entity as input, and returns a single **TextEntity** as output. In addition to defining actions, the JSON file also specifies whether the action provider app should be launched using COM activation or via URI launch. This example will use COM activation.

1. In **Solution Explorer**, right-click the Assets folder and select **Add->New Item....**
2. In the **Add New Item** dialog, select **Text File**. Name the new file "registration.json", and click OK.
3. Add the JSON action definition shown in the code example below to the registration.json file.
4. In **Solution Explorer**, right-click the registration.json file and select **Properties**. In the **Properties** pane, set **Build Action** to "Content" and set **Copy to Output Directory** to "Copy if Newer".
5. Replace the **invocation.clid** value with a new GUID that will identify the provider. You can generate a new GUID in Visual Studio by going to **Tools->Create GUID**. This GUID will be used again in a few different places in this walkthrough.

JSON

```
{  
  "version": 2,  
  "actions": [  
    {  
      "id": "ExampleAppActionProvider.SendMessage",  
      "description": "Send a message",  
      "icon": "ms-resource://Files/Assets/StoreLogo.png",  
      "allowedAppInvokers" : ["*"],  
      "usesGenerativeAI": false,  
      "inputs": [  
        {  
          "name": "message",  
          "kind": "Text"  
        }  
      ],  
      "inputCombinations": [  
        {  
          "inputs": [  
            "message"  
          ],  
          "description": "Send message '${message.Text}'"  
        }  
      ],  
      "outputs": [  
        {  
          "name": "response",  
          "kind": "Text"  
        }  
      ],  
      "invocation": {  
        "comId": "ExampleAppActionProvider",  
        "method": "SendMessage",  
        "parameters": [  
          {  
            "name": "message",  
            "value": "${message.Text}"  
          }  
        ]  
      }  
    }  
  ]  
}
```

```
        "type": "COM",
        "clsid": "00001111-aaaa-2222-bbbb-3333cccc4444"
    }
}
]
```

Add an ActionProvider class to handle action operations

This example in this article manually implements the [IActionProvider](#) interface. This interface requires the implementation of a single method, [InvokeAsync](#), which the system uses to invoke an action.

1. In **Solution Explorer**, right-click the project icon and select **Add->Class**.
2. In the **Add class** dialog, name the class "MyAppActionProvider" and click **Add**.
3. In the generated ActionProvider.cs file, update the class definition to indicate that it implements the [IActionProvider](#) interface.
4. Label the class with the [System.Runtime.InteropServices.GuidAttribute](#). This is used by the COM activation code shown later in this walkthrough. Be sure to update the value to the value specified in the **invocation.clid** field in the registration.json file.
5. Update the class declaration to indicate that inherits the [IActionProvider](#) interface.

C#

```
// MyAppActionProvider.cs
[System.Runtime.InteropServices.GuidAttribute("00001111-aaaa-2222-bbbb-
3333cccc4444")]
public partial class MyAppActionProvider : IActionProvider
```

Implement IActionProvider.InvokeAsync

The [InvokeAsync](#) method has a return type of [IAsyncAction](#). This example uses a helper class that returns a [Task](#), which is then converted to an [IAsyncAction](#) with a call to [AsAsyncAction](#) extension method. Add the following method definition to the [MyAppActionProvider](#) class.

C#

```
// MyAppActionProvider.cs
public IAsyncAction InvokeAsync(ActionInvocationContext context)
{
```

```
        return InvokeAsyncHelper(context).AsAsyncAction();
    }
```

In the helper method, `InvokeAsyncHelper`, the following actions are performed:

- `ActionInvocationContext.GetInputEntities` is called to retrieve the set of entities that are being passed as input into the action.
- An action provider may support multiple actions, so before processing the input values, the `ActionInvocationContext.ActionId` property is evaluated to determine which action is being invoked. The ID will be the value declared for the action in the `id` field in the `registration.json` file.
- In this example, there is a single input entity of type `TextActionEntity` named "message". The helper method loops through the inputs and checks for the expected name.
- When the expected input entity name is found, it is cast to the `TextEntity` type, and the message text is retrieved using the `Text` property. At this point, a real-world implementation of an action would take this input message, do some processing on it, and generate a response.
- This example creates a response `TextEntity`, as specified in the `outputs` field in the `registration.json` file. The entity is created from a hard-coded string and then added as an output by calling to `SetOutputEntity`, passing in the output entity name and the `TextEntity` object.

Add the following method definition to the `MyAppActionProvider` class.

C#

```
// MyAppActionProvider.cs
async Task InvokeAsyncHelper(ActionInvocationContext context)
{
    NamedActionEntity[] inputs = context.GetInputEntities();

    var actionId = context.ActionId;
    switch (actionId)
    {
        case "ExampleAppActionProvider.SendMessage":
            foreach (NamedActionEntity inputEntity in inputs)
            {
                if (inputEntity.Name.Equals("message", StringComparison.OrdinalIgnoreCase))
                {

                    TextActionEntity entity = (TextActionEntity)(inputEntity.Entity);
                    string message = entity.Text;

                    // TODO: Process the message and generate a response

                    string response = "This is the message response";
                    TextActionEntity result =
context.EntityFactory.CreateTextEntity(response);
```

```
        context.SetOutputEntity("response", result);
    }
break;

default:
break;

}
}
```

Update the app package manifest file

The Package.appmanifest file provides the details of the MSIX package for an app. To be registered by the system as a Windows App Action provider, the app must include a [uap3:Extension](#) element with the **Category** set to "windows.appExtension". This element is used to specify the location of the App Action JSON file that defines the app's actions. For more information on the action provider app package manifest format, see [Windows App Action provider package manifest XML format](#).

To enable COM activation, use the [com2:Extension](#) element in the app package manifest. The class ID specified in the [com:Class](#) element in the app package manifest must match the [invocation.clid](#) value specified in the Action definition JSON file.

1. Right-click the Package.appxmanifest file and select [View Code](#)
2. Add the following namespaces to the **Package** element at the root of the file.

XML

```
xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
xmlns:com2="http://schemas.microsoft.com/appx/manifest/com/windows10/2"
xmlns:com3="http://schemas.microsoft.com/appx/manifest/com/windows10/3"
```

3. Add the following **Extensions** element inside the **Application** element and after the **VisualElements** element. For information about the action provider manifest extensions, see [App Actions on Windows package manifest XML format](#). Note that the GUID specified as the **Id** for the [com:Class](#) element will be updated in a later step.

XML

```
<Extensions>
<com2:Extension Category="windows.comServer">
<com2:ComServer>
<com3:ExeServer Executable="ExampleAppActionProvider.exe"
```

```

    DisplayName="ExampleAppActionProvider">
        <com:Class Id="00001111-aaaa-2222-bbbb-3333cccc4444">
    DisplayName="ExampleAppActionProvider" />
        </com3:ExeServer>
    </com2:ComServer>
</com2:Extension>
<uap3:Extension Category="windows.appExtension">
    <uap3:AppExtension Name="com.microsoft.windows.ai.actions">
DisplayName="Example App Action Provider" Id="exampleappactionprovider"
PublicFolder="Assets">
    <uap3:Properties>
        <Registration xmlns="">registration.json</Registration>
    </uap3:Properties>
</uap3:AppExtension>
</uap3:Extension>
</Extensions>

```

4. Update the **Id** attribute of the **com:Class** element to the GUID you created in a previous step.

Implement a custom Main method

In the default project template, the **Main** method entry point is autogenerated by the compiler. This example will disable the autogenerated **Main** so that the necessary activation code can be run at startup.

1. In **Solution Explorer**, right-click the project icon and select **Edit Project File**.
2. In the **PropertyGroup** element, add the following child element to disable the autogenerated main function.

XML

```
<DefineConstants>$(DefineConstants);DISABLE_XAML_GENERATED_MAIN</DefineConstants>
```

3. In **Solution Explorer**, right-click the project icon and select **Add->New Item**.
- 4.

Add a reference to the Microsoft.AI.Actions nuget package

Although this example shows a manual implementation of **IActionProvider**, we will still use the Microsoft.AI.Actions nuget package to autogenerate the COM server activation code needed for the OS to launch the action provider.

1. In **Solution Explorer**, right-click the project name and select **Manage NuGet Packages...**
2. Make sure you are on the **Browse** tab and search for Microsoft.AI.Actions.

3. Select Microsoft.AI.Actions and click Install.

Next, in **Solution Explorer**, right-click the project icon and select **Add->New Item**. Select **Code File**. Change the file name to "Program.cs" and click **Add**.

In the Program.cs file, we will call two helper methods that auto-generate the COM server activation that allows the system to invoke the action provider.

```
C#  
  
Microsoft.AI.Actions.Helpers.ActionRuntimeFactory.CreateActionRuntime();  
  
Microsoft.AI.Actions.Helpers.ActionProviderFactory.RegisterActionProvider<MyAppActionProvider,  
    Windows.AI.Actions.Provider.IActionProvider>(new MyAppActionProvider());
```

The rest of the code in the **Main** method in this example is just the boilerplate code to launch a WinUI app. Replace the contents of Program.cs with the following code.

```
C#  
  
namespace ExampleAppActionProvider;  
  
static void Main(string[] args)  
{  
    Microsoft.AI.Actions.Helpers.ActionRuntimeFactory.CreateActionRuntime();  
  
    Microsoft.AI.Actions.Helpers.ActionProviderFactory.RegisterActionProvider<MyAppActionProvider,  
        Windows.AI.Actions.Provider.IActionProvider>(new MyAppActionProvider());  
    global::Microsoft.UI.Xaml.Application.Start((p) =>  
    {  
        var context = new  
global::Microsoft.UI.Dispatching.DispatcherQueueSynchronizationContext(global::Microsoft.UI.Dispatching.DispatcherQueue.GetForCurrentThread());  
  
        global::System.Threading.SynchronizationContext.SetSynchronizationContext(context)  
        ;  
        new App();  
    });  
}
```

Test the Windows App Action

The App Actions Testing Playground app allows you to validate the registration and functionality of your Windows App Action provider app. For more information on using this tool, see [App Actions Testing Playground](#).

Detect and filter callers for App Actions on Windows

10/21/2025

This article describes several mechanisms that provider apps for App Actions on Windows can implement to limit the set of apps that can query for or invoke an action. This is an optional implementation step that is designed to support scenarios such as actions that consume resources that have per-use costs, such as subscription-based LLM services.

Set allowedAppInvokers in the action definition JSON file

The action definition JSON file, which is used to register an action with the system, includes an *allowedAppInvokers* field for each action definition. The value of this field is a list of Application User Model IDs (AUMID) that can discover the action through a call to [GetActionsForInputs](#) or [GetAllActions](#). Wildcards are supported. "*" will match all AppUserModelIDs. If *allowedAppInvokers* is omitted or is an empty list, no apps will be able to discover the action. For more information on AppUserModelIDs, see [Application User Model IDs](#). For details of the action definition JSON file, see [Action definition JSON schema for App Actions on Windows](#).

Note that the *allowedAppInvokers* field only limits which apps can query for an action. Because of the ways in which action providers are registered with the system, it's still possible for apps to invoke an action, even if they are restricted from querying by *allowedAppInvokers*. The rest of this article discusses different ways of restricting the callers for an action at runtime.

Detect the caller when using URI launch activation

The following section describes different ways of detecting the caller of an action at runtime for action provider apps that use URI launch activation.

Get caller Package Family Name (Rich activation only)

Provider apps that use modern, rich activation can check the value of the [CallerPackageFamilyName](#) property of the [ProtocolForResultsActivatedEventArgs](#) class passed into your provider app on launch. If the value ends with the string "_cw5n1h2txyewy", then the action was invoked by Windows. For more information on rich activation, see [Rich activation with the app lifecycle API](#).

```
const string windowsPFN = "_cw5n1h2txyewy";
```

C#

```
protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    var eventargs = AppInstance.GetCurrent().GetActivatedEventArgs();

    if ((eventargs != null) && (eventargs.Kind ==
ExtendedActivationKind.ProtocolForResults))
    {

        ProtocolForResultsActivatedEventArgs? protocolForResultsArgs =
eventargs.Data as ProtocolForResultsActivatedEventArgs;

        if (protocolForResultsArgs.CallerPackageFamilyName.EndsWith(windowsPFN))
        {
            // Perform the action if invoked by Windows
        }
    }

    _window = new MainWindow();
    _window.Activate();
}
```

Use the `$.Token` URI query string parameter

Action providers that use URI activation declare the URI that launches their action in the action definition JSON manifest file. There is a special query string parameter `$.Token` that instructs the Action Runtime to pass a unique identifier token on the query string of the launch URI when your app is invoked. The following example shows the syntax for declaring a URI with the `$.Token` parameter.

JSON

```
"invocation": {
    "type": "Uri",
    "uri": "urilaunchaction-protocol://messageaction?token=${$.Token}",
    "inputData": {
        "message": "${message.Text}"
    }
}
```

When your action is launched, retrieve the query string parameter `Token` from the activation URI. Pass this token into a call to [GetActionInvocationContextFromToken](#). The Action Runtime will validate the token against the token it provided when launching the action provider. If the values match, a valid [ActionInvocationContext](#) object is returned, validating that the action was invoked by the Action Runtime.

C#

```
protected override void OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    var eventargs = AppInstance.GetCurrent().GetActivatedEventArgs();

    if ((eventargs != null) && (eventargs.Kind ==
ExtendedActivationKind.ProtocolForResults))
    {

        ProtocolForResultsActivatedEventArgs? protocolForResultsArgs =
eventargs.Data as ProtocolForResultsActivatedEventArgs;

        using (ActionRuntime runtime = ActionRuntimeFactory.CreateActionRuntime())
        {

            var launchUri = protocolForResultsArgs.Uri;
            var query = HttpUtility.ParseQueryString(launchUri.Query);
            var token = query["Token"];
            if(token != null)
            {
                var context = runtime.GetActionInvocationContextFromToken(token);
                if(context == null)
                {
                    // Caller is not Action Runtime
                    return;
                }
            }
        }
    }
}
```

Note that this example uses helper methods provided by the Microsoft.AI.Actions NuGet package. For information on referencing this package from an action provider, see [Implement URI launch for App Actions on Windows](#).

Detect the caller at runtime from COM activation

The following section walks through the process of retrieving the AUMID of the app that invoked an action at runtime. The examples in this section will check for the AUMID of the Action Runtime. Apps could choose to filter on the AUMID of other apps or use the Package Family Name (PFN) to filter on all apps that are contained in a particular package.

Add a reference to the CsWin32 Nuget package

The example code shown in this section uses Win32 APIs to get the package family name of the caller. In order to more easily call Win32 APIs from a C# app, this example will use the CsWin32 NuGet package, which generates C# wrappers for Win32 APIs at build time. For more information, see the [CsWin32 github repo](#).

1. In **Solution Explorer**, right-click on your project and select **Manage NuGet Packages....**
2. On the **Browse** tab, search for "Microsoft.Windows.CsWin32".
3. Select this package and click **Install**, following all of the prompts.

When you install this package, it will show you a readme detailing how to specify the Win32 APIs for which C# wrappers will be generated. To do so, you will need to create a text file that lists the methods you will be using.

1. In **Solution Explorer**, right-click on your project and select **Add->New Item....**
2. Select the **TextFile** template.
3. Name the file "NativeMethods.txt" and click **Add**
4. Double-click NativeMethods.txt to open the file and add the following lines.

text

```
CoRegisterClassObject  
CoRevokeClassObject  
CoImpersonateClient  
OpenThreadToken  
GetCurrentThread  
CoRevertToSelf  
GetPackageFamilyNameFromToken  
GetLastError
```

Update your ActionProvider class to check the action invoker PFN

The following helper method uses the Win32 APIs specified in NativeMethods.txt to retrieve the AUMID for the calling process and returns true if it matches the AUMID that is passed in. For detailed explanation about how this code works, see [Impersonating a Client](#).

C#

```
static bool WasInvokedByAUMID(string aumid)  
{  
    Windows.Win32.Foundation HRESULT hr = PInvoke.CoImpersonateClient();  
  
    var RPC_E_CALL_COMPLETE = unchecked((int)0x80010117);
```

```

    if (hr.Value == RPC_E_CALL_COMPLETE)
    {
        return false;
    }

    Microsoft.Win32.SafeHandles.SafeFileHandle hThreadTok;

    bool bRes = PInvoke.OpenThreadToken(
        new Microsoft.Win32.SafeHandles.SafeFileHandle(PInvoke.GetCurrentThread(),
true),
        Windows.Win32.Security.TOKEN_ACCESS_MASK.TOKEN_QUERY,
        true,
        out hThreadTok
    );

    if (bRes == false)
    {
        var e = Marshal.GetLastWin32Error();
        Console.WriteLine("Unable to read thread token. " + e);
        hThreadTok.Close();
        return false;
    }

    PInvoke.CoRevertToSelf();

    var invokerAumid = new Span<char>();
    uint aumidLength = 0;

    PInvoke.GetApplicationUserModelIdFromToken(hThreadTok, ref aumidLength,
invokerAumid);
    invokerAumid = new char[aumidLength].AsSpan();
    PInvoke.GetApplicationUserModelIdFromToken(hThreadTok, ref aumidLength,
invokerAumid);

    hThreadTok.Close();

    if (invokerAumid.Trim('\0').EndsWith(aumid))
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

Because of the way that the [ColImpersonateClient](#) is implemented, it's important that you call the helper method from within the invocation of your action and not from within your app's launch procedure.

The following example illustrates a check for the Action Runtime AUMID in an action implemented using the Microsoft.AI.Actions code generation framework.

C#

```
const string actionRuntimeAUMID = "_cw5n1h2txyewy!ActionRuntime";
```

C#

```
[WindowsActionInputCombination(
    Inputs = ["Contact", "Message"],
    Description = "Send '${Message.Text}' to '${Contact.Text}'"
)]
public async Task<SendMessageResult> SendMessage(
    [Entity(Name = "Contact")] string contact,
    [Entity(Name = "Message")] string? message,
    InvocationContext context)
{
    if (!WasInvokedByAUMID(actionRuntimeAUMID))
    {
        context.Result = ActionInvocationResult.Unavailable;
        return new SendMessageResult
        {
            Text = context.EntityFactory.CreateTextEntity("")
        };
    }

    // Your action logic here
    string result = await ProcessMessageAsync(contact, message);

    return new SendMessageResult
    {
        Text = context.EntityFactory.CreateTextEntity(result)
    };
}
```

Return streaming text with App Actions on Windows

09/16/2025

This article describes how to implement an app action on Windows that incrementally streams text as output back to the caller. This feature is useful for scenarios that leverage LLMs, which typically respond with a series of tokens that incrementally build the reply.

Return streaming text using Microsoft.AI.Actions code generation

This section describes how to implement an action that returns streaming text using the code generation capabilities provided by the [Microsoft.AI.Actions](#) Nuget package. This feature allows you to create strongly typed classes and methods to implement actions, using .NET attributes to trigger the auto-generation of the underlying action infrastructure code at build time. For information on configuring a project to use code generation, see [Get started with App Actions on Windows](#). This section builds on concepts and techniques introduced in [Get started with App Actions on Windows](#). It is strongly recommended that you review the content in that article before continuing as it walks through setting up your project to implement an action.

The code examples in this section should all be added to your action provider class, which is labeled with the [ActionProviderAttribute](#).

```
C#  
  
[ActionProvider]  
public sealed class MyActionsProvider
```

Actions are implemented in a class method that is labeled with the [WindowsActionAttribute](#), which provides metadata about the actions such as a description and an icon. The [WindowsActionInputCombinationAttribute](#) describes the inputs that are supported. For this example, the action will take a single text entity as an input. For all of the actions code generation attributes, you can look at the API reference documentation to see all of the supported properties.

The return value from our action function is a .NET **record** type that contains a single field containing a [StreamingTextEntityWriter](#). This class, provided by the [Microsoft.AI.Actions.Annotations](#) namespace allows you to provide a callback function through which you will return the incremental text updates.

C#

```
[WindowsAction(Description = "Get a streaming response", Icon = "ms-resource://Files/Assets/LockScreenLogo.png", UsesGenerativeAI = false)]
[WindowsActionInputCombination(Inputs = ["message"], Description = "Get a
streaming response for: '${message.Text}'")]
public GetStreamingResponse GetStreamingResponseAction(string message,
InvocationContext context)
{
    return new GetStreamingResponse
    {
        StreamingText = new StreamingTextEntityWriter(
            ActionEntityTextFormat.Plain,
            (textWriter) => GetStreamingTextAsync(textWriter, message))
    };
}

public record GetStreamingResponse
{
    public required StreamingTextEntityWriter StreamingText { get; init; }
}
```

For simplicity, in this example the streaming text callback from our action will return a string constructed incrementally from a hard-coded list of strings. The [SetText](#) method of the [StreamingTextActionEntityWriter](#) passed into the callback is called as each token is processed, passing the accumulated text each time. In this example a delay is used to simulate asynchronous responses from an LLM or web service.

Sometimes an LLM will "backtrack", replacing previously returned text with new text. In this example, we return the finalized string at the end of the method call, illustrating that you can replace the text submitted with previous calls to [SetText](#).

C#

```
static string[] exampleStreamingText = new string[]
{ "This", "is", "example", "streaming", "text" };

private async Task GetStreamingTextAsync(StreamingTextActionEntityWriter
textWriter, string message)
{
    var stringBuilder = new StringBuilder();

    foreach (string token in exampleStreamingText)
    {
        stringBuilder.Append(token);
        textWriter.SetText(stringBuilder.ToString());
        await Task.Delay(500);
    }

    // This call replaces the content submitted in the previous calls to SetText
```

```
// This is useful for handling when an LLM "backtracks".
textWriter.SetText("This is the finalized example streaming text.");
}
```

When you build your solution, the code generation feature will generate the following action registration JSON file.

JSON

```
{
  "version": 2,
  "actions": [
    {
      "id": "ExampleAppActionProvider.MyActionsProvider.GetStreamingResponseAction",
      "description": "Get a streaming response",
      "icon": "ms-resource://Files/Assets/LockScreenLogo.png",
      "usesGenerativeAI": false,
      "inputs": [
        {
          "name": "message",
          "kind": "Text"
        }
      ],
      "inputCombinations": [
        {
          "inputs": [
            "message"
          ],
          "description": "Get a streaming response for: '${message.Text}'"
        }
      ],
      "outputs": [
        {
          "name": "StreamingText",
          "kind": "StreamingText"
        }
      ],
      "invocation": {
        "type": "COM",
        "clsid": "11112222-bbbb-3333-cccc-4444dddd5555"
      }
    }
  ]
}
```

ⓘ Important

With the current release the Microsoft.AI.Actions code generation, you need to manually add the **allowedAppInvokers** field to action definition JSON to specify a list of

AppUserModelIDs that specifies the apps that can query for your actions through a call to [GetActionsForInputs](#). It is recommended that apps specify the wildcard "*" to make actions visible to all apps. For more information, see [Action definition JSON schema for App Actions on Windows](#).

Use `IAsyncEnumerable` to return streaming text

The code generation framework allows you to return an `IAsyncEnumerable` instead of using a `StreamingTextActionEntityWriter`. For some scenarios, this approach may be easier and faster to develop, although it doesn't support specifying a text format or replacing previously returned text.

For this technique, the record returned by our `GetStreamingResponse` helper method should be an `IAsyncEnumerable` of type `string`.

```
C#  
  
public record GetStreamingResponse  
{  
    public required IAsyncEnumerable<string> StreamingText { get; init; }  
}
```

The method that implements our action will simply return a new instance of the record type.

```
C#  
  
[WindowsAction(Description = "Get a streaming response", Icon = "ms-  
resource://Files/Assets/LockScreenLogo.png", UsesGenerativeAI = false)]  
[WindowsActionInputCombination(Inputs = ["message"], Description = "Get a  
streaming response for: '${message.Text}'")]  
public GetStreamingResponse GetStreamingResponseAction(string message,  
InvocationContext context)  
{  
    return new GetStreamingResponse  
    {  
        StreamingText = GetStreamingTextAsync(message)  
    };  
}
```

Finally, our placeholder method for returning the streaming text tokens should return an `IAsyncEnumerable`. Use the `yield` statement to return each incremental string token.

```
C#  
  
static string[] exampleStreamingText = new string[]  
{ "This", "is", "example", "streaming", "text" };
```

```
private async IAsyncEnumerable<string> GetStreamingTextAsync(string message)
{
    foreach (string token in exampleStreamingText)
    {
        yield return token;
        await Task.Delay(500);
    }
}
```

Streaming text with **IActionProvider**

This section describes how to build the same action that was shown in the previous section, but for apps that choose to manually implement the [IActionProvider](#) interface instead of using the automatic code generation capabilities provided by the Microsoft.AI.Actions Nuget package. This section builds on concepts and techniques introduced in [Manually implement IActionProvider](#). It is strongly recommended that you review the content in that article before continuing as it walks through setting up your project to implement an action.

Using **IActionProvider** directly means that you need to manually create your action definition JSON file. The following example JSON file defines an action that takes a single text entity as an input and returns a streaming text entity as an output. It's important to note the **id** field for the action since this will be used in our action provider code to determine which action is being called.

JSON

```
{
  "version": 3,
  "actions": [
    {
      "id": "ExampleActionProvider.MyActionProvider.GetStreamingResponseAction",
      "description": "Get a streaming response",
      "icon": "ms-resource://Files/Assets/LockScreenLogo.png",
      "usesGenerativeAI": false,
      "allowedAppInvokers": ["*"],
      "inputs": [
        {
          "name": "message",
          "kind": "Text"
        }
      ],
      "inputCombinations": [
        {
          "inputs": [
            "message"
          ],
        }
      ]
    }
  ]
}
```

```

        "description": "Get a streaming response for: '${message.Text}'"
    }
],
"outputs": [
{
    "name": "StreamingText",
    "kind": "StreamingText"
}
],
"invocation": {
    "type": "COM",
    "clsid": "11112222-bbbb-3333-cccc-4444dddd5555"
}
}
]
}

```

As described in [Manually implement IActionProvider](#) the system invokes actions by calling the [InvokeAsync](#) method of your **IActionProvider** implementation. This example uses a helper class that returns a [Task](#), which is then converted to an **IAsyncAction** with a call to [AsAsyncAction](#) extension method.

In the [InvokeAsyncHelper](#) method, we check the ID of the invoked action to make sure it is [ExampleActionProvider.MyActionProvider.GetStreamingResponseAction](#). If so, we get the text entity input from the [ActionInvocationContext](#) object passed in by the system. Next we initialize a new [StreamingTextActionEntityWriter](#) object using the factory method [ActionEntityFactory.CreateStreamingTextActionEntityWriter](#). Then we run an asynchronous task to begin sending streaming text updates back to the caller. Finally, using the [ActionInvocationContext](#) object, we set the result of the task to [ActionInvocationResult.Success](#) and set the output entity to the [StreamingTextActionEntity](#) returned by the [StreamingTextActionEntityWriter.ReaderEntity](#) property.

C#

```

public IAsyncAction InvokeAsync(ActionInvocationContext context)
{
    return InvokeAsyncHelper(context).AsAsyncAction();
}

async Task InvokeAsyncHelper(ActionInvocationContext context)
{
    NamedActionEntity[] inputs = context.GetInputEntities();

    var actionId = context.ActionId;
    switch (actionId)
    {
        case "ExampleActionProvider.MyActionProvider.GetStreamingResponseAction":

```

```

        TextActionEntity entity = (TextActionEntity)(inputs.First().Entity);
        string message = entity.Text;

        var streamingTextWriter =
context.EntityFactory.CreateStreamingTextActionEntityWriter(ActionEntityTextFormat
.Plain);

        _ = Task.Run(async () => await
GetStreamingTextAsync(streamingTextWriter, message));

        context.Result = ActionInvocationResult.Success;
        context.SetOutputEntity("StreamingText",
streamingTextWriter.ReaderEntity);

        break;
default:
        break;

    }

}

```

As in the code generation example in the previous section, the streaming text callback from our action will use tokens from a hard-coded list of strings to simulate an incremental response from an LLM. As each string in the list is processed, the accumulated string is passed into the `SetText` method of the `StreamingTextActionEntityWriter`. In this example a delay is used to simulate asynchronous responses from an LLM or web service. At the end of the method, the finalized string is passed to `SetText` to illustrate the scenario where an LLM backtracks and replaces previously returned text.

C#

```

static string[] exampleStreamingText = new string[]
{ "This", "is", "example", "streaming", "text" };

private async Task GetStreamingTextAsync(StreamingTextActionEntityWriter
textWriter, string message)
{
    var stringBuilder = new StringBuilder();

    foreach (string token in exampleStreamingText)
    {
        stringBuilder.Append(token);
        textWriter.SetText(stringBuilder.ToString());
        await Task.Delay(1000);
    }

    textWriter.SetText("This is the finalized example streaming text.");
}

```

Handle remote files with App Actions on Windows

10/24/2025

This article describes how to implement support for remote files that are hosted by a cloud service provider from an App Actions on Windows provider app. The **RemoteFile** entity type, represented in C# by the [RemoteFileActionEntity](#) class, provides properties that are used by many cloud file storage providers, such as unique IDs for files and drives. The examples in this page show how to access the cloud file properties but do not actually retrieve or process the remote files, as these processes will vary between different apps and providers. For more information on the different entity types that can be used as inputs and outputs for actions, see [Action definition JSON schema for App Actions on Windows](#).

Process remote files using Microsoft.AI.Actions code generation

The following example shows how to implement remote file support in an action provider that uses the [Microsoft.AI.Actions NuGet Package](#) which automatically generates the **IActionProvider** implementation based on .NET attributes in your code, allowing you to make strongly typed classes to represent your actions. For information about creating an action provider app using the **Microsoft.AI.Actions NuGet package, see [Get started with App Actions on Windows](#).

Note that a **Where** clause is used in the [WindowsActionInputCombination](#) attribute to specify the cloud providers that are supported by this action. It's always a good idea to use where clauses like this to prevent your action from being presented to users for scenarios that your action doesn't support.

C#

```
[ActionProvider]
public sealed class MyActionsProvider
{
    [WindowsAction(Description = "Summarize a remote file", Icon = "ms-
resource://Files/Assets/LockScreenLogo.png", UsesGenerativeAI = false)]
    [WindowsActionInputCombination(Description = "Summarize
${RemoteFileToSummarize}",
        Inputs = ["RemoteFileToSummarize"],
        Where = ["${RemoteFileToSummarize.SourceId} == 'Contoso.Cloud' ||
${RemoteFileToSummarize.SourceId} == 'Fabrikam.Cloud'"])]
    public async Task<SummarizeRemoteFileResult> SummarizeRemoteFile([Entity(Name =
"RemoteFileToSummarize")] RemoteFileActionEntity remoteFile, InvocationContext
```

```

context)
{
    var summary = "";
    var remoteFileContents = "";

    switch (remoteFile.SourceId)
    {
        case "Contoso.Cloud":
            // Retrieve the file contents using the properties used by the
            "Contoso" service
            remoteFileContents = GetContosoCloudFile(remoteFile.AccountId,
remoteFile.DriveId, remoteFile.FileId);
            summary = SummarizeFile(remoteFileContents);
            break;

        case "Fabrikam.Cloud":
            // Retrieve the file contents using the properties used by the
            "Fabrikam" service
            remoteFileContents = GetFabrikamCloudFile(remoteFile.AccountId,
remoteFile.SourceUri);
            summary = SummarizeFile(remoteFileContents);
            break;
    }

    var entityFactory = context.EntityFactory;

    return new SummarizeRemoteFileResult
    {
        Text = entityFactory.CreateTextEntity(summary)
    };
}

public record SummarizeRemoteFileResult
{
    public required TextActionEntity Text { get; init; }
}

```

JSON action definition with remote file input

The code example below shows an action definition JSON file that declares an action that uses the **RemoteFile** entity. If you are using the **Microsoft.AI.Actions** NuGet to auto-generate your JSON action definition file, this is what the file will look like after you have implemented the action as shown in the previous section.

JSON

```
{
    "version": 3,
    "actions": [

```

```

{
  "id": "ExampleAppActionProvider.MyActionsProvider.SummarizeRemoteFile",
  "description": "Summarize a remote file",
  "icon": "ms-resource://Files/Assets/LockScreenLogo.png",
  "usesGenerativeAI": false,
  "allowedAppInvokers" : ["*"],
  "inputs": [
    {
      "name": "RemoteFileToSummarize",
      "kind": "RemoteFile"
    }
  ],
  "inputCombinations": [
    {
      "inputs": [
        "RemoteFileToSummarize"
      ],
      "where": [
        "${RemoteFileToSummarize.SourceId} == 'Contoso.Cloud' || ${RemoteFileToSummarize.SourceId} == 'Fabrikam.Cloud'"
      ],
      "description": "Summarize ${RemoteFileToSummarize}"
    }
  ],
  "outputs": [
    {
      "name": "Text",
      "kind": "Text"
    }
  ],
  "invocation": {
    "type": "COM",
    "clsid": "00001111-aaaa-2222-bbbb-3333cccc4444"
  }
}
]
}

```

Process remote files with **IActionProvider**

This section shows how to implement an action that accepts a **RemoteFile** entity as an input when you are manually implementing the **IActionProvider** interface. You will also need to manually add an action definition JSON file like the one shown in the previous section to your project. For more information on implementing a COM-based action provider without using the **Microsoft.AI.Actions** code generation feature, see [Manually implement **IActionProvider**](#).

C#

```

public IAsyncAction InvokeAsync(ActionInvocationContext context)
{

```

```

        return InvokeAsyncHelper(context).AsAsyncAction();
    }

async Task InvokeAsyncHelper(ActionInvocationContext context)
{
    NamedActionEntity[] inputs = context.GetInputEntities();

    var actionId = context.ActionId;
    switch (actionId)
    {
        case "ExampleAppActionProvider.MyActionsProvider.SummarizeRemoteFile":
            var summary = "";
            var remoteFileContents = "";

            foreach (NamedActionEntity inputEntity in inputs)
            {
                if (inputEntity.Name.Equals("RemoteFileToSummarize",
StringComparison.OrdinalIgnoreCase))
                {
                    var remoteFile = (RemoteFileActionEntity)inputEntity.Entity;
                    switch (remoteFile.SourceId)
                    {
                        case "Contoso.Cloud":
                            // Retrieve the file contents using the properties
used by the "Contoso" service
                            remoteFileContents =
GetContosoCloudFile(remoteFile.AccountId, remoteFile.DriveId, remoteFile.FileId);
                            summary = SummarizeFile(remoteFileContents);
                            break;

                        case "Fabrikam.Cloud":
                            // Retrieve the file contents using the properties
used by the "Fabrikam" service
                            remoteFileContents =
GetFabrikamCloudFile(remoteFile.AccountId, remoteFile.SourceUri);
                            summary = SummarizeFile(remoteFileContents);
                            break;
                    }
                }
            }

            TextActionEntity result =
context.EntityFactory.CreateTextEntity(summary);
            context.SetOutputEntity("Text", result);

            break;

        default:
            break;
    }
}

```


Toggle availability of an App Action for Windows

Article • 05/19/2025

A Windows App Action provider can specify that one or more of its actions are currently unavailable. This feature enables scenarios such as requiring a login or subscription before an action is made available to the user.

Set initial availability

You can specify the initial availability status of an app action by providing a value for the *isAvailable* field in the Action definition JSON file. The value is optional and defaults to true. The following example illustrates the usage of the *isAvailable* field to make an app action unavailable immediately after installation.

JSON

```
"version": 2,  
"actions": [  
  {  
    "id": "ToDoList.ToDoActionHandler.AddToList",  
    "description": "Add item to your to-do list",  
    "icon": "ms-resource://Files/Assets/LockScreenLogo.png",  
    "usesGenerativeAI": false,  
    "isAvailable": false,  
    ...  
  }]
```

For more information, see [Action definition JSON schema for App Actions on Windows](#).

Change the availability state at runtime

Register a change in availability state of one or more registered actions with the system by calling [ActionRuntime.SetActionAvailability](#).

C#

```
void SetActionAvailability(bool actionIsAvailable)  
{  
  
    using (ActionRuntime runtime = new ActionRuntime())  
    {  
        runtime.SetActionAvailability("ExampleActionProvider.SendMessage",  
        actionIsAvailable);  
    }  
}
```


Position App Action UI relative to the invoking app

This article describes how a provider app for App Actions on Windows can determine the location of the app window that has invoked an action. This allows the provider to place any UI that supports the action near to the calling window, to provide a more seamless user experience.

For more information on implementing a provider app for App Actions on Windows, see [Get started with App Actions on Windows](#).

Declare the `displaysUI` JSON property

The action definition JSON file supports the `displaysUI` property for each action defined in the file. Set this value to `true`, or omit the value to use the default value of `true`, to provide a hint to the invoking app that the action may display UI elements as part of its workflow. For more information about the format of this file, see [Action definition JSON schema for App Actions on Windows](#).

Get the location of the invoker app window

When an action is launched, the `ActionInvocationContext.InvokerWindowId` property contains the `Windows.UI.WindowId` associated with the app window that invoked the action, if it was provided by the invoker. It is optional for action invokers to specify a Window ID, which is done by calling `ActionRuntime.CreateInvocationContextWithWindowId`. The Window ID will be non-zero if it is provided by the invoker app.

For more information about Windows and Window management, see [Manage app windows](#).
For more information about invoking actions, see [Discover and invoke registered App Actions on Windows](#).

The following example shows how to retrieve the `InvokerWindowId` and convert it from a `Windows.UI.WindowId` to a `Microsoft.UI.WindowId`, which is needed to retrieve a WinUI Window. If the invoker Window ID is valid, the example gets the position of the invoker Window and moves the action provider app window to that position.

C#

```
// Convert Windows.UI.WindowId to Microsoft.UI.WindowId by creating a new WindowId
// with the same Value
Microsoft.UI.WindowId invokerWindowIdCast = new Microsoft.UI.WindowId { Value =
context.InvokerWindowId.Value };
```

```
// Get the invoker AppWindow
AppWindow callerWindow = AppWindow.GetFromWindowId(invokerWindowIdCast);

if (callerWindow != null)
{
    // Get the caller window's position
    PointInt32 callerPosition = callerWindow.Position;

    // Move the app Window to the caller window's position
    Window.Current.AppWindow.Move(callerPosition);
}
```

Last updated on 01/13/2026

Action definition JSON schema for App Actions on Windows

This article describes the format of the action definition JSON file format for App Actions on Windows. This file must be included in your project with the **Build Action** set to "Content" and **Copy to Output Directory** set to "Copy if newer". Specify the package-relative path to the JSON file in your package manifest XML file. For more information, see [Action provider package manifest XML format](#).

Example action definition JSON file

JSON

```
"version": 3,
"actions": [
  {
    "id": "Contoso.SampleGreeting",
    "description": "Send greeting with Contoso",
    "icon": "ms-resource://...",
    "displaysUI": false,
    "usesGenerativeAI": false,
    "isAvailable": false,
    "allowedAppInvokers": ["*"],
    "inputs": [
      {
        "name": "UserFriendlyName",
        "kind": "Text"
      },
      {
        "name": "PetName",
        "kind": "Text",
        "required": false
      }
    ],
    "inputCombinations": [
      {
        "inputs": ["UserFriendlyName"],
        "description": "Greet ${UserFriendlyName.Text}"
      },
      {
        "inputs": ["UserFriendlyName", "PetName"],
        "description": "Greet ${UserFriendlyName.Text} and their pet ${PetName.Text}"
      }
    ],
    "contentAgeRating": "child",
    "invocation": {
      {
        "name": "UserFriendlyName"
      }
    }
  }
]
```

```

        "type": "Uri",
        "uri": "contoso:/greetUser?
userName=${UserFriendlyName.Text}&petName=${PetName.Text}",
    },
    "where": [
        "${UserFriendlyName.Length > 3}"
    ]
}
},
{
    "id": "Contoso.SampleGetText",
    "description": "Summarize file with Contoso",
    "icon": "ms-resource://...",
    "inputs": [
        {
            "name": "FileToSummarize",
            "kind": "File"
        }
    ],
    "inputCombinations": [
        {
            "inputs": ["FileToSummarize"],
            "description": "Summarize ${FileToSummarize.Path}"
        },
    ],
    "outputs": [
        {
            "name": "Summary",
            "kind": "Text"
        }
    ],
    "contentAgeRating": "child",
    "invocation": {
        "type": "COM",
        "clsid": "{...}"
    }
}
]
}

```

Action definition JSON properties

The tables below describe the properties of the action definition JSON file.

The **Version** field indicates the schema version in which the property was introduced.

The **PWA** field indicates support for action providers that are implemented as a Progressive Web App (PWA). For more information on app actions with PWAs, see [Enable App Actions on Windows for a PWA](#).

Document root

[\[+\] Expand table](#)

Property	Type	Description	Required	Version
version	string	Schema version. When new functionality added, the version increments by one.	Yes.	2
actions	Action[]	Defines the actions provided by the app.	Yes.	2

Action

[\[+\] Expand table](#)

Property	Type	Description	Required	Version	PWA
id	string	Action identifier. Must be unique per app package. This value is not localizable.	Yes	2	Yes
description	string	User-facing description for this action. This value is localizable.	Yes	2	Yes
icon	string	Localizable icon for the action. This value is an <i>ms-resource</i> string for an icon deployed with the app.	No	2	Yes
allowedAppInvokers	string[]	Specifies a list of Application User Model IDs (AppUserModelIDs) that can discover the action through a call to GetActionsForInputs or GetAllActions . Wildcards are supported. "*" will match all AppUserModelIDs. This is recommended for most actions, unless there is a specific reason to limit the callers that can invoke an action. If allowedAppInvokers is omitted or is an empty list, no apps will be able to discover the action. For more information on	No	3	Yes

Property	Type	Description	Required	Version	PWA
		AppUserModelIDs, see Application User Model IDs .			
displaysUI	Boolean	Specifies whether the action might display UI. The default value is true.	No	2	Yes
usesGenerativeAI	Boolean	Specifies whether the action uses generative AI. The default value is false.	No	2	Yes
isAvailable	Boolean	Specifies whether the action is available for use upon installation. The default value is true.	Yes	2	Yes
inputs	Inputs[]	List of entities that this action accepts as input.	Yes	2	Yes
inputCombinations	InputCombination[]	Provides descriptions for different combinations of inputs.	Yes	2	Yes
outputs	Output[]	If specified, must be an empty string in the current release.	No	2	Yes
invocation	Invocation	Provides information about how the action is invoked.	Yes	2	Yes
contentAgeRating	string	A field name from the UserAgeConsentGroup that specifies the appropriate age rating for the action. The allowed values are "Child", "Minor", "Adult". If no value is specified, the default behavior allows access to all ages.	No	2	Yes

Output

[] [Expand table](#)

Property	Type	Description	Required	Version	PWA
name	string	The variable name of the entity. This value is not localizable.	Yes	2	Yes

Property	Type	Description	Required	Version	PWA
kind	string	A field name from the <code>ActionEntityKind</code> enumeration specifying the entity type. This value is not localizable. The allowed values are "None", "Document", "File", "Photo", "Text", "StreamingText", "RemoteFile", "Table", "Contact".	Yes	2	Yes

The schema version and PWA support vary for the different values of the `kind` property. See the entry for each entity type for details.

InputCombination

[+] Expand table

Property	Type	Description	Required	Version	PWA
inputs	string[]	A list of Input names for an action invocation. The list may be empty.	Yes	2	Yes
description	string	Description for the action invocation. This value is localizable.	No	2	Yes
where	string[]	One or more conditional statements determining the conditions under which the action applies.	No	2	Yes

Invocation

[+] Expand table

Property	Type	Description	Required	Version	PWA
type	string	The instantiation type for the action. The allowed values are "uri" and "com".	Yes	2	Only "uri".
uri	string	The absolute URI for launching the action. Entity usage can be included within the string. A special reserved query string parameter, <code>token=\${\$.Token}</code> can be added to the URI to allow actions to validate that they have been invoked by the Action Runtime. For more information, see Use the <code>\$.Token</code> URI query string parameter .	Yes, for URI instantiated actions.	2	Yes
clsid	string	The class ID for the COM class that implements <code>IActionProvider</code> .	Yes, for COM actions.	2	No

Property	Type	Description	Required	Version	PWA
inputData	JSON name/value pairs	A list of name/value pairs specifying additional data for URI actions.	No. Only valid for URI actions.	2	Yes

ActionEntityKind enumeration

The **ActionEntityKind** enumeration specifies the types of entities that are supported by App Actions on Windows. In the context of a JSON action definition, the entity kinds are string literals that are case-sensitive.

 Expand table

Entity kind string	Description	Version	PWA
"File"	Includes all file types that are not supported by photo or document entity types.	2	Input only.
"Photo"	Image file types. Supported image file extensions are ".jpg", ".jpeg", and ".png"	2	Input only.
"Document"	Document file types. Supported document file extensions are ".doc", ".docx", ".pdf", ".txt"	2	Input only.
"Text"	Supports strings of text.	2	Input only.
"StreamingText"	Supports incrementally streamed strings of text.	2	False
"RemoteFile"	Supports metadata to enable actions to validate and retrieve backing files from a cloud service.	2	Input only.
"Table"	A 2D table of string values that is serialized to a 1 dimensional array of strings.	3	No.
"Contact"	A set of data representing a contact.	3	Input only.

Entity properties

Each entity type supports one or more properties that provide instance data for the entity. Entity property names are case sensitive.

The following example illustrates how entities are referenced in the query string for actions that are launched via URI activation:

```
...?param1=${entityName.property1}&param2=${entityName.property2}
```

For information on using entity properties to create conditional sections in the action definition JSON, see [Where clauses](#).

File entity properties

[] [Expand table](#)

Property	Type	Description	Version	PWA
"FileName"	string	The name of the file.	2	Input only.
"Path"	string	The path of the file.	2	Input only.
"Extension"	string	The extension of the file.	2	Input only.

Document entity properties

The *Document* entity supports the same properties as *File*.

Photo entity properties

The *Photo* entity supports all of the properties of *File* in addition to the following properties.

[] [Expand table](#)

Property	Type	Description	Version	PWA
"IsTemporaryPath"	Boolean	A value specifying whether the photo is stored in a temporary path. For example, this property is true for photos that are stored in memory from a bitmap, not stored permanently in a file.	2	Input only.

Text entity properties

[] [Expand table](#)

Property	Type	Description	Version	PWA
"Text"	string	The full text.	2	Input only.
"ShortText"	string	A shortened version of the text, suitable for UI display.	2	Input only.
"Title"	string	The title of the text.	2	Input only.
"Description"	string	A description of the text.	2	Input only.
"Length"	double	The length of the text in characters.	2	Input only.
"WordCount"	double	The number of words in the text.	2	Input only.

StreamingText entity properties

[] Expand table

Property	Type	Description	Version	PWA
"TextFormat"	string	The format of the streaming text. Supported values are "Plain", "Markdown".	2	No

RemoteFile entity properties

[] Expand table

Property	Type	Description	Version	PWA
"AccountId"	string	The identifier of the cloud service account associated with the remote file.	2	Input only.
"ContentType"	string	The MIME type of the remote file.	2	Input only.
"DriveId"	string	The identifier for the remote drive associated with the remote file.	2	Input only.
"Extension"	string	The extension of the remote file.	2	Input only.
"FileId"	string	The identifier of the remote file.	2	Input only.
"FileKind"	RemoteFileKind	The remote file kind.	2	Input only.

Property	Type	Description	Version	PWA
"SourceId"	string	The identifier of the cloud service that hosts the remote file.	2	Input only.
"SourceUri"	string	The URI of the remote file.	2	Input only.

RemoteFileKind enumeration

The **RemoteFileKind** enumeration specifies the types of files that are supported for the **RemoteFile** entity.

[Expand table](#)

Entity kind	Description	Version
string		
"File"	Includes all file types that are not supported by photo or document entity types.	2
"Photo"	Image file types. Supported image file extensions are ".jpg", ".jpeg", and ".png"	2
"Document"	Document file types. Supported document file extensions are ".doc", ".docx", ".pdf", ".txt"	2

Table entity properties

[Expand table](#)

Property	Type	Description	Required	Version	PWA
"RowCount"	integer	The number of rows in the table.	Yes	3	No
"ColumnCount"	integer	The number of columns in the table.	Yes	3	No
"Title"	string	The title of the table.	No	3	No
"Description"	string	The description of the table.	No	3	No

Contact entity properties

[Expand table](#)

Property	Type	Description	Required	Version	PWA
"Email"	string	The email address of the contact.	No	3	Input only.
"FullName"	integer	The full name of the contact.	No	3	Input only.
"Title"	string	The title of the contact.	No	3	Input only.
"Description"	string	The description of the contact.	No	3	Input only.

Where clauses

The action definition JSON format supports *where* clauses that can be used to implement conditional logic, such as specifying that an action should be invoked only when an entity property has a specified value.

The following operators can be used with *where* clauses.

[Expand table](#)

Operator	Description
==	Equality
~=	Case-insensitive equality
!=	Inequality
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
	Logical OR
&&	Logical AND

Where clauses use the following format:

JSON

```
"where": [
    "${<property_accessor>} <operator> <value>"
]
```

The following example shows a *where* clause that evaluates to true if a **File** entity has the file extension ".txt".

JSON

```
"where": [  
    "${File.Extension} ~= \".txt\""  
]
```

Multiple *where* clauses can be combined using the logical AND and logical OR operators.

JSON

```
"where": [  
    "${File.Extension} ~= \".txt\" || ${File.Extension} ~= \".md\""  
]
```

Related articles

Last updated on 01/13/2026

App Actions on Windows package manifest XML format

Article • 05/19/2025

This article describes the package manifest XML format for App Actions on Windows.

App extension

The app package manifest file supports many different extensions and features for Windows apps. The app package manifest format is defined by a set of schemas that are documented in the [Package manifest schema reference](#). Action providers declare their registration information within the [uap3:AppExtension](#). The **Name** attribute of the extension must be set to "com.microsoft.windows.ai.actions".

Action providers should include the [uap3:Properties](#) as the child of [uap3:AppExtension](#). The package manifest schema does not enforce the structure of the [uap3:Properties](#) element other than requiring well-formed XML.

Action providers must provide a **Registration** element which specifies the path to the action definition JSON file. For more information, see [Action JSON schema for Windows Copilot Action providers](#).

XML

```
<uap3:Extension Category="windows.appExtension">
    <uap3:AppExtension
        Name="com.microsoft.windows.ai.actions"
        DisplayName="..."
        Id="..."
        PublicFolder="Assets">
        <uap3:Properties>
            <Registration>path\to\registration.json</Registration> <!-- path relative
            to the PublicFolder above -->
        </uap3:Properties>
    </uap3:AppExtension>
</uap3:Extension>
```

Additional requirements

Both COM and URI-launched action providers must have package identity. Package identity is declared in the app package manifest file using the [Identity](#) element. For more information, see [An overview of Package Identity in Windows apps](#).

COM-based action providers must be *full trust apps* which have an integrity level of *mediumIL*.

This is declared in the app package manifest file by setting the [*uap10:TrustLevel](#) attribute to "mediumIL".

URI-launched action providers must also have a trust level of *mediumIL*. If a URI-launched action provider will return outputs, the app must implement the ability to be launched for results. For more information, see [Launch an app for results](#). URI-launched action providers that return outputs must also instantiate the runtime.

App Actions Testing Playground app

The App Actions Testing Playground app allows developers to validate that their actions are being successfully registered with the system and to test the functionality of their actions. For information on how to build a Windows App Action provider app, see [Get started with Windows App Actions](#).

This section will walk you through how to use the testing tool when you are ready to test your actions. Before testing, deploy your app so that your actions are registered with the system.

Important

The `allowedAppInvokers` field in the action definition JSON file specifies a list of Application User Model IDs (AppUserModelIDs) that can discover the action through a call to [GetActionsForInputs](#) or [GetAllActions](#). Before beginning testing, make sure to set the `allowedAppInvokers` value to "*" which matches all AppUserModelIDs, allowing the App Actions Testing Playground app to invoke your action. For more information, see [Action definition JSON schema for App Actions on Windows](#).

1. Download and install the [App Actions Testing Playground app](#).
2. Launch the Windows App Actions Testing Playground app.
3. On the **Action catalog** tab, click on the entry for your action.
4. If you registered an action that supports different sets of inputs and outputs, select the version you want to test from the **Overloads** drop-down..
5. Under **Inputs**, select an entity to use as an input to your action. You can add custom entities to use when testing by clicking on the **Add an entity** on the **Action catalog** tab.
6. Click the **Run Action** button.
7. You will see your app launch.
8. In the App Actions Testing Playground app, a modal dialog will launch showing the response from your action.

Viewing your action registrations

On the **Registrations** tab of the App Actions Testing Playground app, you can view the JSON registration for all registered actions. This allows you to quickly view the details of your actions.

Discover and invoke registered App Actions on Windows

09/02/2025

This article shows how to discover and invoke registered App Actions on Windows. This allows devs to incorporate actions provided by other apps into their user experience. For information on creating apps that implement app actions, see [Get started with App Actions on Windows](#).

Create a new Windows app project in Visual Studio

The App Actions feature is supported for multiple app frameworks and languages, but apps must have package identity to be able to register with the system. This walkthrough will implement a Windows App Action provider in a packaged C# WinUI 3 desktop app.

1. In Visual Studio, create a new project.
2. In the **Create a new project** dialog, set the language filter to "C#" and the platform filter to "WinUI", then select the "WinUI Blank App (Packaged)" project template.
3. Name the new project "ExampleAppActionProvider".
4. When the project loads, in **Solution Explorer** right-click the project name and select **Properties**. On the **General** page, scroll down to **Target OS** and select "Windows". For **Target OS version** and **Supported OS version**, select version 10.0.26100.0 or greater.
5. To update the project to support the Action Provider APIs, in **Solution Explorer** right-click the project name and select **Edit Project File**. Inside of **PropertyGroup**, add the following **WindowsSdkPackageVersion** element.

XML

```
<WindowsSdkPackageVersion>10.0.26100.75</WindowsSdkPackageVersion>
```

Add a reference to the Microsoft.AI.Actions nuget package

The Microsoft.AI.Actions nuget package enables you to initialize the app actions runtime, which provides APIs for creating the entity objects that are passed as inputs to and outputs from app actions.

1. In Solution Explorer, right-click the project name and select Manage NuGet Packages...
2. Make sure you are on the Browse tab and search for Microsoft.AI.Actions.
3. Select Microsoft.AI.Actions and click Install.

Query for available app actions based on a set of inputs

The [Windows.AI.Actions.Hosting](#) namespace provides APIs for querying for the app actions that are currently registered with the system. The [ActionCatalog.GetActionsForInputs](#) method allows you to query for actions that accept a specified set of input entities. In a typical action hosting scenario, an app that works with a particular type of content will query for actions that consume that type of content, show the available actions to the user, and then, if the user selects an action, the app will invoke that action on the user's behalf.

This article will show an example implementation of the action consumer scenario. First, we will create some simple UI to display the available actions. The [ListBox](#) in this example is bound to a list of [ActionInstance](#) objects that will be declared in a code example later in this article. To provide a human-readable description of each action, we bind the [ListBoxItem](#) content to the [DisplayInfo.Description](#) property of each [ActionInstance](#).

XAML

```
<ListBox Name="LBActionsList" ItemsSource="{x:Bind actionInstances}"
SelectionChanged="LBActionsList_SelectionChanged">
    <ListBox.ItemTemplate>
        <DataTemplate x:DataType="actions:ActionInstance">
            <ListBoxItem Content="{x:Bind DisplayInfo.Description}"
Background="LightBlue"/>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

In the code behind page, after declaring the list of [ActionInstance](#) objects, we add a helper method for querying for the app actions that take a particular set of inputs. This example will query for actions that take a single photo as an input. We create a [FileOpenPicker](#) to allow the user to pick an image file. Next we instantiate the [ActionRuntimeFactory](#) class, which is provided by the Microsoft.AI.Actions nuget package, by calling [CreateActionRuntime](#). The action runtime is needed in order to create instances of [ActionEntity](#) and its child classes that represent the various types of inputs an action can consume.

We call [CreatePhotoEntity](#), passing in the path to the user-selected image file, to create a [PhotoActionEntity](#) object. And then we create an array of [ActionEntity](#) objects containing the

PhotoEntity. Actions can accept multiple combinations of different inputs, so you can add multiple entities of different types to the array.

Finally, we pass the **ActionEntity** array into **ActionCatalog.GetActionsForInputs** and assign the result to the **List** that is bound to our UI, allowing the user to select an action based on its description.

C#

```
List<ActionInstance>? actionInstances;
private async void GetActionsForInputs()
{
    FileOpenPicker fileOpenPicker = new FileOpenPicker();
    var hwnd = WinRT.Interop.WindowNative.GetWindowHandle(this);
    WinRT.Interop.InitializeWithWindow.Initialize(fileOpenPicker, hwnd);
    fileOpenPicker.FileTypeFilter.Add(".png");
    var imageFile = await fileOpenPicker.PickSingleFileAsync();

    if(imageFile != null)
    {
        using (ActionRuntime runtime =
ActionRuntimeFactory.CreateActionRuntime())
        {
            var photoEntity =
runtime.EntityFactory.CreatePhotoEntity(imageFile.Path);
            var inputEntities = new ActionEntity[] { photoEntity };
            actionInstances =
runtime.ActionCatalog.GetActionsForInputs(inputEntities).ToList();
            this.Bindings.Update();
        }
    }
}
```

When we are ready to initiate the selected action, we retrieve the **ActionInstance** from the **ListBox** and call **InvokeAsync** to invoke the action. When the invocation is complete, get the output from the action by using the **Context** property of the **ActionInstance** to get an instance of **ActionInvocationContext** and then call **GetOutputEntities** to get the list of output entities. In this example, we expect the first output to be a **PhotoActionEntity**, so we validate that the output exists and is of kind **ActionEntityKind.Photo**, and then access the path of the returned photo.

C#

```
private void LBActionsList_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (LBActionsList.SelectedItem != null)
    {
        var actionInstance = (ActionInstance)LBActionsList.SelectedItem;
```

```
await actionInstance.InvokeAsync();
var outputs = actionInstance.Context.GetOutputEntities();

if (outputs.Length > 0 && outputs.First().Entity.Kind ==
ActionEntityKind.Photo)
{
    var outputPhotoEntity = (PhotoActionEntity)outputs.First().Entity;
    var outputPhotoPath = outputPhotoEntity.FullPath;
    // Do something with the output photo
}
}
```

Responsible AI and safety for App Actions on Windows

Article • 05/19/2025

Learn about the Responsible AI and safety considerations when implementing App Actions on Windows.

Responsible AI

When building AI backed actions, it is your responsibility as the action author to perform content moderation and abuse monitoring when it comes to entities returned to the user. For more information about Microsoft Responsible AI policies for more information, see [Microsoft Responsible AI: Principles and approach](#).

Content Age Rating

When designing app actions, consider whether the content and functionality is appropriate for users of all ages. In the action definition JSON file, action providers can use the *contentAgeRating* property to specify the appropriate ages for each action. The value of this property is a member name from the [UserAgeConsentGroup](#) enumeration. The supported values are "Child", "Minor", "Adult". If no value is specified, the default behavior allows access to all ages. For more information on the action definition JSON file format, see [Action definition JSON schema for App Actions on Windows](#).

Agent Launchers on Windows overview

Agent Launchers on Windows provides a standardized way for apps to register AI agents and make them discoverable across the system. This enables users to access agents from any supporting experience, such as from the Start menu, search, or within applications, without needing to know which app provides each agent.

What is an Agent Launcher?

An Agent Launcher is a registered entry point for an AI agent on Windows. Without Agent Launchers, each experience would need custom integration code for every agent—whether through Model Context Protocol (MCP), App Actions, or proprietary APIs. Agent Launchers solve this by providing a unified registration and discovery mechanism where apps register their agents once, making them available to all supporting experiences.

What is an agent?

In the context of Agent Launchers, agents are AI-powered assistants designed for active, ongoing conversations that help users accomplish complex tasks. They are more than chatbots or one-off request processors:

- **Interactive and conversational:** Engage in multi-turn dialogues, asking clarifying questions and providing contextual responses
- **Task-oriented:** Help users complete specific goals, from planning trips to analyzing data to creating content
- **Contextually aware:** Understand and maintain context throughout conversations, remembering previous interactions
- **Action-capable:** Take actions on behalf of users and integrate with app functionality to get things done
- **Visible and accessible:** Open a user interface where users can actively interact, see progress, and guide their work

Agent Launchers are designed for agents that provide interactive experiences where users and AI collaborate, not for background services or silent automation.

Benefits of using Agent Launchers?

For users

- **Unified discovery:** Find all available agents from any supporting experience without remembering which app contains which agent.
- **Seamless integration:** Access agents from different contexts, including from the Start menu, search, or within other applications.
- **Consistent experience:** Interact with agents through consistent, familiar patterns regardless of the provider.

For developers

- **Single integration:** Register your agent once and make it available to all supporting experiences.
- **Flexible deployment:** Register agents statically at install time or dynamically at runtime based on authentication, subscriptions, or other conditions.
- **Ecosystem reach:** Leverage the standardized App Actions framework to tap into a growing ecosystem.

For experiences and platforms

- **Easy discovery:** Query the On-Device Registry (ODR) to find all registered agents on the system.
- **Reliable invocation:** Launch agents through a standardized mechanism with well-defined inputs.
- **No custom integrations:** Support all agents without app-specific code.

How Agent Launchers work

Agent Launchers are built on the Windows App Actions framework. An Agent Launcher consists of:

- **Agent definition manifest:** A JSON file with metadata including display name, description, unique identifier, and the app action to invoke
- **App extension declaration:** An entry in the app package manifest that registers the agent with Windows
- **App Action with required entities:** An App Action with required `agentName` and `prompt` inputs, plus optional entities like `attachedFile`

Agents are registered with and retrieved through the On-Device Registry (ODR) via the `odr.exe` command-line tool. Registration can be static (at install time) or dynamic (at runtime). When invoked, the system locates the associated App Action and launches it with the user's prompt and context, opening the agent's interface for interaction.

Get started

To learn how to create an Agent Launcher for your Windows app, see [Get started with Agent Launchers on Windows](#).

For detailed information about the agent definition JSON schema, see [Agent definition JSON schema](#).

Last updated on 12/12/2025

Get started with Agent Launchers on Windows

This article describes the steps for creating Agent Launchers and the components of an Agent Launcher provider app. Agent Launchers on Windows enable a Windows app to implement and register agents so that other apps and experiences can invoke them. For more information, see [Agent Launchers on Windows Overview](#).

Implement an action provider

Agent Launchers rely on a specialized implementation of an App Actions on Windows provider app. For this tutorial, start by implementing an action provider by following the guidance in [Get started with App Actions on Windows](#). The rest of this article shows the additions and modifications you need to make to the app action to register and call it as an Agent Launcher. It demonstrates how to register the agent both statically (registered at install time) and dynamically so that you can add and remove Agent Launchers per application logic.

Modify the action provider to support the required input entities

For an app action to be invoked as an Agent Launcher, it must meet the following requirements for input entities:

- One input must be a [TextActionEntity](#) with the name `agentName`.
- One input must be a [TextActionEntity](#) with the name `prompt`.
- All input combinations for the action must accept both the `agentName` and `prompt` entities.
- Invoking the App Action should open an application where the user can actively interact with the agent, not just complete work in the background.

C#

```
using Microsoft.AI.Actions.Annotations;
using System.Threading.Tasks;
using Windows.AI.Actions;

namespace ContosoAgentProvider
{
    [ActionProvider]
```

```

public sealed class ContosoAgentActionsProvider
{
    [WindowsAction(
        Id = "ContosoAgentAction",
        Description = "Start an agent for Contoso",
        Icon = "ms-resource://Files/Assets/ContosoLogo.png",
        UsesGenerativeAI = true
    )]
    [WindowsActionInputCombination(
        Inputs = ["agentName", "prompt"],
        Description = "Start Contoso Agent with '${agentName.Text}'."
    )]
    [WindowsActionInputCombination(
        Inputs = ["agentName", "prompt", "attachedFile"],
        Description = "Start Contoso Agent with '${agentName.Text}' and
additional context."
    )]

    public async Task StartContosoAgent(
        [Entity(Name = "agentName")] string agentName,
        [Entity(Name = "prompt")] string prompt,
        [Entity(Name = "attachedFile")] FileActionEntity? attachedFile,
        InvocationContext context)
    {
        // Your agent invocation logic here
        await InvokeAgentAsync(agentName, prompt, attachedFile);
    }

    public async Task InvokeAgentAsync(string agentName, string prompt,
FileActionEntity? attachedFile)
    {
        // Process the agent invocation with the provided inputs
        if (attachedFile != null)
        {
            await Task.Run(() => $"Starting agent '{agentName}' with prompt
'{prompt}' and file context");
        }
        else
        {
            await Task.Run(() => $"Starting agent '{agentName}' with prompt
'{prompt}'");
        }
    }
}

```

The following example shows the action definition file generated from the action provider class definition shown in the previous example.

JSON

```
{
    "version": 3,
```

```

"actions": [
{
  "id": "ContosoAgentAction",
  "description": "Start an agent for Contoso",
  "icon": "ms-resource://Files/Assets/ContosoLogo.png",
  "usesGenerativeAI": true,
  "allowedAppInvokers": ["*"],
  "inputs": [
    {
      "name": "agentName", "kind": "Text"
    },
    {
      "name": "prompt", "kind": "Text"
    },
    {
      "name": "attachedFile", "kind": "File"
    }
  ],
  "inputCombinations": [
    // If we don't always expect attachedFile to be present, we can
    // declare two different inputCombinations
    {
      "inputs": [ "agentName", "prompt" ],
      "description": "Start Contoso Agent with '${agentName.Text}'."
    },
    {
      "inputs": [ "agentName", "prompt", "attachedFile" ],
      "description": "Start Contoso Agent with '${agentName.Text}' and
additional context."
    }
  ],
  "outputs": [],
  "invocation": {
    "type": "Uri",
    // Example of a valid launch URI using the inputs defined above
    "uri": "contosoLaunch:invokeAgent?
agentName=${agentName.Text}&prompt=${prompt.Text}&context=${attachedFile.Path}"
  }
}
]
}

```

Test your App Action

Before registering your action as an Agent Launcher, verify that your App Action works correctly. Follow the testing guidance in the [Get started with App Actions on Windows](#) article to ensure your action:

1. **Registers successfully** - Verify that your action appears in the action catalog.
2. **Accepts the required inputs** - Test that your action can receive the `agentName` and `prompt` text entities.

3. **Invokes correctly** - Confirm that your action logic executes when invoked.
4. **Handles optional inputs** - If you have optional inputs like `attachedFile`, test both with and without them.

You can test your App Action by using the Windows.AI.Actions APIs or the the [App Actions Testing Playground app](#). Once you've confirmed your App Action works as expected, you can proceed to register it as an agent launcher.

Create an agent definition JSON file

Create a new JSON file in your project's `Assets` folder (or your preferred location) to define your agent registration. The agent definition links your agent to the App Action you created in the previous step.

The value of the `action_id` field in the agent definition manifest must match the `id` field specified in the action definition manifest for an action included in the same app package.

JSON

```
{  
  "manifest_version": "0.1.0",  
  "version": "1.0.0",  
  "name": "Contoso.ContosoAgent",  
  "display_name": "ms-resource://contosoAgentDisplayName",  
  "description": "ms-resource://contosoAgentDescription",  
  "icon": "ms-resource://Files/Assets/ContosoLogo.png",  
  "action_id": "ContosoAgentAction"  
}
```

Set the JSON file to **Copy to Output Directory** in your project properties:

- **For C# projects:** Right-click the JSON file in Solution Explorer, select **Properties**, and set **Copy to Output Directory** to **Copy if newer** or **Copy always**.
- **For C++ projects:** Add the following code to your project file:

XML

```
<Content Include="Assets\agentRegistration.json">  
  <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>  
</Content>
```

Static registration via app package manifest

Agent Launchers can register statically (at install time) or dynamically (at runtime). This section covers static registration.

The Package.appxmanifest file provides the details of the MSIX package for an app. If you followed the get started tutorial for app actions, you already added a `uap3:Extension` element to register the action by setting the extension **Name** attribute to

`com.microsoft.windows.ai.actions`. To register the action as an Agent Launcher, you need to add another app extension with the **Name** set to `com.microsoft.windows.ai.appAgent`. Under the **Properties** element of the app extension element, you must provide a **Registration** element that specifies the location of your agent definition JSON file.

 **Note**

Each statically registered Agent Launcher should have its own AppExtension entry.

XML

```
<uap3:Extension Category="windows.appExtension">
    <uap3:AppExtension
        Name="com.microsoft.windows.ai.appAgent"
        Id="ContosoAgent"
        DisplayName="Contoso Agent"
        PublicFolder="Assets">
        <uap3:Properties>
            <Registration>agentRegistration.json</Registration>
        </uap3:Properties>
    </uap3:AppExtension>
</uap3:Extension>
```

Dynamic registration via On Device Registry (ODR)

In addition to static registration, you can register Agent Launchers dynamically at runtime by using the Windows On Device Registry (ODR). This method is useful when you need to add or remove agents based on application logic.

Add an Agent Launcher dynamically

Use the `odr add-app-agent` command to register an Agent Launcher dynamically. This command takes the path to your agent registration JSON file.

C#

```

using System.Diagnostics;

// Create process start info
ProcessStartInfo startInfo = new ProcessStartInfo
{
    FileName = "odr.exe",
    Arguments = $"add-app-agent \\<path to agentDefinition.json>\\""",
    UseShellExecute = false,
    RedirectStandardOutput = true,
    RedirectStandardError = true,
    CreateNoWindow = true
};

// Start the ODR process
using Process process = new Process { StartInfo = startInfo };
process.Start();

// Read the output
string output = await process.StandardOutput.ReadToEndAsync();
string error = await process.StandardError.ReadToEndAsync();

await process.WaitForExitAsync();

```

The command returns a JSON response with the following structure:

JSON

```
{
    "success": true,
    "agent": {
        "id": "ContosoAgent_cw5n1h2txyewy_Contoso.ContosoAgent",
        "version": "1.0.0",
        "name": "Contoso.ContosoAgent",
        "display_name": "Contoso Agent",
        "description": "Description for Contoso agent.",
        "icon": "C://pathToContosoIcon.png",
        "package_family_name": "ContosoPackageFamilyName",
        "action_id": "ContosoAgentAction"
    },
    "message": "Agent registered successfully"
}
```

Remove an Agent Launcher dynamically

Use the `odr remove-app-agent` command to remove a dynamically registered Agent Launcher. You can only remove agents that the same package adds dynamically.

C#

```

using System.Diagnostics;

// Create process start info
ProcessStartInfo startInfo = new ProcessStartInfo
{
    FileName = "odr.exe",
    Arguments = $"remove-app-agent
\"ContosoAgent_cw5n1h2txyewy_Contoso.ContosoAgent\",
    UseShellExecute = false,
    RedirectStandardOutput = true,
    RedirectStandardError = true,
    CreateNoWindow = true
};

// Start the ODR process
using Process process = new Process { StartInfo = startInfo };
process.Start();

// Read the output
string output = await process.StandardOutput.ReadToEndAsync();
string error = await process.StandardError.ReadToEndAsync();

await process.WaitForExitAsync();

```

The command returns:

JSON

```
{
    "success": true,
    "message": "Agent removed successfully"
}
```

i Important

Due to package identity requirements, you can't use `add-app-agent` and `remove-app-agent` from an unpackaged app. You must run these commands from within a packaged application that also contains the associated App Action.

List available Agent Launchers

To discover all registered Agent Launchers on the system, use the `odr list-app-agents` command. This command returns all Agent Launchers that you can invoke.

PowerShell

```
odr list-app-agents
```

This command returns a JSON array of agent definitions:

JSON

```
{  
  "agents": [  
    {  
      "id": "ContosoAgent_cw5n1h2txyewy_Contoso.ContosoAgent",  
      "version": "1.0.0",  
      "name": "Contoso.ContosoAgent",  
      "display_name": "Contoso Agent",  
      "description": "Description for Contoso agent.",  
      "icon": "C://pathToContosoIcon.png",  
      "package_family_name": "ContosoPackageFamilyName",  
      "action_id": "ContosoAgentAction"  
    }  
  ]  
}
```

Use the `package_family_name` and `action_id` fields together to identify and invoke the associated App Action.

Invoke an Agent Launcher

To invoke an Agent Launcher, follow these steps:

1. Call `odr list-app-agents` to get all available Agent Launchers.
2. Select the agent you want to invoke based on your application logic (for example, user interaction).
3. Use the Windows.AI.Actions APIs to invoke the agent's associated App Action

Here's an example of invoking an Agent Launcher by using the Windows.AI.Actions APIs:

C#

```
using Windows.AI.Actions;  
using System.Threading.Tasks;  
  
public async Task InvokeAgentLauncherAsync(string packageFamilyName, string  
actionId, string agentName, string prompt)  
{  
  // Get the action catalog  
  var catalog = ActionCatalog.GetDefault();  
  
  // Get all actions for the package  
  var actions = await catalog.GetAllActionsAsync();
```

```

// Find the specific action by package family name and action ID
var targetAction = actions.FirstOrDefault(a =>
    a.PackageFamilyName == packageFamilyName &&
    a.Id == actionId);

if (targetAction != null)
{
    // Create the input entities
    var entityFactory = new ActionEntityFactory();
    var agentNameEntity = entityFactory.CreateTextEntity(agentName);
    var promptEntity = entityFactory.CreateTextEntity(prompt);

    // Create input dictionary
    var inputs = new Dictionary<string, ActionEntity>
    {
        { "agentName", agentNameEntity },
        { "prompt", promptEntity }
    };

    // Invoke the action
    await targetAction.InvokeAsync(inputs);
}
}

```

Test your Agent Launcher

After you verify the functionality of your App Action, test your Agent Launcher registration and invocation.

Test static registration

1. Build and deploy your packaged application with the agent extension in the manifest.
2. Open a terminal and run:

PowerShell

```
odr list-app-agents
```

3. Verify that your Agent Launcher appears in the output with the correct `package_family_name` and `action_id`.

Test dynamic registration

1. Run the `odr add-app-agent` command from within your packaged application as shown in the dynamic registration section.

2. Check the command output to confirm successful registration.

3. Verify the registration by running:

PowerShell

```
odr list-app-agents
```

4. Confirm your agent appears in the list.

5. Test removal by running the `odr remove-app-agent` command with your agent's ID.

6. Confirm removal by running `odr list-app-agents` again and verifying the agent no longer appears.

Test Agent Launcher invocation

After you register your Agent Launcher, test the end-to-end invocation:

1. Run `odr list-app-agents` to get your agent's `package_family_name` and `action_id` values.
2. Use the App Action testing approach from the [Get started with App Actions](#) article or the Action Test Tool to invoke your action with the required `agentName` and `prompt` inputs.
3. Verify that your app receives the inputs correctly and your agent logic executes as expected.
4. Test optional inputs like `attachedFile` if your action supports them.

Last updated on 12/12/2025

Agent definition JSON schema for Agent Launchers on Windows

This article describes the format of the agent definition JSON file for Agent Launchers on Windows. This file must be included in your project with the **Build Action** set to "Content" and **Copy to Output Directory** set to "Copy if newer". Specify the package-relative path to the JSON file in your package manifest XML file.

An Agent Launcher registration links an agent to an App Action that handles the agent invocation. For information on creating the App Action, see [Get started with Agent Launchers on Windows](#).

Example agent definition JSON file

JSON

```
{  
  "manifest_version": "0.1.0",  
  "version": "1.0.0",  
  "name": "Contoso.ContosoAgent",  
  "display_name": "ms-resource://contosoAgentDisplayName",  
  "description": "ms-resource://contosoAgentDescription",  
  "icon": "ms-resource://Files/Assets/ContosoLogo.png",  
  "action_id": "ContosoAgentAction"  
}
```

Agent definition JSON properties

The table below describes the properties of the agent definition JSON file.

Document root

[+] [Expand table](#)

Property	Type	Description	Required
manifest_version	string	The schema version of the agent definition manifest. Current version is "0.1.0".	Yes
version	string	The version of your agent. Use semantic versioning (e.g., "1.0.0").	Yes
name	string	A unique identifier for your agent, typically using reverse domain notation (e.g., "Contoso.ContosoAgent"). This value is not	Yes

Property	Type	Description	Required
		localizable and must be unique within your package.	
display_name	string	The user-facing display name for the agent. This value is localizable using the <code>ms-resource://</code> format to reference a string resource in your app package.	Yes
description	string	A user-facing description of what the agent does. This value is localizable using the <code>ms-resource://</code> format to reference a string resource in your app package.	Yes
icon	string	The icon for the agent. This value is localizable using the <code>ms-resource://</code> format to reference an icon resource deployed with your app package.	Yes
action_id	string	The identifier of the App Action that will handle invocations of this agent. This must match the <code>id</code> field of an action defined in the same app package. For information on creating the associated App Action, see Get started with Agent Launchers on Windows .	Yes

Localization

The `display_name`, `description`, and `icon` properties support localization through the `ms-resource://` URI scheme. This allows you to provide localized strings and resources for different languages.

String resources

To localize string properties, use the following format:

JSON

```
"display_name": "ms-resource://resourceName"
```

The resource name corresponds to a string resource defined in your app package's resource files (`.resw` files for C# projects, or `.rc` files for C++ projects).

Icon resources

To localize icon properties, use the following format:

JSON

```
"icon": "ms-resource://Files/Assets/iconName.png"
```

The path is relative to your package root and can reference different icons for different languages through your app's resource system.

Relationship to App Actions

Each Agent Launcher must reference an App Action through the `action_id` property. The App Action defines how the agent is invoked, including:

- Required input entities (`agentName` and `prompt`)
- Optional input entities (such as `attachedFile`)
- The invocation mechanism (URI activation or COM)

The App Action and Agent Launcher must be in the same app package. When an Agent Launcher is invoked, the system uses the `action_id` to locate the corresponding App Action and invokes it with the appropriate inputs.

For detailed information on creating the App Action for your Agent Launcher, see [Get started with Agent Launchers on Windows](#).

Related articles

- [Get started with Agent Launchers on Windows](#)
- [Agent Launchers on Windows Overview](#)
- [Get started with App Actions on Windows](#)

Last updated on 12/12/2025

DirectML Overview

Article • 02/10/2025

Summary

Direct Machine Learning (DirectML) is a low-level API for machine learning (ML). The API has a familiar (native C++, nano-COM) programming interface and workflow in the style of DirectX 12. You can integrate machine learning inferencing workloads into your game, engine, middleware, backend, or other application. DirectML is supported by all DirectX 12-compatible hardware.

Hardware-accelerated machine learning primitives (called operators) are the building blocks of DirectML. From those building blocks, you can develop such machine learning techniques as upscaling, anti-aliasing, and style transfer, to name but a few. Denoising and super-resolution, for example, allow you to achieve impressive raytraced effects with fewer rays per pixel.

You can integrate machine learning inferencing workloads into your game, engine, middleware, backend, or other application. DirectML has a familiar (native C++, nano-COM) DirectX 12-style programming interface and workflow, and it's supported by all DirectX 12-compatible hardware. For DirectML sample applications, including a sample of a minimal DirectML application, see [DirectML sample applications](#).

DirectML was introduced in Windows 10, version 1903, and in the corresponding version of the Windows SDK.

Is DirectML appropriate for my project?

DirectML is a low-level hardware abstraction layer that enables you to run machine learning workloads on any DirectX 12 compatible GPU.

If you need to optimize your machine learning performance for real-time, high-performance, low-latency, or resource-constrained scenarios, DirectML gives you the most control and flexibility. You can use DirectML to integrate machine learning directly into your existing engine or rendering pipeline, or to build your own custom machine learning frameworks and middleware on Windows.

You can also use DirectML indirectly through the ONNX Runtime, which is a cross-platform library that supports the open standard ONNX format for machine learning models. The ONNX Runtime can use DirectML as one of its execution providers, along

with other backends such as CPU, CUDA, or TensorRT. This way, you can leverage the performance and compatibility of DirectML without writing any DirectML code yourself.

Alternatively, you can use the WinML API, which is a higher-level, model-focused API that simplifies the machine learning workflow with its load-bind-evaluate pattern.

WinML also uses the ONNX format for models, and can use DirectML as its backend.

WinML is designed for scenarios where you need to quickly and easily integrate machine learning into your Windows applications, without worrying about the details of the underlying hardware or framework.

What work does DirectML do; and what work must I do as the developer?

DirectML efficiently executes the individual layers of your inference model on the GPU (or on AI-acceleration cores, if present). Each layer is an operator, and DirectML provides you with a library of low-level, hardware-accelerated machine learning primitive operators. You can execute DirectML operations in isolation or as a graph (see the section [Layer-by-layer and graph-based workflows in DirectML](#)).

Operators and graphs apply hardware-specific and architecture-specific optimizations. At the same time, you as the developer see a single, vendor-agnostic interface for executing those operators.

The library of operators in DirectML supplies all of the usual operations that you'd expect to be able to use in a machine learning workload.

- Activation operators, such as `linear`, `ReLU`, `sigmoid`, `tanh`, and more.
- Element-wise operators, such as `add`, `exp`, `log`, `max`, `min`, `sub`, and more.
- Convolution operators, such as 2D and 3D `convolution`, and more.
- Reduction operators, such as `argmin`, `average`, `l2`, `sum`, and more.
- Pooling operators, such as `average`, `lp`, and `max`.
- Neural network (NN) operators, such as `gemm`, `gru`, `lstm`, and `rnn`.
- And many more.

For maximal performance, and so that you don't pay for what you don't use, DirectML puts the control into your hands as a developer over how your machine learning workload is executed on the hardware. Figuring out which operators to execute, and when, is your responsibility as the developer. Tasks that are left to your discretion include: transcribing the model; simplifying and optimizing your layers; loading weights; resource allocation, binding, memory management (just as with Direct3D 12); and execution of the graph.

You retain high-level knowledge of your graphs (you can hard-code your model directly, or you can write your own model loader). You might design an upscaling model, for example, using several layers each of **upsample**, **convolution**, **normalization**, and **activation** operators. With that familiarity, careful scheduling, and barrier management, you can extract the most parallelism and performance from the hardware. If you're developing a game, then your careful resource management and control over scheduling enables you to interleave machine learning workloads and traditional rendering work in order to saturate the GPU.

What's the high-level DirectML workflow?

Here's the high-level recipe for how we expect DirectML to be used. Within the two main phases of initialization and execution, you record work into command lists and then you execute them on a queue.

Initialization

1. Create your Direct3D 12 resources—the Direct3D 12 device, command queue, command list, and resources such as descriptor heaps.
2. Since you're doing machine learning inferencing as well as your rendering workload, create DirectML resources—the DirectML device and operator instances. If you have a machine learning model where you need to perform a particular type of convolution with a particular size of filter tensor with a particular data type, then those are all parameters into DirectML's **convolution** operator.
3. DirectML records work into Direct3D 12 command lists. So, once initialization is done, you record the binding and initialization of (for example) your convolution operator into your command list. Then, close and execute your command list on your queue as usual.

Execution

1. Upload your weight tensors into resources. A tensor in DirectML is represented using a regular Direct3D 12 resource. For example, if you want to upload your weight data to the GPU, then you do that the same way you would with any other Direct3D 12 resource (use an upload heap, or the copy queue).
2. Next, you need to bind those Direct3D 12 resources as your input and output tensors. Record into your command list the binding and the execution of your operators.
3. Close and execute your command list.

Just as with Direct3D 12, resource lifetime and synchronization are your responsibility. For example, don't release your DirectML objects at least until they've completed execution on the GPU.

Layer-by-layer and graph-based workflows in DirectML

DirectML supports both layer-by-layer and graph-based approaches to model execution. When executing layer-by-layer, you're responsible for creating and initializing each DirectML operator, and individually recording them for execution on a command list. In contrast, when executing a graph you instead build a set of nodes and edges—where each node represents a DirectML operator, and edges represent tensor data flowing between nodes. The entire graph is then submitted for initialization or execution all at once, and DirectML handles the scheduling and recording of the individual operators on your behalf.

Both patterns are useful in different situations. A layer-by-layer approach gives you maximal control over the ordering and scheduling of compute work. For example, this level of control allows you to interleave Direct3D 12 rendering workloads with your DirectML compute dispatches. This can be useful for taking advantage of asynchronous compute or shader units on your GPU that would otherwise be idle. Manually executing layer by layer also gives explicit developer control over tensor layouts and memory usage.

However, machine learning models are often expressed in terms of *graphs* of layers. As an alternative to the layer-by-layer approach, DirectML allows you to express your model as a directed acyclic graph of nodes (DirectML operators) and edges between them (tensor descriptions). After building up a description of the graph, you can compile and submit it all at once to DirectML for initialization and execution. In this approach, DirectML decides on a traversal order, and handles each individual operator and the flow of data between them on your behalf. This is often a simpler and more natural way to express a machine learning model, and allow architecture-specific optimizations to be applied automatically. Additionally, the [DirectMLX](#) helper library provides a clean and convenient syntax for building up complex graphs of DirectML operators.

Whichever approach you prefer, you'll always have access to the same extensive suite of DirectML operators. This means that you never have to sacrifice functionality whether you prefer the fine-grained control of the layer-by-layer approach, or the convenience of the graph approach.

In this section

 Expand table

Topic	Description
QuickStart Guide	Get started using DirectML.
Developer Tools	Tools to profile, optimize, and debug DirectML.
Programming Guide	Topics on data binding, barriers, slides, fusions, error handling, device removal, and helper functions.
Troubleshooting	Handling error conditions and using the debug layer.
DirectML version history	DirectML is a system component of Windows 10, and is also available as a standalone redistributable package.
DirectML feature level history	A manifest of the types introduced in each feature level.
DirectML sample applications	Links to DirectML sample applications, including a sample of a minimal DirectML application.
GPU accelerated ML training	Covers what's currently supported by the GPU accelerated machine learning (ML) training for the Windows Subsystem for Linux (WSL) and native Windows.
DirectML API reference	This section covers Direct Machine Learning (DirectML) APIs declared in <code>DirectML.h</code> .

See also

- [DirectML GitHub repo](#) ↗
- [Windows AI](#)
- [Direct3D 12 programming guide](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) ↗ | Get help at Microsoft Q&A

Get Started with DirectML

ⓘ Important

DirectML is in sustained engineering. DirectML continues to be supported, but new feature development has moved to Windows ML for Windows-based ONNX Runtime deployments. Windows ML provides the same ONNX Runtime APIs while dynamically selecting the best execution provider based on your hardware. See [What is Windows ML](#) to learn more.

Pairing DirectML with the ONNX Runtime is often the most straightforward way for many developers to bring hardware-accelerated AI to their users at scale. These three steps are a general guide for using this powerful combo.

1. Convert

The ONNX format enables you to leverage ONNX Runtime with DirectML, which provides cross-hardware capabilities.

To convert your model to the ONNX format, you can utilize [ONNXMLTools](#) or [Olive](#).

2. Optimize

Once you have an .onnx model, leverage [Olive](#) powered by DirectML to optimize your model. You'll see dramatic performance improvements that you can deploy across the Windows hardware ecosystem.

3. Integrate

When your model is ready, it's time to bring hardware-accelerated inferencing to your app with ONNX Runtime and DirectML. For Generative AI models, we recommend you use the [ONNX Runtime Generate\(\) API](#)

We built some samples to show how you can use DirectML and the ONNX Runtime:

- [Phi-3-mini](#)
- [Large Language Models \(LLMs\)](#)
- [Stable Diffusion](#)
- [Style transfer](#)
- [Inference on NPUs](#)

DirectML and PyTorch

The DirectML backend for Pytorch enables high-performance, low-level access to the GPU hardware, while exposing a familiar Pytorch API for developers. [More information on how to use PyTorch with DirectML can be found here](#)

DirectML for web applications (Preview)

The Web Neural Network API (WebNN) is an emerging web standard that allows web apps and frameworks to accelerate deep neural networks with on-device hardware such as GPUs, CPUs, or purpose-built AI accelerators such as NPUs. The WebNN API leverages the DirectML API on Windows to access the native hardware capabilities and optimize the execution of neural network models. [For more information on WebNN can be found here](#)

Last updated on 01/09/2026

DirectML Tools

Article • 02/10/2025

The following tools are available to enhance DirectML and incorporate it into your AI app.

ONNX Runtime Go Live (Olive)

Olive is an easy-to-use hardware-aware model optimization tool that composes industry-leading techniques across model compression, optimization, and compilation. You can pass a model through Olive with DirectML as the target backend and Olive composes the best suitable optimization techniques to output the most efficient model(s). For more information and samples on how to use Olive, please see [Olive's documentation](#).

DxDispatch

DxDispatch is simple command-line executable for launching DirectX 12 compute programs without writing all the C++ boilerplate. The input to the tool is a JSON model that defines resources, dispatchables (compute shaders, DirectML operators, and ONNX models), and commands to execute. For more information, please see [the DxDispatch guide on Github](#).

DirectMLX

DirectMLX is a C++ header-only helper library for DirectML, intended to make it easier to compose individual operators into graphs. For more information, please visit [the DirectMLX documentation](#)

ONNX Runtime Perf Tests

The onnxruntime perf test is a tool that measures the performance of running ONNX models with different execution providers (EPs) in the onnxruntime framework. It can report metrics such as latency, throughput, memory usage, and CPU/GPU utilization for each EP and model. The onnxruntime perf test can also compare the results of different EPs and models and generate charts and tables for analysis.

To use the onnxruntime perf test with the directml ep, install the onnxruntime-directml package and specify the directml as the EP in the command line arguments. For

example, the following command runs the perf test for the resnet50 model with the directml ep and the default settings:

```
onnxruntime_perf_test -m resnet50 -e directml
```

The perf test will output the average latency, peak working set memory, and average CPU/GPU utilization for the directml ep and the resnet50 model. One can also use other options to customize the perf test, such as changing the number of iterations, the batch size, the concurrency, the warmup runs, the model inputs, and the output formats. For more details, refer to the [onnxruntime perf test documentation](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DirectMLX

Article • 02/10/2025

DirectMLX is a C++ header-only helper library for DirectML, intended to make it easier to compose individual operators into graphs.

DirectMLX provides convenient wrappers for all DirectML (DML) operator types, as well as intuitive operator overloads, which makes it simpler to instantiate DML operators, and chain them into complex graphs.

Where to find `DirectMLX.h`

`DirectMLX.h` is distributed as open-source software under the MIT license. The latest version can be found on the [DirectML GitHub](#).

Version requirements

DirectMLX requires DirectML version 1.4.0, or newer (see [DirectML version history](#)). Older versions of DirectML are not supported.

DirectMLX.h requires a C++11-capable compiler, including (but not limited to):

- Visual Studio 2017
- Visual Studio 2019
- Clang 10

Note that a C++17 (or newer) compiler is the option that we recommend. Compiling for C++11 is possible, but it requires the use of third-party libraries (such as [GSL](#) and [Abseil](#)) to replace missing standard library functionality.

If you have a configuration that fails to compile `DirectMLX.h`, then please [file an issue on our GitHub](#).

Basic usage

C++

```
#include <DirectML.h>
#include <DirectMLX.h>

IDMLDevice* device;
```

```

/* ... */

dml::Graph graph(device);

// Input tensor of type FLOAT32 and sizes { 1, 2, 3, 4 }
auto x = dml::InputTensor(graph, 0,
dml::TensorDesc(DML_TENSOR_DATA_TYPE_FLOAT32, {1, 2, 3, 4}));

// Create an operator to compute the square root of x
auto y = dml::Sqrt(x);

// Compile a DirectML operator from the graph. When executed, this compiled
operator will compute
// the square root of its input.
DML_EXECUTION_FLAGS flags = DML_EXECUTION_FLAG_NONE;
ComPtr<IDMLCompiledOperator> op = graph.Compile(flags, { y });

// Now initialize and dispatch the DML operator as usual

```

Here's another example, which creates a DirectML graph capable of computing the quadratic formula $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

C++

```

#include <DirectML.h>
#include <DirectMLX.h>

IDMLDevice* device;

/* ... */

std::pair<dml::Expression, dml::Expression>
QuadraticFormula(dml::Expression a, dml::Expression b, dml::Expression c)
{
    // Quadratic formula: given an equation of the form  $ax^2 + bx + c = 0$ , x
    can be found by:
    //  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ 
    // https://en.wikipedia.org/wiki/Quadratic_formula

    // Note: DirectMLX provides operator overloads for common mathematical
    expressions. So for
    // example  $a*c$  is equivalent to dml::Multiply(a, c).
    auto x1 = -b + dml::Sqrt(b*b - 4*a*c) / (2*a);
    auto x2 = -b - dml::Sqrt(b*b - 4*a*c) / (2*a);

    return { x1, x2 };
}

/* ... */

dml::Graph graph(device);

```

```

dml::TensorDimensions inputSizes = {1, 2, 3, 4};
auto a = dml::InputTensor(graph, 0,
dml::TensorDesc(DML_TENSOR_DATA_TYPE_FLOAT32, inputSizes));
auto b = dml::InputTensor(graph, 1,
dml::TensorDesc(DML_TENSOR_DATA_TYPE_FLOAT32, inputSizes));
auto c = dml::InputTensor(graph, 2,
dml::TensorDesc(DML_TENSOR_DATA_TYPE_FLOAT32, inputSizes));

auto [x1, x2] = QuadraticFormula(a, b, c);

// When executed with input tensors a, b, and c, this compiled operator
computes the two outputs
// of the quadratic formula, and returns them as two output tensors x1 and
x2
DML_EXECUTION_FLAGS flags = DML_EXECUTION_FLAG_NONE;
ComPtr<IDMLCompiledOperator> op = graph.Compile(flags, { x1, x2 });

// Now initialize and dispatch the DML operator as usual

```

More examples

Complete samples using DirectMLX can be found on the [DirectML GitHub repo](#).

Compile-time options

DirectMLX supports compile-time #define's to customize various parts of the header.

[] Expand table

Option	Description
DMLX_NO_EXCEPTIONS	If #define'd, causes errors to result in a call to <code>std::abort</code> rather than throwing an exception. This is defined by default if exceptions are unavailable (for example, if exceptions have been disabled in the compiler options).
DMLX_USE_WIL	If #define'd, exceptions are thrown using Windows Implementation Library exception types. Otherwise, standard exception types (such as <code>std::runtime_error</code>) are used instead. This option has no effect if <code>DMLX_NO_EXCEPTIONS</code> is defined.
DMLX_USE_ABSEIL	If #define'd, uses Abseil as drop-in replacements for standard library types unavailable in C++11. These types include <code>absl::optional</code> (in place of <code>std::optional</code>), <code>absl::Span</code> (in place of <code>std::span</code>), and <code>absl::InlinedVector</code> .
DMLX_USE GSL	Controls whether to use GSL as the replacement for <code>std::span</code> . If #define'd, uses of <code>std::span</code> are replaced by <code>gsl::span</code> on compilers

Option	Description
	without native <code>std::span</code> implementations. Otherwise, an inline drop-in implementation is provided instead. Note that this option is only used when compiling on a pre-C++20 compiler without support for <code>std::span</code> , and when no other drop-in standard library replacement (like Abseil) is in use.

Controlling tensor layout

For most operators, DirectMLX computes the properties of the operator's output tensors on your behalf. For example when performing a `dml::Reduce` across axes `{ 0, 2, 3 }` with an input tensor of sizes `{ 3, 4, 5, 6 }`, DirectMLX will automatically compute the properties of the output tensor including the correct shape of `{ 1, 4, 1, 1 }`.

However, the other properties of an output tensor include the *Strides*, *TotalTensorSizeInBytes*, and *GuaranteedBaseOffsetAlignment*. By default, DirectMLX sets these properties such that the tensor has no striding, no guaranteed base offset alignment, and a total tensor size in bytes as computed by [DMLCalcBufferSizeTensorSize](#).

DirectMLX supports the ability to customize these output tensor properties, using objects known as *tensor policies*. A `TensorPolicy` is a customizable callback that is invoked by DirectMLX, and returns output tensor properties given a tensor's computed data type, flags, and sizes.

Tensor policies can be set on the `dml::Graph` object, and will be used for all subsequent operators on that graph. Tensor policies can also be set directly when constructing a `TensorDesc`.

The layout of tensors produced by DirectMLX can therefore be controlled by setting a `TensorPolicy` that sets the appropriate strides on its tensors.

Example 1

C++

```
// Define a policy, which is a function that returns a TensorProperties
// given a data type,
// flags, and sizes.
dml::TensorProperties MyCustomPolicy(
    DML_TENSOR_DATA_TYPE dataType,
    DML_TENSOR_FLAGS flags,
    Span<const uint32_t> sizes)
{
    // Compute your custom strides, total tensor size in bytes, and
```

```
guaranteed base
    // offset alignment
    dml::TensorProperties props;
    props.strides = /* ... */;
    props.totalTensorSizeInBytes = /* ... */;
    props.guaranteedBaseOffsetAlignment = /* ... */;
    return props;
};

// Set the policy on the dml::Graph
dml::Graph graph(/* ... */);
graph.SetTensorPolicy(dml::TensorPolicy(&MyCustomPolicy));
```

Example 2

DirectMLX also provides some alternative tensor policies built-in. The **InterleavedChannel** policy, for example, is provided as a convenience, and it can be used to produce tensors with strides such that they are written in NHWC order.

C++

```
// Set the InterleavedChannel policy on the dml::Graph
dml::Graph graph(/* ... */);
graph.SetTensorPolicy(dml::TensorPolicy::InterleavedChannel());

// When executed, the tensor `result` will be in NHWC layout (rather than
// the default NCHW)
auto result = dml::Convolution(/* ... */);
```

See also

- [Windows AI](#)
- [DirectML GitHub](#)
- [DirectMLX yolov4 sample](#)
- [Using strides to express padding and memory layout](#)
- [DML_GRAPH_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DirectML version history

ⓘ Important

DirectML is in sustained engineering. DirectML continues to be supported, but new feature development has moved to Windows ML for Windows-based ONNX Runtime deployments. Windows ML provides the same ONNX Runtime APIs while dynamically selecting the best execution provider based on your hardware. See [What is Windows ML](#) to learn more.

DirectML is distributed as a system component of Windows, and is available as part of the Windows operating system (OS) in Windows 10, version 1903 (10.0; Build 18362) and newer.

Starting with DirectML version 1.4.0, DirectML is also available as a standalone redistributable package (see [Microsoft.AI.DirectML](#)), which is useful for applications that wish to use a fixed version of DirectML, or when running on older versions of Windows 10.

DirectML follows the [semantic versioning](#) conventions. That is, version numbers follow the form `major.minor.patch`. The first release of DirectML has a version of 1.0.0.

Version table

ⓘ [Expand table](#)

DirectML version	Feature level supported (see DirectML feature level history)	DML_TARGET_VERSION	First available in (OS)	First available in (Redistributable)
1.15.4	DML_FEATURE_LEVEL_6_4	0x6400	N/A	DirectML-1.15.4
1.15.3	DML_FEATURE_LEVEL_6_4	0x6400	N/A	DirectML-1.15.3
1.15.2	DML_FEATURE_LEVEL_6_4	0x6400	N/A	DirectML-1.15.2
1.15.1	DML_FEATURE_LEVEL_6_4	0x6400	N/A	DirectML-1.15.1
1.15.0	DML_FEATURE_LEVEL_6_4 and DML_FEATURE_LEVEL_6_3	0x6400 or 0x6300	N/A	DirectML-1.15.0
1.13.1	DML_FEATURE_LEVEL_6_2	0x6200	N/A	DirectML-1.13.1
1.13.0	DML_FEATURE_LEVEL_6_2	0x6200	N/A	DirectML-1.13.0
1.12.0	DML_FEATURE_LEVEL_6_1	0x6100	N/A	DirectML-1.12.0

DirectML version	Feature level supported (see DirectML feature level history)	DML_TARGET_VERSION	First available in (OS)	First available in (Redistributable)
1.11.0	DML_FEATURE_LEVEL_6_0	0x6000	N/A	DirectML-1.11.0 ↗
1.10.0	DML_FEATURE_LEVEL_5_2	0x5200	N/A	DirectML-1.10.0 ↗
1.9.0	DML_FEATURE_LEVEL_5_1	0x5100	N/A	DirectML-1.9.0 ↗
1.8.0	DML_FEATURE_LEVEL_5_0	0x5000	Windows 11 (Build 10.0.22621; 22H2)	DirectML-1.8.0 ↗
1.7.0	DML_FEATURE_LEVEL_4_1	0x4100	N/A	DirectML-1.7.0 ↗
1.6.0	DML_FEATURE_LEVEL_4_0	0x4000	Windows 11 (Build 10.0.22000; 21H2)	DirectML-1.6.0 ↗
1.5.0	DML_FEATURE_LEVEL_3_1	0x3100	N/A	DirectML-1.5.0 ↗
1.4.0 ¹	DML_FEATURE_LEVEL_3_0	0x3000	N/A	DirectML-1.4.0 ↗
1.1.0	DML_FEATURE_LEVEL_2_0	0x2000	Windows 10, version 2004 (10.0; Build 19041) (Windows 10 May 2020 Update). Aka "20H1".	N/A
1.0.0	DML_FEATURE_LEVEL_1_0	0x1000	Windows 10, version 1903 (10.0; Build 18362) (Windows 10 May 2019 Update). Aka "19H1".	N/A

¹ The 1.2.0 and 1.3.0 intermediate releases of DirectML weren't made widely available.

Selecting a DirectML target version

For convenience, certain features in the `DirectML.h` header file are declared conditionally based on the value of the `DML_TARGET_VERSION` macro. By setting the `DML_TARGET_VERSION` macro to certain values, you can exclude parts of `DirectML.h` from your application.

That can be helpful if you're using a newer copy of `DirectML.h`, but you're targeting a lower version of the DirectML binary, because it ensures that any attempt to use features beyond the chosen target level won't compile. This mechanism is similar to the `NTDDI_VERSION` macro (see [Macros for conditional declarations](#)).

Here are the valid values for the `DML_TARGET_VERSION` macro.

[Expand table](#)

<code>DML_TARGET_VERSION</code>	<code>Effect</code>
<code>0x6400</code>	Any features that require a version of DirectML newer than 1.15.0 are excluded from <code>DirectML.h</code> .
<code>0x6300</code>	Any features that require a version of DirectML newer than 1.15.0, or that are <code>DML_FEATURE_LEVEL_6_4</code> features, are excluded from <code>DirectML.h</code> .
<code>0x6200</code>	Any features that require a version of DirectML newer than 1.13.0 are excluded from <code>DirectML.h</code> .
<code>0x6100</code>	Any features that require a version of DirectML newer than 1.12.0 are excluded from <code>DirectML.h</code> .
<code>0x6000</code>	Any features that require a version of DirectML newer than 1.11.0 are excluded from <code>DirectML.h</code> .
<code>0x5200</code>	Any features that require a version of DirectML newer than 1.10.0 are excluded from <code>DirectML.h</code> .
<code>0x5100</code>	Any features that require a version of DirectML newer than 1.9.0 are excluded from <code>DirectML.h</code> .
<code>0x5000</code>	Any features that require a version of DirectML newer than 1.8.0 are excluded from <code>DirectML.h</code> .
<code>0x4100</code>	Any features that require a version of DirectML newer than 1.7.0 are excluded from <code>DirectML.h</code> .
<code>0x4000</code>	Any features that require a version of DirectML newer than 1.6.0 are excluded from <code>DirectML.h</code> .
<code>0x3100</code>	Any features that require a version of DirectML newer than 1.5.0 are excluded from <code>DirectML.h</code> .

DML_TARGET_VERSION	Effect
0x3000	Any features that require a version of DirectML newer than 1.4.0 are excluded from <code>DirectML.h</code> .
0x2000	Any features that require a version of DirectML newer than 1.1.0 are excluded from <code>DirectML.h</code> .
0x1000	Any features that require a version of DirectML newer than 1.0.0 are excluded from <code>DirectML.h</code> .
Not set	The target version is selected automatically for you. See below for details.

If `DML_TARGET_VERSION` is not set, then it is selected automatically by the following.

- If the `DML_TARGET_VERSION_USE_LATEST` macro is defined, then the latest target version is selected.
- Otherwise, the target version is selected based on the value of the `NTDDI_VERSION` macro.
 - `NTDDI_WIN10_ZN` results in a target version of `0x6000`.
 - `NTDDI_WIN10_NI` results in a target version of `0x5000`.
 - `NTDDI_WIN10_CO` results in a target version of `0x4000`.
 - `NTDDI_WIN10_FE` results in a target version of `0x3000`.
 - `NTDDI_WIN10_VB` results in a target version of `0x2000`.
 - `NTDDI_WIN10_19H1` results in a target version of `0x1000`.
 - If `NTDDI_VERSION` is not defined, then the latest target version is selected (as if `DML_TARGET_VERSION_USE_LATEST` were specified).

Example

Consider an application using version 10.0.19041.0 (Windows 10, version 2004) of the Windows Software Development Kit (SDK). From the table above, the version of DirectML that this corresponds to is 1.1.0, and the corresponding `DML_TARGET_VERSION` is `0x2000`.

If you don't set either the `DML_TARGET_VERSION` nor the `NTDDI_VERSION` macros, then the selected target version will default to `0x2000`, and everything in `DirectML.h` will be available for use.

If you want your application to be able to run on Windows 10, version 1903 (10.0; Build 18362), then you could `#define DML_TARGET_VERSION 0x1000`, which will exclude all content in `DirectML.h` that isn't supported by DirectML version 1.0.0. This ensures that any attempt to use functionality that requires a greater version will fail to compile.

DirectML version versus feature level

The DirectML version (for example, 1.0.0, or 1.4.0) describes a particular release of DirectML, including its associated `DirectML.h` header file and `.lib` file.

The feature level (for example, `DML_FEATURE_LEVEL_1_0`, or `DML_FEATURE_LEVEL_2_0`) describes the capability of the underlying implementation of the API, which can vary from the version of `DirectML.h` used.

For example, an application building against a newer SDK, but running on an older version of Windows, might (at runtime) see a lower supported feature level, even if it's compiled against the latest SDK.

See also

- [Windows AI](#)
- [DirectML feature level history](#)
- [DML_FEATURE_LEVEL enumeration](#)
- [Microsoft.AI.DirectML redistributable package ↗](#)

Last updated on 01/09/2026

DirectML feature level history

Article • 02/13/2025

For general DirectML version history, see [DirectML version history](#).

DML_FEATURE_LEVEL_6_4

Introduced in DirectML version 1.15.0.

Added the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_RESAMPLE3
- DML_OPERATOR_FOLD
- DML_OPERATOR_UNFOLD

Extended the following operators to accept the [DML_PADDING_MODE_WRAP](#) padding mode.

- DML_OPERATOR_PADDING
- DML_OPERATOR_PADDING1

Updated [DML_OPERATOR_ACTIVATION_SOFTPLUS](#) to allow Steepness < 1.

DML_FEATURE_LEVEL_6_3

Introduced in DirectML version 1.15.0

Added the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION2
- DML_OPERATOR_MULTIHEAD_ATTENTION1
- DML_OPERATOR_QUANTIZE
- DML_OPERATOR_DEQUANTIZE

Introduced [DML_TENSOR_DATA_TYPE_UINT4](#) and [DML_TENSOR_DATA_TYPE_INT4](#) data types, currently supported by the following operators:

- DML_OPERATOR_QUANTIZE
- DML_OPERATOR_DEQUANTIZE

Optimizations:

- (LLM) Added INT4 Dequantize + GEMM fusion metacommand and DXIL lowerings.
- (LLM) Added Multihead Attention fusion.
- Added Gemm fusion optimizations.
- (Intel ARC GPU) Fix pooling metacommand calls by driver version.

Bug fixes:

- Swish now produces correct output when invoked with strided input tensors.
- Intel:
 - (Precision) FP16 GemmWave emulated on FP32.

DML_FEATURE_LEVEL_6_2

Introduced in DirectML version 1.13.0.

Added the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ACTIVATION_HARD_SWISH
- DML_OPERATOR_ACTIVATION_SWISH
- DML_OPERATOR_AVERAGE_POOLING1
- DML_OPERATOR_LP_POOLING1
- DML_OPERATOR_MATRIX_MULTIPLY_INTEGER_TO_FLOAT
- DML_OPERATOR_QUANTIZED_LINEAR_AVERAGE_POOLING

Extended data type support for the following operators, documented in [DML_OPERATOR_TYPE](#). For details on the specific support added in [DML_FEATURE_LEVEL_6_2](#), see each operator's structure topic.

- DML_OPERATOR_RESAMPLE2

Made *ZeroPointTensor* optional for the following operators:

- DML_OPERATOR_ELEMENT_WISE_DEQUANTIZE_LINEAR
- DML_OPERATOR_ELEMENT_WISE_QUANTIZE_LINEAR

Added a new graph node type [DML_GRAPH_NODE_TYPE_CONSTANT](#) to enable compile-time optimizations that require content of small tensors.

DML_FEATURE_LEVEL_6_1

Introduced in DirectML version 1.12.0.

The operator types mentioned below are documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- Added [DML_OPERATOR_MULTIHEAD_ATTENTION](#).
- [DML_OPERATOR_GEMM](#). *FusedActivation* now supports [DML_OPERATOR_ACTIVATION_SOFTMAX](#) and [DML_OPERATOR_ACTIVATION_SOFTMAX1](#).

DML_FEATURE_LEVEL_6_0

Introduced in DirectML version 1.11.0.

The operator types mentioned below are documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- Added [UINT64](#) and [INT64](#) data type support for [DML_OPERATOR_ELEMENT_WISE_DIVIDE](#), [DML_OPERATOR_ELEMENT_WISE_MODULUS_FLOOR](#), and [DML_OPERATOR_ELEMENT_WISE_MODULUS_TRUNCATE](#).
- Added [FLOAT16](#) data type support in *ScaleTensor* for [DML_OPERATOR_ELEMENT_WISE_QUANTIZE_LINEAR](#).
- Added [FLOAT16](#) data type support in *ScaleTensor* and *OutputTensor* for [DML_OPERATOR_ELEMENT_WISE_DEQUANTIZE_LINEAR](#).
- Added [DML_OPERATOR_ELEMENT_WISE_CLIP](#) operator to the supported fused activation list.

DML_FEATURE_LEVEL_5_2

Introduced in DirectML version 1.10.0.

The operator types mentioned below are documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

The range of tensor dimension has been increased to 1 to 4 for the following parameters:

- [DML_OPERATOR_MATRIX_MULTIPLY_INTEGER](#), *BZeroPointTensor* parameter.
- [DML_OPERATOR_QUANTIZED_LINEAR_CONVOLUTION](#), *FilterScaleTensor* parameter.

ScaleTensor and *BiasTensor* can be null independent of each other for the following operators:

- DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION
- DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION1

DML_FEATURE_LEVEL_5_1

Introduced in DirectML version 1.9.0.

Added the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ACTIVATION_GELU
- DML_OPERATOR_ACTIVATION_SOFTMAX1
- DML_OPERATOR_ACTIVATION_LOG_SOFTMAX1
- DML_OPERATOR_ACTIVATION_HARDMAX1
- DML_OPERATOR_RESAMPLE2
- DML_OPERATOR_RESAMPLE_GRAD1
- DML_OPERATOR_DIAGONAL_MATRIX1

Extended data type support for the following operators, documented in [DML_OPERATOR_TYPE](#). For details on the specific support added in [DML_FEATURE_LEVEL_5_1](#), see each operator's structure topic.

- DML_OPERATOR_ACTIVATION_RELU
- DML_OPERATOR_ACTIVATION_RELU_GRAD
- DML_OPERATOR_ACTIVATION_PARAMETERIZED_RELU
- DML_OPERATOR_ELEMENT_WISE_ADD
- DML_OPERATOR_ELEMENT_WISE_DIVIDE
- DML_OPERATOR_ELEMENT_WISE_MULTIPLY
- DML_OPERATOR_ELEMENT_WISE_SUBTRACT
- DML_OPERATOR_DIAGONAL_MATRIX

DML_FEATURE_LEVEL_5_0

Introduced in DirectML version 1.8.0.

Added the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ELEMENT_WISE_CLIP1
- DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD1
- DML_OPERATOR_ELEMENT_WISE_NEGATE
- DML_OPERATOR_PADDING1

Extended data type support for the following operators, documented in [DML_OPERATOR_TYPE](#). For details on the specific support added in [DML_FEATURE_LEVEL_5_0](#), see each operator's structure topic.

- [DML_OPERATOR_CUMULATIVE_PRODUCT](#)
- [DML_OPERATOR_CUMULATIVE_SUMMATION](#)
- [DML_OPERATOR_DEPTH_TO_SPACE](#)
- [DML_OPERATOR_DEPTH_TO_SPACE1](#)
- [DML_OPERATOR_ELEMENT_WISE_CLIP](#)
- [DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD](#)
- [DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD1](#)
- [DML_OPERATOR_ELEMENT_WISE_CLIP1](#)
- [DML_OPERATOR_ELEMENT_WISE_IF](#)
- [DML_OPERATOR_ELEMENT_WISE_MAX](#)
- [DML_OPERATOR_ELEMENT_WISE_MIN](#)
- [DML_OPERATOR_ELEMENT_WISE_NEGATE](#)
- [DML_OPERATOR_FILL_VALUE_SEQUENCE](#)
- [DML_OPERATOR_MAX_POOLING](#)
- [DML_OPERATOR_MAX_POOLING1](#)
- [DML_OPERATOR_MAX_POOLING2](#)
- [DML_OPERATOR_MAX_UNPOOLING](#)
- [DML_OPERATOR_PADDING](#)
- [DML_OPERATOR_PADDING1](#)
- [DML_OPERATOR_REDUCE](#), when using one of the following reduce functions.
 - [DML_REDUCE_FUNCTION_L1](#)
 - [DML_REDUCE_FUNCTION_MAX](#)
 - [DML_REDUCE_FUNCTION_MIN](#)
 - [DML_REDUCE_FUNCTION_MULTIPLY](#)
 - [DML_REDUCE_FUNCTION_SUM](#)
 - [DML_REDUCE_FUNCTION_SUM_SQUARE](#)
- [DML_OPERATOR_REVERSE_SUBSEQUENCES](#)
- [DML_OPERATOR_ROI_ALIGN](#)
- [DML_OPERATOR_ROI_ALIGN1](#)
- [DML_OPERATOR_SPACE_TO_DEPTH](#)
- [DML_OPERATOR_SPACE_TO_DEPTH1](#)
- [DML_OPERATOR_TOP_K](#)
- [DML_OPERATOR_TOP_K1](#)

DML_FEATURE_LEVEL_4_1

Introduced in DirectML version 1.7.0.

Added the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ROI_ALIGN_GRAD
- DML_OPERATOR_BATCH_NORMALIZATION_TRAINING
- DML_OPERATOR_BATCH_NORMALIZATION_TRAINING_GRAD

Extended data type support for the following operators, documented in [DML_OPERATOR_TYPE](#). For details on the specific support added in [DML_FEATURE_LEVEL_4_1](#), see each operator's structure topic.

- DML_OPERATOR_ELEMENT_WISE_IDENTITY
- DML_OPERATOR_ELEMENT_WISE_ADD
- DML_OPERATOR_ELEMENT_WISE_SUBTRACT
- DML_OPERATOR_ELEMENT_WISE_MULTIPLY
- DML_OPERATOR_ELEMENT_WISE_ABS
- DML_OPERATOR_ELEMENT_WISE_SIGN
- DML_OPERATOR_ELEMENT_WISE_LOGICAL_EQUALS
- DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN
- DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN
- DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL
- DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL
- DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_LEFT
- DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_RIGHT
- DML_OPERATOR_ELEMENT_WISE_BIT_AND
- DML_OPERATOR_ELEMENT_WISE_BIT_OR
- DML_OPERATOR_ELEMENT_WISE_BIT_NOT
- DML_OPERATOR_ELEMENT_WISE_BIT_XOR
- DML_OPERATOR_ELEMENT_WISE_BIT_COUNT
- DML_OPERATOR_ARGMIN
- DML_OPERATOR_ARGMAX
- DML_OPERATOR_CAST
- DML_OPERATOR_SLICE
- DML_OPERATOR_SLICE1
- DML_OPERATOR_SLICE_GRAD
- DML_OPERATOR_SPLIT
- DML_OPERATOR_JOIN
- DML_OPERATOR_GATHER
- DML_OPERATOR_GATHER_ELEMENTS
- DML_OPERATOR_GATHER_ND
- DML_OPERATOR_GATHER_ND1
- DML_OPERATOR_SCATTER

- DML_OPERATOR_SCATTER_ND
- DML_OPERATOR_FILL_VALUE_CONSTANT
- DML_OPERATOR_TILE
- DML_OPERATOR_ONE_HOT

DML_FEATURE_LEVEL_4_0

Introduced in DirectML version 1.6.0.

Added support for the following operator types, documented in [DML_OPERATOR_TYPE](#).

For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ELEMENT_WISE_QUANTIZED_LINEAR_ADD
- DML_OPERATOR_DYNAMIC_QUANTIZE_LINEAR
- DML_OPERATOR_ROI_ALIGN1

Extended data type and dimension count support for the following operators, documented in [DML_OPERATOR_TYPE](#). For details on the specific support added in [DML_FEATURE_LEVEL_4_0](#), see each operator's structure topic.

- DML_OPERATOR_ACTIVATION_RELU_GRAD
- DML_OPERATOR_ADAM_OPTIMIZER
- DML_OPERATOR_CONVOLUTION
- DML_OPERATOR_CONVOLUTION_INTEGER
- DML_OPERATOR_CUMULATIVE_PRODUCT
- DML_OPERATOR_CUMULATIVE_SUMMATION
- DML_OPERATOR_DIAGONAL_MATRIX
- DML_OPERATOR_FILL_VALUE_CONSTANT
- DML_OPERATOR_FILL_VALUE_SEQUENCE
- DML_OPERATOR_GEMM
- DML_OPERATOR_MATRIX_MULTIPLY_INTEGER
- DML_OPERATOR_MAX_POOLING_GRAD
- DML_OPERATOR_NONZERO_COORDINATES
- DML_OPERATOR_QUANTIZED_LINEAR_CONVOLUTION
- DML_OPERATOR_QUANTIZED_LINEAR_MATRIX_MULTIPLY
- DML_OPERATOR_RANDOM_GENERATOR
- DML_OPERATOR_REVERSE_SUBSEQUENCES

DML_FEATURE_LEVEL_3_1

Introduced in DirectML version 1.5.0.

Added support for the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ELEMENT_WISE_ATAN_YX
- DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD
- DML_OPERATOR_ELEMENT_WISE_DIFFERENCE_SQUARE
- DML_OPERATOR_LOCAL_RESPONSE_NORMALIZATION_GRAD
- DML_OPERATOR_CUMULATIVE_PRODUCT
- DML_OPERATOR_BATCH_NORMALIZATION_GRAD

The maximum number of supported dimensions for the following operators has increased from 4 to 8.

- DML_OPERATOR_BATCH_NORMALIZATION
- DML_OPERATOR_CAST
- DML_OPERATOR_JOIN
- DML_OPERATOR_LP_NORMALIZATION
- DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION1
- DML_OPERATOR_PADDING
- DML_OPERATOR_ACTIVATION_RELU_GRAD
- DML_OPERATOR_SLICE_GRAD
- DML_OPERATOR_TILE
- DML_OPERATOR_TOP_K
- DML_OPERATOR_TOP_K1

DML_FEATURE_LEVEL_3_0

Introduced in DirectML version 1.4.0.

Added support for the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ELEMENT_WISE_BIT_AND
- DML_OPERATOR_ELEMENT_WISE_BIT_OR
- DML_OPERATOR_ELEMENT_WISE_BIT_XOR
- DML_OPERATOR_ELEMENT_WISE_BIT_NOT
- DML_OPERATOR_ELEMENT_WISE_BIT_COUNT
- DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL

- DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL
- DML_OPERATOR_ACTIVATION_CELU
- DML_OPERATOR_ACTIVATION_RELU_GRAD
- DML_OPERATOR_AVERAGE_POOLING_GRAD
- DML_OPERATOR_MAX_POOLING_GRAD
- DML_OPERATOR_RANDOM_GENERATOR
- DML_OPERATOR_NONZERO_COORDINATES
- DML_OPERATOR_RESAMPLE_GRAD
- DML_OPERATOR_SLICE_GRAD
- DML_OPERATOR_ADAM_OPTIMIZER
- DML_OPERATOR_ARGMIN
- DML_OPERATOR_ARGMAX
- DML_OPERATOR_ROI_ALIGN
- DML_OPERATOR_GATHER_ND1

Added the following enhancements.

- The maximum number of tensor dimensions has been increased from 5 to 8. See [DML_TENSOR_DIMENSION_COUNT_MAX1](#).
- Additional support for integer datatypes has been added to the following operators.
 - DML_OPERATOR_ELEMENT_WISE_POW
 - DML_OPERATOR_ELEMENT_WISE_CONSTANT_POW
 - DML_OPERATOR_MAX_POOLING, DML_OPERATOR_MAX_POOLING1, and DML_OPERATOR_MAX_POOLING2
 - DML_OPERATOR_REDUCE, when using [DML_REDUCE_FUNCTION_ARGMIN](#) or [DML_REDUCE_FUNCTION_ARGMAX](#)
- The following 64-bit data types have been added, and are supported by select operators.
 - DML_TENSOR_DATA_TYPE_FLOAT64
 - DML_TENSOR_DATA_TYPE_UINT64
 - DML_TENSOR_DATA_TYPE_INT64

Deprecated functionality.

- [DML_REDUCE_FUNCTION_ARGMAX](#) and [DML_REDUCE_FUNCTION_ARGMIN](#) have been deprecated. You should prefer to use the standalone [DML_OPERATOR_ARGMIN](#) and [DML_OPERATOR_ARGMAX](#) operators in their place.

DML_FEATURE_LEVEL_2_1

Introduced in DirectML version 1.2.0.

Added the following APIs.

- [IDMLDevice1 interface](#)
- Operator graph support (see [IDMLDevice1::CompileGraph](#))

Added support for the following operator types, documented in [DML_OPERATOR_TYPE](#).

For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_LEFT
- DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_RIGHT
- DML_OPERATOR_ELEMENT_WISE_ROUND
- DML_OPERATOR_ELEMENT_WISE_IS_INFINITY
- DML_OPERATOR_ELEMENT_WISE_MODULUS_TRUNCATE
- DML_OPERATOR_ELEMENT_WISE_MODULUS_FLOOR
- DML_OPERATOR_FILL_VALUE_CONSTANT
- DML_OPERATOR_FILL_VALUE_SEQUENCE
- DML_OPERATOR_CUMULATIVE_SUMMATION
- DML_OPERATOR_REVERSE_SUBSEQUENCES
- DML_OPERATOR_GATHER_ELEMENTS
- DML_OPERATOR_GATHER_ND
- DML_OPERATOR_SCATTER_ND
- DML_OPERATOR_MAX_POOLING2
- DML_OPERATOR_SLICE1
- DML_OPERATOR_TOP_K1
- DML_OPERATOR_DEPTH_TO_SPACE1
- DML_OPERATOR_SPACE_TO_DEPTH1
- DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION1
- DML_OPERATOR_RESAMPLE1
- DML_OPERATOR_MATRIX_MULTIPLY_INTEGER
- DML_OPERATOR_QUANTIZED_LINEAR_MATRIX_MULTIPLY
- DML_OPERATOR_CONVOLUTION_INTEGER
- DML_OPERATOR_QUANTIZED_LINEAR_CONVOLUTION

Added the following enhancements.

- Additional support for integer datatypes has been added to the following operators.
 - DML_OPERATOR_ELEMENT_WISE_IDENTITY
 - DML_OPERATOR_ELEMENT_WISE_ABS
 - DML_OPERATOR_ELEMENT_WISE_ADD

- DML_OPERATOR_ELEMENT_WISE_CLIP
 - DML_OPERATOR_ELEMENT_WISE_DIVIDE
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_EQUALS
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN
 - DML_OPERATOR_ELEMENT_WISE_MAX
 - DML_OPERATOR_ELEMENT_WISE_MEAN
 - DML_OPERATOR_ELEMENT_WISE_MIN
 - DML_OPERATOR_ELEMENT_WISE_MULTIPLY
 - DML_OPERATOR_ELEMENT_WISE_SUBTRACT
 - DML_OPERATOR_ELEMENT_WISE_THRESHOLD
 - DML_OPERATOR_ELEMENT_WISE_QUANTIZE_LINEAR
 - DML_OPERATOR_ELEMENT_WISE_DEQUANTIZE_LINEAR
 - DML_OPERATOR_ELEMENT_WISE_SIGN
 - DML_OPERATOR_ELEMENT_WISE_IF
 - DML_OPERATOR_ACTIVATION_SHRINK
 - DML_OPERATOR_PADDING
 - DML_OPERATOR_GATHER
 - DML_OPERATOR_SCATTER
 - DML_OPERATOR_DEPTH_TO_SPACE
 - DML_OPERATOR_SPACE_TO_DEPTH
 - DML_OPERATOR_TILE
 - DML_OPERATOR_TOP_K and DML_OPERATOR_TOP_K1
 - DML_OPERATOR_ONE_HOT
 - DML_OPERATOR_REDUCE, when using one of the following reduce functions.
 - DML_REDUCE_FUNCTION_ARGMIN
 - DML_REDUCE_FUNCTION_ARGMAX
 - DML_REDUCE_FUNCTION_MAX
 - DML_REDUCE_FUNCTION_MIN
 - DML_REDUCE_FUNCTION_MULTIPLY
 - DML_REDUCE_FUNCTION_SUM
- Relaxed tensor shape restrictions for DML_OPERATOR_GATHER

DML_FEATURE_LEVEL_2_0

Introduced in DirectML version 1.1.0.

Added the following APIs.

- [DMLCreateDevice1 function](#)
- [DML_FEATURE_LEVEL enumeration](#)

- Feature level queries (see [DML_FEATURE_QUERY_FEATURE_LEVELS](#))

Added support for the following operator types, documented in [DML_OPERATOR_TYPE](#). For each operator type constant, that topic provides a link to the corresponding structure.

- DML_OPERATOR_ELEMENT_WISE_SIGN
- DML_OPERATOR_ELEMENT_WISE_IS_NAN
- DML_OPERATOR_ELEMENT_WISE_ERF
- DML_OPERATOR_ELEMENT_WISE_SINH
- DML_OPERATOR_ELEMENT_WISE_COSH
- DML_OPERATOR_ELEMENT_WISE_TANH
- DML_OPERATOR_ELEMENT_WISE_ASINH
- DML_OPERATOR_ELEMENT_WISE_ACOSH
- DML_OPERATOR_ELEMENT_WISE_ATANH
- DML_OPERATOR_ELEMENT_WISE_IF
- DML_OPERATOR_ELEMENT_WISE_ADD1
- DML_OPERATOR_ACTIVATION_SHRINK
- DML_OPERATOR_MAX_POOLING1
- DML_OPERATOR_MAX_UNPOOLING
- DML_OPERATOR_DIAGONAL_MATRIX
- DML_OPERATOR_SCATTER_ELEMENTS
- DML_OPERATOR_SCATTER
- DML_OPERATOR_ONE_HOT
- DML_OPERATOR_RESAMPLE

Added the following enhancements.

- When binding an input resource for dispatch of an [IDMLOperatorInitializer](#), it's now legal to provide a resource with [D3D12_HEAP_TYPE_CUSTOM](#) (in addition to [D3D12_HEAP_TYPE_DEFAULT](#)), as long as appropriate heap properties are also set. See [Binding in DirectML](#).
- The following logical boolean operators now support **UINT8** output tensors, in addition to the existing support for **UINT32**.
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_AND
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_EQUALS
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_NOT
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_OR
 - DML_OPERATOR_ELEMENT_WISE_LOGICAL_XOR

- 5D activation functions now support the use of strides on their input and output tensors.

DML_FEATURE_LEVEL_1_0

The feature level in which DirectML was introduced.

See also

- [Windows AI](#)
- [DirectML version history](#)
- [DML_FEATURE_LEVEL enumeration](#)
- [DMLCreateDevice1 function](#)
- [DML_FEATURE_QUERY_FEATURE_LEVELS structure](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Binding in DirectML

Article • 02/10/2025

In DirectML, *binding* refers to the attachment of resources to the pipeline for the GPU to use during the initialization and execution of your machine learning operators. These resources can be input and output tensors, for example, as well as any temporary or persistent resources that the operator needs.

This topic addresses the conceptual and procedural details of binding. We recommend that you also fully read the documentation for the APIs that you call, including parameters and Remarks.

Important ideas in binding

The list of steps below contain a high-level description of binding-related tasks. You need to follow these steps each time you execute a [dispatchable](#)—a dispatchable is either an operator initializer or a compiled operator. These steps introduce the important ideas, structures, and methods involved in DirectML binding.

Subsequent sections in this topic dig deeper and explain these binding tasks in more detail, with illustrative code snippets taken from the [minimal DirectML application](#) code example.

- Call [IDMLDispatchable::GetBindingProperties](#) on the dispatchable to determine how many descriptors it needs, and also its temporary/persistent resource needs.
- Create a Direct3D 12 descriptor heap large enough for the descriptors, and bind it to the pipeline.
- Call [IDMLDevice::CreateBindingTable](#) to create a DirectML binding table to represent the resources bound to the pipeline. Use the [DML_BINDING_TABLE_DESC](#) structure to describe your binding table, including the subset of the descriptors that it points to in the descriptor heap.
- Create temporary/persistent resources as Direct3D 12 buffer resources, describe them with [DML_BUFFER_BINDING](#) and [DML_BINDING_DESC](#) structures, and add them to the binding table.
- If the dispatchable is a compiled operator, then create a buffer of tensor elements as a Direct3D 12 buffer resource. Populate/upload it, describe it with [DML_BUFFER_BINDING](#) and [DML_BINDING_DESC](#) structures, and add it to the binding table.
- Pass your binding table as a parameter when you call [IDMLCommandRecorder::RecordDispatch](#).

Retrieve the binding properties of a dispatchable

The [DML_BINDING_PROPERTIES](#) structure describes the binding needs of a dispatchable (operator initializer or compiled operator). These binding-related properties include the number of descriptors that you should bind to the dispatchable, as well as the size in bytes of any temporary and/or persistent resource that it needs.

ⓘ Note

Even for multiple operators of the same type, don't make assumptions about them having the same binding requirements. Query the binding properties for every initializer and operator that you create.

Call [IDMLDispatchable::GetBindingProperties](#) to retrieve a **DML_BINDING_PROPERTIES**.

C++/WinRT

```
winrt::com_ptr<::IDMLCompiledOperator> dmlCompiledOperator;
// Code to create and compile a DirectML operator goes here.

DML_BINDING_PROPERTIES executeDmlBindingProperties{
    dmlCompiledOperator->GetBindingProperties()
};

winrt::com_ptr<::IDMLOperatorInitializer> dmlOperatorInitializer;
// Code to create a DirectML operator initializer goes here.

DML_BINDING_PROPERTIES initializeDmlBindingProperties{
    dmlOperatorInitializer->GetBindingProperties()
};

UINT descriptorCount = ...
```

The `descriptorCount` value that you retrieve here determines the (minimum) size of the descriptor heap and of the binding table that you create in the next two steps.

DML_BINDING_PROPERTIES also contains a `TemporaryResourceSize` member, which is the minimum size in bytes of the temporary resource that must be bound to the binding table for this dispatchable object. A value of zero means that a temporary resource is not required.

And a `PersistentResourceSize` member, which is the minimum size in bytes of the persistent resource that must be bound to the binding table for this dispatchable object. A value of zero means that a persistent resource is not required. A persistent resource, if one is needed, must be supplied during initialization of a compiled operator (where it is bound as an output of the operator initializer) as well as during execution. There's more about this later in this topic. Only compiled operators have persistent resources—operator initializers always return a value of 0 for this member.

If you call `IDMLDispatchable::GetBindingProperties` on an operator initializer both before and after a call to `IDMLOperatorInitializer::Reset`, then the two sets of binding properties retrieved are not guaranteed to be identical.

Describe, create, and bind a descriptor heap

In terms of descriptors, your responsibility begins and ends with the descriptor heap itself. DirectML itself takes care of creating and managing the descriptors inside of the heap that you provide.

So, use a `D3D12_DESCRIPTOR_HEAP_DESC` structure to describe a heap large enough for the number of descriptors that the dispatchable needs. Then create it with `ID3D12Device::CreateDescriptorHeap`. And, lastly, call `ID3D12GraphicsCommandList::SetDescriptorHeaps` to bind your descriptor heap to the pipeline.

C++/WinRT

```
winrt::com_ptr<::ID3D12DescriptorHeap> d3D12DescriptorHeap;

D3D12_DESCRIPTOR_HEAP_DESC descriptorHeapDescription{};
descriptorHeapDescription.Type = D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV;
descriptorHeapDescription.NumDescriptors = descriptorCount;
descriptorHeapDescription.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE;

winrt::check_hresult(
    d3D12Device->CreateDescriptorHeap(
        &descriptorHeapDescription,
        _uuidof(d3D12DescriptorHeap),
        d3D12DescriptorHeap.put_void()
    )
);

std::array<ID3D12DescriptorHeap*, 1> d3D12DescriptorHeaps{
    d3D12DescriptorHeap.get() };
d3D12GraphicsCommandList->SetDescriptorHeaps(
    static_cast<UINT>(d3D12DescriptorHeaps.size()),
```

```
d3D12DescriptorHeaps.data()  
);
```

Describe and create a binding table

A DirectML binding table represents the resources that you bind to the pipeline for a dispatchable to use. Those resources could be input and output tensors (or other parameters) for an operator, or they could be various persistent and temporary resources that a dispatchable works with.

Use the [DML_BINDING_TABLE_DESC](#) structure to describe your binding table, including the dispatchable for which the binding table will represent the bindings, and the range of descriptors (from the descriptor heap that you just created) that you wish the binding table to refer to (and into which DirectML may write descriptors). The `descriptorCount` value (one of the binding properties that we retrieved in the first step) tells us what minimum size is, in descriptors, of the binding table required for the dispatchable object. Here, we use that value to indicate the maximum number of descriptors that DirectML is permitted to write into our heap, from the start of both the supplied CPU and GPU descriptor handles.

Then call [IDMLDevice::CreateBindingTable](#) to create the DirectML binding table. In later steps, after we've created further resources for the dispatchable, we'll add those resources to the binding table.

Instead of passing a [DML_BINDING_TABLE_DESC](#) to this call, you can pass `nullptr`, indicating an empty binding table.

C++/WinRT

```
DML_BINDING_TABLE_DESC dmlBindingTableDesc{};  
dmlBindingTableDesc.Dispatchable = dmlOperatorInitializer.get();  
dmlBindingTableDesc.CPUDescriptorHandle = d3D12DescriptorHeap-  
>GetCPUDescriptorHandleForHeapStart();  
dmlBindingTableDesc.GPUDescriptorHandle = d3D12DescriptorHeap-  
>GetGPUDescriptorHandleForHeapStart();  
dmlBindingTableDesc.SizeInDescriptors = descriptorCount;  
  
winrt::com_ptr<::IDMLBindingTable> dmlBindingTable;  
winrt::check_hresult(  
    dmlDevice->CreateBindingTable(  
        &dmlBindingTableDesc,  
        __uuidof(dmlBindingTable),  
        dmlBindingTable.put_void()  
    )  
);
```

The order in which DirectML writes descriptors into the heap is unspecified, so your application must take care not to overwrite the descriptors wrapped by the binding table. The supplied CPU and GPU descriptor handles may come from different heaps, however it is then your application's responsibility to ensure that the entire descriptor range referred to by the CPU descriptor handle is copied into the range referred to by the GPU descriptor handle prior to execution using this binding table. The descriptor heap from which the handles are supplied must have type `D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV`. Additionally, the heap referred to by the `GPUDescriptorHandle` must be a shader-visible descriptor heap.

You can reset a binding table to remove any resources that you've added to it, while at the same time changing any property that you set on its initial `DML_BINDING_TABLE_DESC` (to wrap a new range of descriptors, or to re-use it for a different dispatchable). Just make the changes to the description structure, and call [IDMLBindingTable::Reset](#).

C++/WinRT

```
dmlBindingTableDesc.Dispatchable = pIDMLCompiledOperator.get();

winrt::check_hresult(
    pIDMLBindingTable->Reset(
        &dmlBindingTableDesc
    )
);
```

Describe and bind any temporary/persistent resources

The `DML_BINDING_PROPERTIES` structure that we populated when we [retrieved the binding properties](#) of our dispatchable contains the size in bytes of any temporary and/or persistent resource that the dispatchable needs. If either of these sizes is non-zero, then create a Direct3D 12 buffer resource and add it to the binding table.

In the code example below, we create a temporary resource (`temporaryResourceSize` bytes in size) for the dispatchable. We describe how we wish to bind the resource, and then we add that binding to the binding table.

Since we're binding a single buffer resource, we describe our binding with a `DML_BUFFER_BINDING` structure. In that structure, we specify the Direct3D 12 buffer resource (the resource must have dimension `D3D12_RESOURCE_DIMENSION_BUFFER`), as well as an offset-and-size into the buffer. It's also possible to describe a binding for

an array of buffers (rather than for a single buffer), and the [DML_BUFFER_ARRAY_BINDING](#) structure exists for that purpose.

To abstract away the distinction between a buffer binding and a buffer array binding, we use the [DML_BINDING_DESC](#) structure. You can set the `Type` member of the [DML_BINDING_DESC](#) to either [DML_BINDING_TYPE_BUFFER](#) or [DML_BINDING_TYPE_BUFFER_ARRAY](#). And you can then set the `Desc` member to point to either a [DML_BUFFER_BINDING](#) or to a [DML_BUFFER_ARRAY_BINDING](#), depending on `Type`.

We're dealing with the temporary resource in this example, so we add it to the binding table with a call to [IDMLBindingTable::BindTemporaryResource](#).

C++/WinRT

```
D3D12_HEAP_PROPERTIES defaultHeapProperties{  
    CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_DEFAULT) };  
winrt::com_ptr<::ID3D12Resource> temporaryBuffer;  
  
D3D12_RESOURCE_DESC temporaryBufferDesc{  
    CD3DX12_RESOURCE_DESC::Buffer(temporaryResourceSize) };  
winrt::check_hresult(  
    d3D12Device->CreateCommittedResource(  
        &defaultHeapProperties,  
        D3D12_HEAP_FLAG_NONE,  
        &temporaryBufferDesc,  
        D3D12_RESOURCE_STATE_COMMON,  
        nullptr,  
        __uuidof(temporaryBuffer),  
        temporaryBuffer.put_void()  
    )  
);  
  
DML_BUFFER_BINDING bufferBinding{ temporaryBuffer.get(), 0,  
    temporaryResourceSize };  
DML_BINDING_DESC bindingDesc{ DML_BINDING_TYPE_BUFFER, &bufferBinding };  
dmlBindingTable->BindTemporaryResource(&bindingDesc);
```

A temporary resource (if one is needed) is scratch memory that's used internally during the execution of the operator, so you don't need to be concerned with its contents. Nor do you need to keep it around after your call to

[IDMLCommandRecorder::RecordDispatch](#) has completed on the GPU. This means that your application may release or overwrite the temporary resource in between dispatches of the compiled operator. The supplied buffer range to be bound as the temporary resource must have its start offset aligned to [DML_TEMPORARY_BUFFER_ALIGNMENT](#). The type of the heap underlying the buffer must be [D3D12_HEAP_TYPE_DEFAULT](#).

If the dispatchable reports a non-zero size for its more long-lived persistent resource, though, then the procedure is a little different. You should create a buffer and describe a binding following the same pattern as shown above. But add it to your operator initializer's binding table with a call to [IDMLBindingTable::BindOutputs](#), because it's the operator initializer's job to initialize the persistent resource. Then add it to your compiled operator's binding table with a call to

[IDMLBindingTable::BindPersistentResource](#). See the [minimal DirectML application code example](#) to see this workflow in action. The persistent resource's contents and lifetime must persist as long as the compiled operator does. That is, if an operator requires a persistent resource, then your application must supply it during initialization and subsequently also supply it to all future executes of the operator without modifying its contents. The persistent resource is typically used by DirectML to store lookup tables or other long-lived data that is computed during initialization of an operator and reused on future executions of that operator. The supplied buffer range to be bound as the persistent buffer must have its start offset aligned to

[DML_PERSISTENT_BUFFER_ALIGNMENT](#). The type of the heap underlying the buffer must be [D3D12_HEAP_TYPE_DEFAULT](#).

Describe and bind any tensors

If you're dealing with a compiled operator (rather than with an operator initializer), then you need to bind input and output resources (for tensors and other parameters) to the operator's binding table. The number of bindings must exactly match the number of inputs of the operator, including optional tensors. The particular input and output tensors and other parameters that an operator takes are documented in the topic for that operator (for example, [DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC](#)).

A tensor resource is a buffer that contains the individual element values of the tensor. You upload and read back such a buffer to/from the GPU using the regular Direct3D 12 techniques ([Upload resources](#) and [Read back data via a buffer](#)). See the [minimal DirectML application](#) code example to see these techniques in action.

Lastly, describe your input and output resource bindings with [DML_BUFFER_BINDING](#) and [DML_BINDING_DESC](#) structures, and then add them to the compiled operator's binding table with calls to [IDMLBindingTable::BindInputs](#) and [IDMLBindingTable::BindOutputs](#). When you call an [IDMLBindingTable::Bind*](#) method, DirectML writes one or more descriptors into the range of CPU descriptors.

C++/WinRT

```
DML_BUFFER_BINDING inputBufferBinding{ inputBuffer.get(), 0,
tensorBufferSize };
```

```
DML_BINDING_DESC inputBindingDesc{ DML_BINDING_TYPE_BUFFER,  
&inputBufferBinding };  
dmlBindingTable->BindInputs(1, &inputBindingDesc);  
  
DML_BUFFER_BINDING outputBufferBinding{ outputBuffer.get(), 0,  
tensorBufferSize };  
DML_BINDING_DESC outputBindingDesc{ DML_BINDING_TYPE_BUFFER,  
&outputBufferBinding };  
dmlBindingTable->BindOutputs(1, &outputBindingDesc);
```

One of the steps in creating a DirectML operator (see [IDMLDevice::CreateOperator](#)) is to declare one or more [DML_BUFFER_TENSOR_DESC](#) structures to describe the tensor data buffers that the operator takes and returns. As well as the tensor buffer's type and size, you can optionally specify the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag.

[DML_TENSOR_FLAG OWNED_BY_DML](#) indicates that the tensor data should be owned and managed by DirectML. DirectML makes a copy of the tensor data during initialization of the operator, and stores it in the persistent resource. This allows DirectML to perform reformatting of the tensor data into other, more efficient forms. Setting this flag may increase performance, but it's typically only useful for tensors whose data doesn't change for the lifetime of the operator (for example, weight tensors). And the flag may only be used on input tensors. When the flag is set on a particular tensor description, the corresponding tensor must be bound to the binding table during operator initialization, and not during execution (which will result in an error). That's the opposite of the default behavior (the behavior without the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag), where the tensor is expected to be bound during execution, and not during initialization. All resources bound to DirectML must be DEFAULT or CUSTOM heap resources.

For more info, see [IDMLBindingTable::BindInputs](#) and [IDMLBindingTable::BindOutputs](#).

Execute the dispatchable

Pass your binding table as a parameter when you call [IDMLCommandRecorder::RecordDispatch](#).

When you use the binding table during a call to [IDMLCommandRecorder::RecordDispatch](#), DirectML binds the corresponding GPU descriptors to the pipeline. The CPU and GPU descriptor handles aren't required to point to the same entries in a descriptor heap, however it is then your application's responsibility to ensure that the entire descriptor range referred to by the CPU descriptor handle is copied into the range referred to by the GPU descriptor handle prior to execution using this binding table.

```
winrt::com_ptr<::ID3D12GraphicsCommandList> d3D12GraphicsCommandList;
// Code to create a Direct3D 12 command list goes here.

winrt::com_ptr<::IDMLCommandRecorder> dmlCommandRecorder;
// Code to create a DirectML command recorder goes here.

dmlCommandRecorder->RecordDispatch(
    d3D12GraphicsCommandList.get(),
    dmlOperatorInitializer.get(),
    dmlBindingTable.get()
);
```

Finally, close your Direct3D 12 command list, and submit it for execution as you would any other command list.

Prior to execution of **RecordDispatch** on the GPU, you must transition all bound resources to the **D3D12_RESOURCE_STATE_UNORDERED_ACCESS** state, or to a state implicitly promotable to **D3D12_RESOURCE_STATE_UNORDERED_ACCESS**, such as **D3D12_RESOURCE_STATE_COMMON**. After this call completes, the resources remain in the **D3D12_RESOURCE_STATE_UNORDERED_ACCESS** state. The only exception to this is for upload heaps bound when executing an operator initializer and while one or more tensors has the **DML_TENSOR_FLAG OWNED_BY_DML** flag set. In that case, any upload heaps bound for input must be in the **D3D12_RESOURCE_STATE_GENERIC_READ** state and will remain in that state, as required by all upload heaps. If **DML_EXECUTION_FLAG DESCRIPTORS_VOLATILE** was not set when compiling the operator, then all bindings must be set on the binding table before **RecordDispatch** is called, otherwise the behavior is undefined. Otherwise, if an operator supports **late binding**, then binding of resources may be deferred until the Direct3D 12 command list is submitted to the command queue for execution.

RecordDispatch acts logically like a call to **ID3D12GraphicsCommandList::Dispatch**. As such, unordered access view (UAV) barriers are necessary to ensure correct ordering if there are data dependencies between dispatches. This method does not insert UAV barriers on input nor output resources. Your application must ensure that the correct UAV barriers are performed on any inputs if their contents depend on an upstream dispatch, and on any outputs if there are downstream dispatches that depend on those outputs.

Lifetime and synchronization of descriptors and binding table

A good mental model of binding in DirectML is that behind the scenes the DirectML binding table itself is creating and managing unordered access view (UAV) descriptors inside the descriptor heap that you provide. So, all of the usual Direct3D 12 rules apply around synchronizing access to that heap and to its descriptors. It's your application's responsibility to perform correct synchronization between the CPU and GPU work that uses a binding table.

A binding table can't overwrite a descriptor while the descriptor is in use (by a prior frame, for example). So, if you want to reuse an already-bound descriptor heap (for example, by calling `Bind*` again on a binding table that points to it, or by overwriting the descriptor heap manually), then you should wait for the dispatchable that's currently using the descriptor heap to finish executing on the GPU. A binding table doesn't maintain a strong reference on the descriptor heap that it writes into, so you mustn't release the backing shader-visible descriptor heap until all work using that binding table has completed execution on the GPU.

On the other hand, while a binding table does specify and manage a descriptor heap, the table doesn't itself *contain* any of that memory. So, you may release or reset a binding table any time after you've called [`IDMLCommandRecorder::RecordDispatch`](#) with it (you don't need to wait for that call to complete on the GPU, so long as the underlying descriptors remain valid).

The binding table doesn't keep strong references on any resources bound using it—your application must ensure that resources are not deleted while still in use by the GPU. Also, a binding table isn't thread safe—your application must not call methods on a binding table simultaneously from different threads without synchronization.

And consider that in any case rebinding is necessary only when you change which resources are bound. If you don't need to change the bound resources, then you can bind once at startup, and pass the same binding table each time you call `RecordDispatch`.

For interleaving machine learning and rendering workloads, just ensure that each frame's binding tables points to ranges of the descriptor heap that are not already in use on the GPU.

Optionally specify late-bound operator bindings

If you're dealing with a compiled operator (rather than with an operator initializer), then you have the option to specify late binding for the operator. Without late binding, you must set all bindings on the binding table before you record an operator into a

command list. With late binding, you can set (or change) bindings on operators that you've already recorded into a command list, before it has been submitted to the command queue.

To specify late binding, call `IDMLDevice::CompileOperator` with a `flags` argument of `DML_EXECUTION_FLAG_DESCRIPTORS_VOLATILE`.

See also

- [Windows AI](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

UAV barriers and resource state barriers in DirectML

Article • 02/10/2025

Unordered Access View (UAV) barrier requirements

UAV barriers in Direct3D 12

In Direct3D 12, adjacent compute shader dispatches within the same command list are permitted to execute in parallel on the GPU unless they're synchronized with an intervening unordered access view (UAV) barrier. This can improve performance by increasing utilization of GPU hardware. However, by default, without the use of a UAV barrier, the parallel execution of two adjacent dispatches can cause a race condition if there exists a data dependency between the two dispatches; or if both dispatches perform UAV writes to the same regions of memory.

A UAV barrier forces all previously-submitted dispatches to complete execution on the GPU before subsequent dispatches may begin. UAV barriers are used to synchronize between dispatches on the same command list to avoid data races. You can issue a UAV barrier by using the [ID3D12GraphicsCommandList::ResourceBarrier](#) method.

UAV barriers in DirectML

In DirectML, operators are dispatched in a way that's similar to the way compute shaders are dispatched in Direct3D 12. That is, adjacent dispatches of operators are permitted to execute in parallel on the GPU unless there exists an intervening UAV barrier between them. A typical machine learning model contains data dependencies between its operators; for instance, the output of one operator feeds into the input of another. It's therefore important to use UAV barriers to correctly synchronize dispatches.

DirectML guarantees that it will only ever read from (and never write to) input tensors. It also guarantees that it will never manufacture writes to an output tensor outside the range of the tensor's [DML_BUFFER_TENSOR_DESC::TotalTensorSizeInBytes](#) member. This means that data dependencies between operators in DirectML can be reasoned about by looking only at an operator's input and output bindings.

For example, these guarantees allow you to dispatch two operators that bind the same region of a resource as an input, without having to issue an intervening UAV barrier. This is always safe because DirectML never writes to input tensors. As another example, it's always safe to bind the output tensors of two concurrent operator dispatches to the same Direct3D 12 resource (so long as their tensors don't overlap), because DirectML never writes outside the bounds of a tensor (as defined by the tensor's `DML_BUFFER_TENSOR_DESC::TotalTensorSizeInBytes`).

As UAV barriers are a form of synchronization, unnecessary use of UAV barriers might negatively impact performance. Therefore, it's best for you to use the minimum number of UAV barriers necessary to correctly synchronize the dispatches within a command list.

Example 1

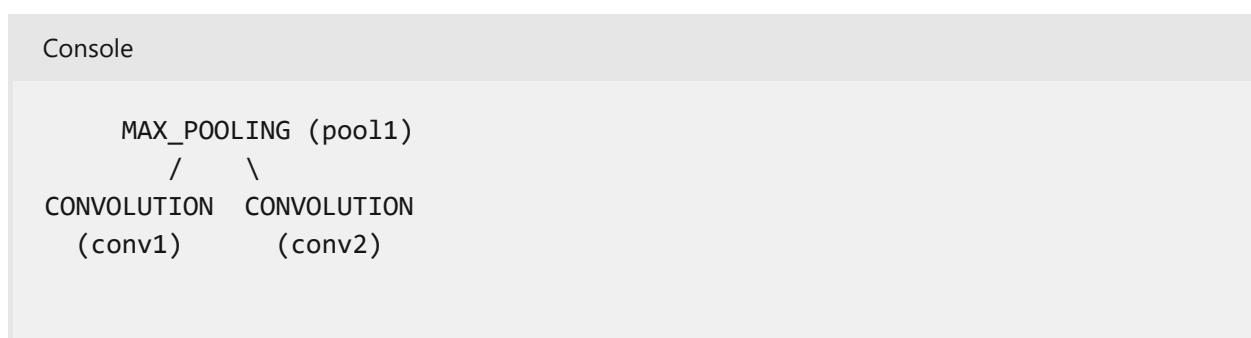
In the following example, a convolution operator's output is fed into a ReLU activation, followed by a batch normalization.



Since a data dependency exists between all three operators, you'll need a UAV barrier between each successive dispatch (see [IDMLCommandRecorder::RecordDispatch](#)).

1. `dmlCommandRecorder->RecordDispatch(d3d12CommandList, conv1)`
2. `d3d12CommandList->ResourceBarrier(UAV barrier)`
3. `dmlCommandRecorder->RecordDispatch(d3d12CommandList, relu1)`
4. `d3d12CommandList->ResourceBarrier(UAV barrier)`
5. `dmlCommandRecorder->RecordDispatch(d3d12CommandList, batch1)`

Example 2



```
\  /
JOIN (join1)
```

Here the output of pooling is fed into two convolutions, whose outputs are then concatenated together using the JOIN operator. A data dependency exists between `pool1` and both `conv1` and `conv2`; as well as between both `conv1` and `conv2` and `join1`. Here's one valid way to execute this graph.

1. `dmlCommandRecorder->RecordDispatch(d3d12CommandList, pool1)`
2. `d3d12CommandList->ResourceBarrier(UAV barrier)`
3. `dmlCommandRecorder->RecordDispatch(d3d12CommandList, conv1)`
4. `dmlCommandRecorder->RecordDispatch(d3d12CommandList, conv2)`
5. `d3d12CommandList->ResourceBarrier(UAV barrier)`
6. `dmlCommandRecorder->RecordDispatch(d3d12CommandList, join1)`

In this case, `conv1` and `conv2` are able to execute concurrently on the GPU, which may improve performance.

Resource barrier state requirements

As the caller, it's your responsibility to ensure that all Direct3D 12 resources are in the correct resource barrier state prior to executing DirectML dispatches on the GPU. DirectML doesn't perform any transition barriers on your behalf.

Prior to execution of [IDMLCommandRecorder::RecordDispatch](#) on the GPU, you must transition all bound resources to the `D3D12_RESOURCE_STATE_UNORDERED_ACCESS` state, or to a state implicitly promotable to `D3D12_RESOURCE_STATE_UNORDERED_ACCESS`, such as `D3D12_RESOURCE_STATE_COMMON`. After this call completes, the resources remain in the `D3D12_RESOURCE_STATE_UNORDERED_ACCESS` state. For more details, see [Binding in DirectML](#).

See also

- [Windows AI](#)
- [Using resource barriers to synchronize resource states in Direct3D 12](#)
- [Binding in DirectML](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Resource lifetime and synchronization

Article • 02/10/2025

Just as with Direct3D 12, your DirectML application must (in order to avoid undefined behavior) correctly manage object lifetimes and synchronization between the CPU and the GPU. DirectML follows an identical resource lifetime model to that of Direct3D 12.

- Lifetime dependencies between two CPU objects are maintained by DirectML using strong reference counts. Your application needn't manually manage CPU lifetime dependencies. For example, every device child holds a strong reference to its parent device.
- Lifetime dependencies between GPU objects—or dependencies that span across CPU and GPU—are not automatically managed. It's your application's responsibility to ensure that GPU resources live at least until all work using that resource has completed execution on the GPU.

DirectML devices

The DirectML device is a thread-safe stateless factory object. Every device child (see [IDMLDeviceChild](#)) holds a strong reference to its parent DirectML device (see [IDMLDevice](#)). That means that you can always retrieve the parent device interface from any device child interface.

A DirectML device in turn holds a strong reference to the Direct3D 12 device that was used to create it (see [ID3D12Device](#), and derived interfaces).

Because the DirectML device is stateless, it's implicitly thread-safe. You may call methods on the DirectML device from multiple threads simultaneously without the need for external synchronization.

However, unlike the Direct3D 12 device, the DirectML device is not a singleton object. You're free to create as many DirectML devices as you wish. However, you may not mix and match device children that belong to different devices. For example,

[IDMLBindingTable](#) and [IDMLCompiledOperator](#) are two kinds of device children (both interfaces derive directly or indirectly from [IDMLDeviceChild](#)). And you may not use a binding table ([IDMLBindingTable](#)) to bind for an operator ([IDMLCompiledOperator](#)) if the operator and the binding table belong to different DirectML device instances.

Because the DirectML device is not a singleton, device removal occurs on a per-device basis, rather than being a process-wide event as it is for a Direct3D 12 device. For more information, see [Handling errors and device removal in DirectML](#).

Lifetime requirements of GPU resources

Like Direct3D 12, DirectML doesn't automatically synchronize between the CPU and GPU; nor does it automatically keep resources alive while they're in use by the GPU. Instead, these are responsibilities of your application.

When executing a command list that contains DirectML dispatches, your application must ensure that GPU resources are kept alive until all work using those resources has completed execution on the GPU.

In the case of [IDMLCommandRecorder::RecordDispatch](#) for a DirectML operator, that includes the following objects.

- The [IDMLCompiledOperator](#) being executed (or [IDMLOperatorInitializer](#) instead, if performing operator initialization).
- The [IDMLCompiledOperator](#) backing the binding table being used to bind the operator.
- The [ID3D12Resource](#) objects bound as the inputs/outputs of the operator.
- The [ID3D12Resource](#) objects bound as persistent and temporary resources, if applicable.
- The [ID3D12CommandAllocator](#) backing the command list itself.

Not all DirectML interfaces represent GPU resources. For example, a binding table does *not* need to be kept alive until all dispatches using it have completed execution on the GPU. That's because the binding table itself doesn't own any GPU resources. Rather, the descriptor heap does. Therefore, the underlying *descriptor heap* is the object that must be kept alive until execution completes, and not the binding table itself.

A similar concept exists in Direct3D 12. A command *allocator* must be kept alive until all executions using it have completed on the GPU; since it owns GPU memory. But, a command *list* doesn't own GPU memory, so it can therefore be reset or released as soon as it's been submitted for execution.

In DirectML, compiled operators ([IDMLCompiledOperator](#)) and operator initializers ([IDMLOperatorInitializer](#)) both own GPU resources directly, so they must be kept alive until all dispatches using them have completed execution on the GPU. In addition, any Direct3D 12 resource being used (command allocators, descriptor heaps, buffers, as examples) must similarly be kept alive by your application.

If you prematurely release an object while it's still in use by the GPU, the result is undefined behavior, which has the potential to cause device removal or other errors.

CPU and GPU synchronization

DirectML doesn't itself submit any work for execution on the GPU. Instead, the [IDMLCommandRecorder::RecordDispatch](#) method *records* the dispatch of that work into a command list for later execution. Your application must then close and submit its command list for execution by calling [ID3D12CommandQueue::ExecuteCommandLists](#), as with any Direct3D 12 command list.

Because DirectML doesn't itself submit any work for execution on the GPU, it also doesn't create any fences, nor perform any form of CPU/GPU synchronization on your behalf. It is the responsibility of your application to use the appropriate Direct3D 12 primitives to wait for submitted work to complete execution on the GPU, if necessary. For more info, see [ID3D12Fence](#) and [ID3D12CommandQueue::Signal](#).

See also

- [Windows AI](#)
- [Executing and synchronizing command lists](#)
- [Handling errors and device removal in DirectML](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using strides to express padding and memory layout

Article • 02/10/2025

DirectML tensors—which are backed by Direct3D 12 buffers—are described by properties known as the *sizes* and the *strides* of the tensor. The tensor's *sizes* describe the logical dimensions of the tensor. For example, a 2D tensor might have a height of 2 and a width of 3. Logically, the tensor has 6 distinct elements, although the sizes don't specify how those elements are stored in memory. The tensor's *strides* describe the physical memory layout of the tensor's elements.

Two-dimensional (2D) arrays

Consider a 2D tensor that has a height of 2 and a width of 3; the data comprises textual characters. In C/C++, this might be expressed using a multi-dimensional array.

C++

```
constexpr int rows = 2;
constexpr int columns = 3;
char tensor[rows][columns];
tensor[0][0] = 'A';
tensor[0][1] = 'B';
tensor[0][2] = 'C';
tensor[1][0] = 'D';
tensor[1][1] = 'E';
tensor[1][2] = 'F';
```

The logical view of the above tensor is visualized below.

Console

```
A B C
D E F
```

In C/C++, a multi-dimensional array is stored in row-major order. In other words, the consecutive elements along the width dimension are stored contiguously in linear memory space.

[+] Expand table

Offset:	0	1	2	3	4	5
Value:	A	B	C	D	E	F

The *stride* of a dimension is the number of elements to skip in order to access the next element in that dimension. Strides express the layout of the tensor in memory. With a row-major order, the stride of the width dimension is always 1, since adjacent elements along the dimension are stored contiguously. The stride of the height dimension depends on the size of the width dimension; in the above example, the distance between consecutive elements along the height dimension (for example, A to D) is equal to the width of the tensor (which is 3 in this example).

To illustrate a different layout, consider column-major order. In other words, the consecutive elements along the height dimension are stored contiguously in linear memory space. In this case, the height-stride is always 1, and the width-stride is 2 (the size of the height dimension).

[\[\] Expand table](#)

Offset:	0	1	2	3	4	5
Value:	A	D	B	E	C	F

Higher dimensions

When it comes to greater than two dimensions, it's unwieldy to refer to a layout as either row-major or column-major. So, the rest of this topic uses terms and labels such as these.

- 2D: "HW"—height is the highest-order dimension (row-major).
- 2D: "WH"—width is the highest-order dimension (column-major).
- 3D: "DHW"—depth is the highest-order dimension, followed by height, and then width.
- 3D: "WHD"—width is the highest-order dimension, followed by height, and then depth.
- 4D: "NCHW"—the number of images (batch size), then the number of channels, then height, then width.

In general, the *packed* stride of a dimension is equal to the product of the sizes of the lower-order dimensions. For example, with a "DHW" layout, the D-stride is equal to H * W; the H-stride is equal to W; and the W-stride is equal to 1. Strides are said to be

packed when the total physical size of the tensor is equal to the total logical size of the tensor; in other words, there's no extra space nor overlapping elements.

Let's extend the 2D example to three dimensions, so that we have a tensor with depth 2, height 2, and width 3 (for a total of 12 logical elements).

Console
A B C D E F G H I J K L

With a "DHW" layout, this tensor is stored as follows.

Offset:	0	1	2	3	4	5	6	7	8	9	10	11
Value:	A	B	C	D	E	F	G	H	I	J	K	L

- D-stride = height (2) * width (3) = 6 (for example, the distance between 'A' and 'G').
- H-stride = width (3) = 3 (for example, the distance between 'A' and 'D').
- W-stride = 1 (for example, the distance between 'A' and 'B').

The dot product of the indices/coordinates of an element and the strides provides the offset to that element in the buffer. For example, the offset of the H element (d=1, h=0, w=1) is 7.

$$\{1, 0, 1\} \cdot \{6, 3, 1\} = 1 * 6 + 0 * 3 + 1 * 1 = 7$$

Packed tensors

The examples above illustrate *packed* tensors. A tensor is said to be *packed* when the logical size of the tensor (in elements) is equal to the physical size of the buffer (in elements), and each element has a unique address/offset. For example, a 2x2x3 tensor is packed if the buffer is 12 elements in length and no pair of elements share the same offset in the buffer. Packed tensors are the most common case; but strides allow more complex memory layouts.

Broadcasting with strides

If a tensor's buffer size (in elements) is smaller than the product of its logical dimensions, then it follows that there must be some overlapping of elements. The usual case for this is known as *broadcasting*; where the elements of a dimension are a duplicate of another dimension. For example, let's revisit the 2D example. Let's say that we want a tensor that is logically 2x3, but the second row is identical to the first row. Here's how that looks.

```
Console
A B C
A B C
```

This could be stored as a packed HW/row-major tensor. But a more compact storage would contain only 3 elements (A, B, and C) and use a height-stride of 0 instead of 3. In this case, the physical size of the tensor is 3 elements, but the logical size is 6 elements.

In general, if the stride of a dimension is 0, then all elements in the lower-order dimensions are repeated along the broadcasted dimension; for example, if the tensor is NCHW and the C-stride is 0, then each channel has the same values along H and W.

Padding with strides

A tensor is said to be *padded* if its physical size is larger than the minimum size needed to fit its elements. When there is no broadcasting nor overlapping elements, the minimum size of the tensor (in elements) is simply the product of its dimensions. You can use the helper function `DMLCalcBufferSize` (see [DirectML helper functions](#) for a listing of that function) to calculate the *minimum* buffer size for your DirectML tensors.

Let's say that a buffer contains the following values (the 'x' elements indicate padding values).

[+] Expand table

0	1	2	3	4	5	6	7	8	9
A	B	C	x	x	D	E	F	x	x

The padded tensor can be described by using a height-stride of 5 instead of 3. Instead of stepping by 3 elements to get to the next row, the step is 5 elements (3 *real* elements plus 2 padding elements). Padding is common in computer graphics, for example, to ensure that an image has a power-of-two alignment.

```
Console
```

```
A B C  
D E F
```

DirectML buffer tensor descriptions

DirectML can work with a variety of physical tensor layouts, since the [DML_BUFFER_TENSOR_DESC structure](#) has both `Sizes` and `Strides` members. Some operator implementations might be more efficient with a specific layout, so it's not uncommon to change how tensor data is stored for better performance.

Most DirectML operators require either 4D or 5D tensors, and the order of the sizes and strides values is fixed. By fixing the order of the sizes and stride values in a tensor description, it's possible for DirectML to infer different physical layouts.

4D

- [DML_BUFFER_TENSOR_DESC::Sizes](#) = { N-size, C-size, H-size, W-size }
- [DML_BUFFER_TENSOR_DESC::Strides](#) = { N-stride, C-stride, H-stride, W-stride }

5D

- [DML_BUFFER_TENSOR_DESC::Sizes](#) = { N-size, C-size, D-size, H-size, W-size }
- [DML_BUFFER_TENSOR_DESC::Strides](#) = { N-stride, C-stride, D-stride, H-stride, W-stride }

If a DirectML operator requires a 4D or a 5D tensor, but the actual data has a smaller rank (for example, 2D), then the leading dimensions should be filled with 1s. For example, an "HW" tensor is set using [DML_BUFFER_TENSOR_DESC::Sizes](#) = { 1, 1, H, W }.

If tensor data is stored in NCHW/NCDHW, then it's not necessary to set [DML_BUFFER_TENSOR_DESC::Strides](#), unless you want broadcasting or padding. You can set the strides field to `nullptr`. However, if the tensor data is stored in another layout, such as NHWC, then you need strides in order to express the transformation from NCHW to that layout.

For a simple example, consider the description of a 2D tensor with height 3 and width 5.

Packed NCHW (implicit strides)

- [DML_BUFFER_TENSOR_DESC::Sizes](#) = { 1, 1, 3, 5 }
- [DML_BUFFER_TENSOR_DESC::Strides](#) = `nullptr`

Packed NCHW (explicit strides)

- N-stride = C-size * H-size * W-size = $1 * 3 * 5 = 15$
- C-stride = H-size * W-size = $3 * 5 = 15$
- H-stride = W-size = 5
- W-stride = 1
- **DML_BUFFER_TENSOR_DESC::Sizes** = { 1, 1, 3, 5 }
- **DML_BUFFER_TENSOR_DESC::Strides** = { 15, 15, 5, 1 }

Packed NHWC

- N-stride = H-size * W-size * C-size = $3 * 5 * 1 = 15$
- H-stride = W-size * C-size = $5 * 1 = 5$
- W-stride = C-size = 1
- C-stride = 1
- **DML_BUFFER_TENSOR_DESC::Sizes** = { 1, 1, 3, 5 }
- **DML_BUFFER_TENSOR_DESC::Strides** = { 15, 1, 5, 1 }

See also

- [Windows AI](#)
- [DirectML helper functions](#)
- [DML_BUFFER_TENSOR_DESC structure](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using fused operators to improve performance

Article • 02/10/2025

Some DirectML operators support a concept known as *fusion*. Operator fusion is a way to improve performance by merging one operator (typically, an activation function) into a different operator so that they are executed together without requiring a roundtrip to memory.

When to fuse activations

Fused activations are a performance optimization. An extremely common scenario in many machine learning (ML) models is to apply a nonlinearity (an activation function) to the output of each layer in the model.

Ordinarily, this requires a roundtrip to graphics memory. For example if a Convolution is followed by a non-fused Relu activation, then the GPU must wait for the results of the Convolution to be written into GPU memory before it can begin computing the Relu activation layer. As the compute workload of most activation functions tends to be small, this roundtrip to graphics memory can be a major performance bottleneck.

Operator fusion allows the activation function (Relu in the above example) to be performed as part of the preceding operator (Convolution, for example). This allows the GPU to compute the activation function without waiting for the results of the preceding operator to be written into memory—and that improves performance.

Because fused activations produce the same result, but are faster in many cases, we recommend that you eliminate activation layers by fusing them into their preceding operator wherever possible.

How to fuse activations

Operators that support fused activations have an additional optional parameter in their operator struct, `const DML_OPERATOR_DESC* FusedActivation`. Convolution, for example, supports fused activation, and it has a corresponding *FusedActivation* in its operator description (see [DML_CONVOLUTION_OPERATOR_DESC](#)).

C++

```

struct DML_CONVOLUTION_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* FilterTensor;
    _Maybenull_ const DML_TENSOR_DESC* BiasTensor;
    const DML_TENSOR_DESC* OutputTensor;
    DML_CONVOLUTION_MODE Mode;
    DML_CONVOLUTION_DIRECTION Direction;
    UINT DimensionCount;
    _Field_size_(DimensionCount) const UINT* Strides;
    _Field_size_(DimensionCount) const UINT* Dilations;
    _Field_size_(DimensionCount) const UINT* StartPadding;
    _Field_size_(DimensionCount) const UINT* EndPadding;
    _Field_size_(DimensionCount) const UINT* OutputPadding;
    UINT GroupCount;
    _Maybenull_ const DML_OPERATOR_DESC* FusedActivation;
};

```

To fuse an activation, construct a [DML_OPERATOR_DESC](#) that describes the type of activation to be fused. For example to fuse a Relu function, the correct operator type would be [DML_OPERATOR_ACTIVATION_RELU](#).

Note

When constructing the operator description for the activation function, you must set the *InputTensor* and *OutputTensor* parameters for the activation function to **NULL**.

Example

C++

```

DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC leakyReluDesc;
leakyReluDesc.InputTensor = nullptr;
leakyReluDesc.OutputTensor = nullptr;
leakyReluDesc.Alpha = 0.01f;

DML_OPERATOR_DESC activationDesc = { DML_OPERATOR_ACTIVATION_LEAKY_RELU,
&leakyReluDesc };

DML_CONVOLUTION_OPERATOR_DESC convDesc;
// ...
convDesc.FusedActivation = &activationDesc;

```

For a complete example, the [DirectMLSuperResolution sample](#) utilizes fused activations to improve performance.

Operators that support fused activation

The list below is based on constants from the [DML_OPERATOR_TYPE](#) enumeration. Each constant in that topic links to the appropriate description structure to use.

- [DML_OPERATOR_BATCH_NORMALIZATION](#)
- [DML_OPERATOR_BATCH_NORMALIZATION_TRAINING](#)
- [DML_OPERATOR_CONVOLUTION](#)
- [DML_OPERATOR_ELEMENT_WISE_ADD1](#)
- [DML_OPERATOR_GEMM](#)
- [DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION](#)
- [DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION1](#)

Activations that are supported for fusion

The list below is based on constants from the [DML_OPERATOR_TYPE](#) enumeration. Each constant in that topic links to the appropriate description structure to use.

- [DML_OPERATOR_ACTIVATION_LINEAR](#)
- [DML_OPERATOR_ACTIVATION_SIGMOID](#)
- [DML_OPERATOR_ACTIVATION_HARD_SIGMOID](#)
- [DML_OPERATOR_ACTIVATION_TANH](#)
- [DML_OPERATOR_ACTIVATION_SCALED_TANH](#)
- [DML_OPERATOR_ACTIVATION_RELU](#)
- [DML_OPERATOR_ACTIVATION_LEAKY_RELU](#)
- [DML_OPERATOR_ACTIVATION_THRESHOLDED_RELU](#)
- [DML_OPERATOR_ACTIVATION_ELU](#)
- [DML_OPERATOR_ACTIVATION_CELU](#)
- [DML_OPERATOR_ACTIVATION_SCALED_ELU](#)
- [DML_OPERATOR_ACTIVATION_SOFTPLUS](#)
- [DML_OPERATOR_ACTIVATION_PARAMETRIC_SOFTPLUS](#)
- [DML_OPERATOR_ACTIVATION_SOFTSIGN](#)
- [DML_OPERATOR_ACTIVATION_IDENTITY](#)
- [DML_OPERATOR_ACTIVATION_SHRINK](#)
- [DML_OPERATOR_ACTIVATION_GELU](#)
- [DML_OPERATOR_ELEMENT_WISE_CLIP](#) (For Convolution and GEMM only)

Any operators that aren't in this list aren't supported for fused activation.

See also

- Windows AI
 - DirectMLSuperResolution sample ↗
 - DML_CONVOLUTION_OPERATOR_DESC
-

Feedback

Was this page helpful?



Yes



No

Provide product feedback ↗ | Get help at Microsoft Q&A

DirectML helper functions

Article • 02/10/2025

DMLCalcBufferSizeTensorSize

This helper function calculates the minimum number of bytes required to store a buffer tensor with the specified type, sizes, and strides. The formula can be expressed as the following.

C++

```
IndexOfLastElement = dot(Sizes - 1, Strides);
MinimumImpliedSizeInBytes = roundup((IndexOfLastElement + 1) *
ElementSizeInBytes, 4)
```

In other words, the minimum size of a tensor is the index of the one-past-the-end element, multiplied by the element size (for example, 2 bytes for a **FLOAT16** tensor). Additionally, DirectML requires that all bound buffers must have a total size that is **DWORD**-aligned, and hence the minimum implied size in bytes must be rounded up to the nearest 4-byte boundary.

C++/WinRT

```
inline UINT64 DMLCalcBufferSizeTensorSize(
    DML_TENSOR_DATA_TYPE dataType,
    UINT dimensionCount,
    _In_reads_(dimensionCount) const UINT* sizes,
    _In_reads_opt_(dimensionCount) const UINT* strides)
{
    UINT elementSizeInBytes = 0;
    switch (dataType)
    {
        case DML_TENSOR_DATA_TYPE_FLOAT32:
        case DML_TENSOR_DATA_TYPE_UINT32:
        case DML_TENSOR_DATA_TYPE_INT32:
            elementSizeInBytes = 4;
            break;

        case DML_TENSOR_DATA_TYPE_FLOAT16:
        case DML_TENSOR_DATA_TYPE_UINT16:
        case DML_TENSOR_DATA_TYPE_INT16:
            elementSizeInBytes = 2;
            break;

        case DML_TENSOR_DATA_TYPE_UINT8:
        case DML_TENSOR_DATA_TYPE_INT8:
```

```

        elementSizeInBytes = 1;
        break;

    case DML_TENSOR_DATA_TYPE_FLOAT64:
    case DML_TENSOR_DATA_TYPE_UINT64:
    case DML_TENSOR_DATA_TYPE_INT64:
        elementSizeInBytes = 8;
        break;

    default:
        return 0; // Invalid data type
    }

UINT64 minimumImpliedSizeInBytes = 0;
if (!strides)
{
    minimumImpliedSizeInBytes = sizes[0];
    for (UINT i = 1; i < dimensionCount; ++i)
    {
        minimumImpliedSizeInBytes *= sizes[i];
    }
    minimumImpliedSizeInBytes *= elementSizeInBytes;
}
else
{
    UINT indexOfLastElement = 0;
    for (UINT i = 0; i < dimensionCount; ++i)
    {
        indexOfLastElement += (sizes[i] - 1) * strides[i];
    }

    minimumImpliedSizeInBytes = (static_cast<UINT64>(indexOfLastElement)
+ 1) * elementSizeInBytes;
}

// Round up to the nearest 4 bytes.
minimumImpliedSizeInBytes = (minimumImpliedSizeInBytes + 3) & ~3ull;

return minimumImpliedSizeInBytes;
}

```

CalculateStrides

This helper function calculates strides for 4D tensors with either NCHW or NHWC layout, and optional broadcasting.

C++/WinRT

```

enum class Layout
{
    NCHW,

```

```
NHWC
};

// Given dimension sizes (in NCHW order), calculates the strides to achieve
// a desired layout.
std::array<uint32_t, 4> CalculateStrides(
    Layout layout,
    std::array<uint32_t, 4> sizes,
    std::array<bool, 4> broadcast)
{
    enum DML_ORDER { N, C, H, W };

    uint32_t n = broadcast[N] ? 1 : sizes[N];
    uint32_t c = broadcast[C] ? 1 : sizes[C];
    uint32_t h = broadcast[H] ? 1 : sizes[H];
    uint32_t w = broadcast[W] ? 1 : sizes[W];

    uint32_t nStride = 0, cStride = 0, hStride = 0, wStride = 0;

    switch (layout)
    {
        case Layout::NCHW:
            nStride = broadcast[N] ? 0 : c * h * w;
            cStride = broadcast[C] ? 0 : h * w;
            hStride = broadcast[H] ? 0 : w;
            wStride = broadcast[W] ? 0 : 1;
            break;

        case Layout::NHWC:
            nStride = broadcast[N] ? 0 : h * w * c;
            hStride = broadcast[H] ? 0 : w * c;
            wStride = broadcast[W] ? 0 : c;
            cStride = broadcast[C] ? 0 : 1;
            break;
    }

    return { nStride, cStride, hStride, wStride };
}
```

See also

- [Windows AI](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

Using the DirectML debug layer

Article • 02/10/2025

The DirectML debug layer is an optional development-time component that helps you in debugging your DirectML code. When enabled, the DirectML debug layer wraps DirectML API calls, and provides additional validation and messages to you as the developer. The debug layer is implemented in a separate library, `DirectML.Debug.dll`, which is conditionally loaded at runtime by the core runtime library `DirectML.dll`.

We highly recommend that you enable the debug layer while developing applications using DirectML, because it can provide invaluable information in the event of invalid API usage.

Overview of debug layer messages

The code example below illustrates how the debug layer can help in diagnosing incorrect API usage. This code attempts to construct a DirectML identity operation; so the input and output tensors should have the same shape and data type. However, in this example we illustrate a mistake in the output tensor parameters.

C++

```
uint32_t sizes[] = { 1 };

DML_BUFFER_TENSOR_DESC inputBufferDesc = {};
inputBufferDesc.DataType = DML_TENSOR_DATA_TYPE_FLOAT32;
inputBufferDesc.DimensionCount = ARRSIZE(sizes);
inputBufferDesc.Sizes = sizes;
inputBufferDesc.TotalTensorSizeInBytes = 256;

DML_BUFFER_TENSOR_DESC outputBufferDesc = {};
outputBufferDesc.DataType = DML_TENSOR_DATA_TYPE_FLOAT16; // Invalid:
// doesn't match input type!
outputBufferDesc.DimensionCount = ARRSIZE(sizes);
outputBufferDesc.Sizes = sizes;
outputBufferDesc.TotalTensorSizeInBytes = 256;

DML_TENSOR_DESC inputDesc = { DML_TENSOR_TYPE_BUFFER, &inputBufferDesc };
DML_TENSOR_DESC outputDesc = { DML_TENSOR_TYPE_BUFFER, &outputBufferDesc };

DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC identityDesc = {};
identityDesc.InputTensor = &inputDesc;
identityDesc.OutputTensor = &outputDesc;

DML_OPERATOR_DESC opDesc = { DML_OPERATOR_ELEMENT_WISE_IDENTITY,
&identityDesc };
```

```
Microsoft::WRL::ComPtr<IDMLOperator> op;
THROW_IF_FAILED(dm1Device->CreateOperator(&opDesc, IID_PPV_ARGS(&op)));
```

Without the DirectML debug layer, the final line to create the operator fails and returns `E_INVALIDARG` (0x80070057). The `THROW_IF_FAILED` macro (for more details, see [WIL](#)) translates that error code into the generic message "the parameter is incorrect", and prints it to the debugger output window.

Console

```
TensorValidator.h(203)\DirectML.dll!00007FF83D25ADC9: (caller:
00007FF83D267523) Exception(1) tid(3b54) 80070057 The parameter is
incorrect.
```

But when the DirectML debug layer *is* enabled, you'll see additional information to narrow down the cause:

Console

```
D3D12 ERROR: Mismatched tensor data types. Tensor 'Output' has DataType of
DML_TENSOR_DATA_TYPE_FLOAT16, while tensor 'Input' has DataType of
DML_TENSOR_DATA_TYPE_FLOAT32. Both tensors are expected to have the same
DataType. [ UNKNOWN ERROR #1: STRING_FROM_APPLICATION]
```

```
TensorValidator.h(203)\DirectML.Debug.dll!00007FF86DF66ADA: (caller:
00007FF86DF81646) Exception(1) tid(9f34) 80070057 The parameter is
incorrect.
```

Notice how the extended information starts with *D3D12 ERROR*. When the DirectML debug layer detects an issue, it always prefers to send error messages to the **ID3D12InfoQueue** associated with the **ID3D12Device** passed in during DirectML device creation. Error messages in the info queue are always prefixed with *D3D12 ERROR*, as shown above; and they're also accessible programmatically using a Direct3D 12 debug layer message callback (see the blog post [D3D12 debug layer message callback](#)).

The **ID3D12InfoQueue** is available only when the Direct3D 12 debug layer is enabled with **ID3D12Debug::EnableDebugLayer**. While it's always preferable to enable (or disable) both the Direct3D 12 and DirectML debug layers together, newer versions of DirectML support basic parameter validation without the Direct3D 12 debug layer. If you create a DirectML device with **DML_CREATE_DEVICE_FLAG_DEBUG** while the Direct3D 12 debug layer hasn't been enabled, error messages are instead printed using **OutputDebugStringA**:

Console

```
[DIRECTML WARNING]: enable the D3D debug layer for enhanced validation with  
DML_CREATE_DEVICE_FLAG_DEBUG.
```

```
[DIRECTML ERROR]: Mismatched tensor data types. Tensor 'Output' has DataType  
of DML_TENSOR_DATA_TYPE_FLOAT16, while tensor 'Input' has DataType of  
DML_TENSOR_DATA_TYPE_FLOAT32. Both tensors are expected to have the same  
DataType.
```

```
TensorValidator.h(218)\DirectML.Debug.dll!00007FF820C43AFB: (caller:  
00007FF820C01CD1) Exception(1) tid(5df8) 80070057 The parameter is  
incorrect.
```

As the warning message suggests, it's best to enable the Direct3D 12 debug layer when also using the DirectML debug layer. Some types of validation are possible only when both debug layers are enabled.

Installing the DirectML and Direct3D 12 debug layers (system component)

When using DirectML as a system component (see [DirectML version history](#)), the debug layer is part of a separate Graphics Tools package, distributed as a feature-on-demand (FOD) (see [Features On Demand](#)). The Graphics Tools FOD must be added to your system in order to use the debug layer with the system version of DirectML. The FOD also contains the Direct3D 12 debug layer, which is also useful (but not required) for debugging DirectML applications.

To add the optional Graphics Tools FOD package, run the following command from an administrator Powershell prompt.

```
PowerShell
```

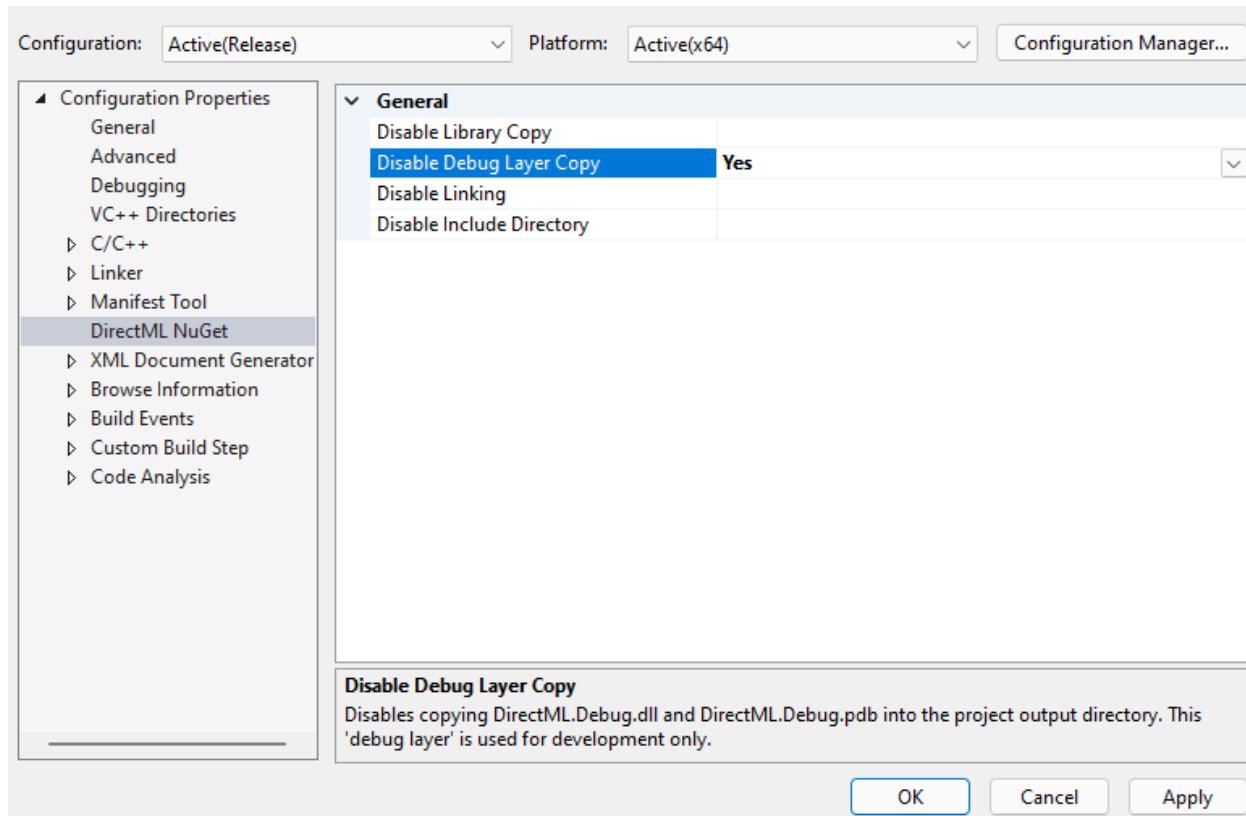
```
Add-WindowsCapability -Online -Name "Tools.Graphics.DirectX~~~~~0.0.1.0"
```

Alternatively, you can add the Graphics Tools package from within Windows Settings. On Windows 10 22H2 and Windows 11, navigate to **Settings > System > Optional features > Add an optional feature**, then search for **Graphics Tools**. On versions older than Windows 10 22H2, navigate to **Settings > Apps > Apps & features > Optional features > Add an optional feature** instead.

Installing the DirectML debug layer (standalone redistributable)

When using DirectML as a standalone redistributable library (see [Microsoft.AI.DirectML](#)), the DirectML debug layer is provided in the package alongside the core runtime library. Place both `DirectML.Debug.dll` and `DirectML.dll` next to your application's executable.

If you use Visual Studio to add `Microsoft.AI.DirectML` as a NuGet package dependency, the project will show options in the project configuration page to copy or skip copying the core runtime and debug layer libraries. By default, the DirectML NuGet package is configured to always copy both DLLs to your project output folder. However, you might want to skip copying the debug layer in release builds if the debug layer isn't used.



Enabling the Direct3D 12 Debug Layer

The [debug layer for Direct3D 12](#) (`d3d12sdklayers.dll`) is independent of the DirectML debug layer (`DirectML.Debug.dll`): the DirectML debug layer provides enhanced validation for DirectML API usage, and the Direct3D 12 debug layer covers Direct3D 12 API usage. In practice, however, it is best to enable *both* debug layers when developing DirectML applications. The Direct3D 12 debug layer is installed as a part of the Graphics Tools FOD, which is explained above. Refer to [ID3D12Debug::EnableDebugLayer](#) for an example of how to activate the Direct3D 12 debug layer.

Important

You must first enable the Direct3D 12 debug layer. And *then* enable the DirectML debug layer by calling [DMLCreateDevice](#).

Enabling the DirectML debug layer

You can enable the DirectML debug layer by supplying [DML_CREATE_DEVICE_FLAG_DEBUG](#) when you call [DMLCreateDevice](#).

Once you've enabled the DirectML debug layer, any DirectML errors or invalid API calls will cause debugging information to be emitted as debug output. Here's an example.

Console

```
DML_OPERATOR_CONVOLUTION: invalid D3D12_HEAP_TYPE. DirectML requires all  
bound buffers to be D3D12_HEAP_TYPE_DEFAULT.
```

Up until [DML_FEATURE_LEVEL_5_2](#), it's a *requirement* to enable the Direct3D 12 debug layer in order to enable the DirectML debug layer. In earlier versions of DirectML, if the [DML_CREATE_DEVICE_FLAG_DEBUG](#) flag is specified in *flags* and the debug layers are not installed, then [DMLCreateDevice](#) returns [DXGI_ERROR_SDK_COMPONENT_MISSING](#). In newer versions of DirectML, messages are sent to [OutputDebugStringA](#) when [ID3D12InfoQueue](#) isn't available.

Code example

The following code illustrates enabling both the Direct3D 12 and DirectML debug layers for debug builds only.

C++

```
// By default, disable the DirectML debug layer.  
DML_CREATE_DEVICE_FLAGS dmlCreateDeviceFlags = DML_CREATE_DEVICE_FLAG_NONE;  
  
#if defined(_DEBUG)  
// If the project is in a debug build, then enable the Direct3D 12 debug  
layer.  
// This is optional (starting in DML_FEATURE_LEVEL_5_2) but strongly  
recommended!  
Microsoft::WRL::ComPtr<ID3D12Debug> debugController;  
if (SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(&debugController)))  
{  
    debugController->EnableDebugLayer();  
}  
  
// If the project is in a debug build, then enable debugging via DirectML
```

```
debug layers with this flag.  
dmlCreateDeviceFlags |= DML_CREATE_DEVICE_FLAG_DEBUG;  
#endif  
  
// Create the DirectML device.  
Microsoft::WRL::ComPtr<IDMLDevice> dmlDevice;  
THROW_IF_FAILED(DMLCreateDevice(  
    d3D12Device.Get(),  
    dmlCreateDeviceFlags,  
    IID_PPV_ARGS(&dmlDevice));
```

See also

- [Windows AI](#)
- [DMLCreateDevice function](#)
- [Available features-on-demand](#)
- [Use GPU-based validation with the Direct3D 12 Debug Layer](#)
- [Direct3D 12 Debug Layer reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Handling errors and device-removal in DirectML

Article • 02/10/2025

Device-removal

If an unrecoverable error occurs, the DirectML device may enter a "device-removed" state. Unrecoverable errors that cause device-removal include invalid API usage (for methods that don't return an [HRESULT](#)), driver error, hardware fault, or out-of-memory (OOM) conditions.

When a DirectML device is removed, all method calls on the device, and every object created by that device, become no-ops. For methods that return an [HRESULT](#), a [DXGI_ERROR_DEVICE_REMOVED](#) error code is returned. You can use the [IDMLDevice::GetDeviceRemovedReason](#) method to check whether the DirectML device has been removed, and to retrieve a more detailed error code.

You can't recover from device-removal except by releasing the affected device and all its children, then re-creating the DirectML device from scratch.

Device-removal of the underlying Direct3D 12 device also causes the DirectML device to be removed. However, the reverse is not true. DirectML device-removal may not necessarily cause the underlying Direct3D 12 device to become removed.

Debugging DirectML device-removal, and other errors

The most common cause of DirectML errors is invalid API usage. Invalid API usage might result in an [E_INVALIDARG](#) HRESULT error code, or it might result in device-removal.

We strongly recommend that you enable the [DirectML debug layer](#) during your development, in order to catch and debug such errors. The DirectML debug layer performs extensive validation of method parameters and API usage, and it will emit debug output messages to help you debug.

See also

- [Windows AI](#)

- Using the DirectML debug layer
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Enable PyTorch with DirectML on Windows

PyTorch with DirectML provides an easy-to-use way for developers to try out the latest and greatest AI models on their Windows machine. You can download PyTorch with DirectML by installing the [torch-directml](#) PyPi package. Once set up, you can start with our [samples](#) or use the AI Toolkit for VS Code.

Check your version of Windows

The [torch-directml](#) package on native Windows works starting with Windows 10, version 1709 (Build 16299 or higher). You can check your build version number by running `winver` via the Run command (Windows logo key + R).

Check for GPU driver updates

Ensure that you have the latest GPU driver installed. Select **Check for updates** in the **Windows Update** section of Windows **Settings**.

Set up Torch-DirectML

We recommend setting up a virtual Python environment inside Windows. There are many tools that you can use to set up a virtual Python environment—for these instructions, we'll use [Anaconda's Miniconda](#). The rest of this setup assumes that you use a Miniconda environment.

Set up a Python environment

Download and install the [Miniconda Windows installer](#) on your system. There's [additional guidance for setup](#) on Anaconda's site. Once Miniconda is installed, create an environment using Python named `pytdml`, and activate it through the following commands.

```
conda create --name pytdml -y  
conda activate pytdml
```

Install PyTorch and Torch-DirectML

Note

The **torch-directml** package supports up to PyTorch 2.3.1

All that is needed to get setup is installing the latest release of **torch-directml** by running the following command:

```
pip install torch-directml
```

Verification and Device Creation

Once you've installed the **torch-directml** package, you can verify that it runs correctly by adding two tensors. First start an interactive Python session, and import Torch with the following lines:

```
import torch
import torch_directml
dml = torch_directml.device()
```

The current release of **torch-directml** is mapped to the "PrivateUse1" Torch backend. The `torch_directml.device()` API is a convenient wrapper for sending your tensors to the DirectML device.

With the DirectML device created, you can now define two simple tensors; one tensor containing a 1 and another containing a 2. Place the tensors on the "dml" device.

```
tensor1 = torch.tensor([1]).to(dml) # Note that dml is a variable, not a string!
tensor2 = torch.tensor([2]).to(dml)
```

Add the tensors together, and print the results.

```
dml_algebra = tensor1 + tensor2
dml_algebra.item()
```

You should see the number 3 being output, as in the example below.

```
>>> import torch
>>> tensor1 = torch.tensor([1]).to(dml)
>>> tensor2 = torch.tensor([2]).to(dml)
>>> dml_algebra = tensor1 + tensor2
>>> dml_algebra.item()
3
```

PyTorch with DirectML samples and feedback

Check out [our samples](#) to see more uses of PyTorch with DirectML. If you run into issues, or have feedback on the PyTorch with DirectML package, then please [connect with our team here](#).

Last updated on 09/18/2025

Enable PyTorch with DirectML on WSL 2

Article • 02/10/2025

PyTorch with DirectML provides an easy-to-use way for developers to try out the latest and greatest AI models on their Windows machine. You can download PyTorch with DirectML by installing the [torch-directml](#) PyPi package. Once set up, you can start with our [samples](#) or use the AI Toolkit for VS Code.

Check your version of Windows

The [torch-directml](#) package in the Windows Subsystem for Linux (WSL) 2 works starting with Windows 11 (Build 22000 or higher). You can check your build version number by running `winver` via the **Run** command (Windows logo key + R).

Check for GPU driver updates

Ensure you have the latest GPU driver installed. Select **Check for updates** in the **Windows Update** section of the **Settings** app.

Set up Torch-DirectML

Install WSL 2

To install the Windows Subsystem for Linux (WSL) 2, see the instructions in [Install WSL](#).

Then install the WSL GUI driver by following the instructions in the `README.md` file in the [microsoft/wslg](#) GitHub repository.

Set up a Python environment

We recommend that you set up a virtual Python environment inside WSL 2. There are many tools that you can use to set up a virtual Python environment—in this topic we'll use Anaconda's [Miniconda](#). The rest of this setup assumes that you use a Miniconda environment.

Install Miniconda by following the [Linux installer guidance](#) on Anaconda's site, or by running the following commands in WSL 2.

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh  
bash Miniconda3-latest-Linux-x86_64.sh
```

Once Miniconda is installed, create a Python environment named **pytdml**, and activate it through the following commands:

```
conda create --name pytdml -y  
conda activate pytdml
```

Install PyTorch and Torch-DirectML

ⓘ Note

The **torch-directml** package supports up to PyTorch 2.3.1

All that is needed to get setup is installing the latest release of **torch-directml** by running the following command:

```
pip install torch-directml
```

Verification and Device Creation

Once you've installed the **torch-directml** package, you can verify that it runs correctly by adding two tensors. First start an interactive Python session, and import Torch with the following lines:

```
import torch  
import torch_directml  
dml = torch_directml.device()
```

The current release of **torch-directml** is mapped to the "PrivateUse1" Torch backend. The `torch_directml.device()` API is a convenient wrapper for sending your tensors to the DirectML device.

With the DirectML device created, you can now define two simple tensors; one tensor containing a 1 and another containing a 2. Place the tensors on the "dml" device.

```
tensor1 = torch.tensor([1]).to(dml) # Note that dml is a variable, not a string!
tensor2 = torch.tensor([2]).to(dml)
```

Add the tensors together, and print the results.

```
dml_algebra = tensor1 + tensor2
dml_algebra.item()
```

You should see the number 3 being output, as in the example below.

```
>>> import torch
>>> tensor1 = torch.tensor([1]).to(dml)
>>> tensor2 = torch.tensor([2]).to(dml)
>>> dml_algebra = tensor1 + tensor2
>>> dml_algebra.item()
3
```

PyTorch with DirectML samples and feedback

Check out [our samples](#) to see more uses of PyTorch with DirectML. If you run into issues, or have feedback on the PyTorch with DirectML package, then please [connect with our team here](#).

Feedback

Was this page helpful?

Yes

No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

WebNN Overview

Article • 02/10/2025

The Web Neural Network (WebNN) API is an emerging web standard that allows web apps and frameworks to accelerate deep neural networks with GPUs, CPUs, or purpose-built AI accelerators such as NPUs. The WebNN API leverages the DirectML API on Windows to access the native hardware capabilities and optimize the execution of neural network models.

As the use of AI/ML in apps become more popular, the WebNN API provides the following benefits:

- *Performance Optimizations* – By utilizing DirectML, WebNN helps to enable web apps and frameworks to take advantage of the best available hardware and software optimizations for each platform and device, without requiring complex and platform-specific code.
- *Low Latency* - In-browser inference helps enable novel use cases with local media sources, such as real-time video analysis, face detection, and speech recognition, without the need to send data to remote servers and wait for responses.
- *Privacy Preservation* - User data stays on-device and preserves user-privacy, as web apps and frameworks do not need to upload sensitive or personal information to cloud services for processing.
- *High Availability* - No reliance on the network after initial asset caching for offline case, as web apps and frameworks can run neural network models locally even when the internet connection is unavailable or unreliable.
- *Low Server Cost* - Computing on client devices means no servers needed, which helps web apps to reduce the operational and maintenance costs of running AI/ML services in the cloud.

AI/ML scenarios supported by WebNN include generative AI, person detection, face detection, semantic segmentation, skeleton detection, style transfer, super resolution, image captioning, machine translation, and noise suppression.

Note

The WebNN API is still in progress, with GPU and NPU support in a preview state. The WebNN API should not currently be used in a production environment.

Framework support

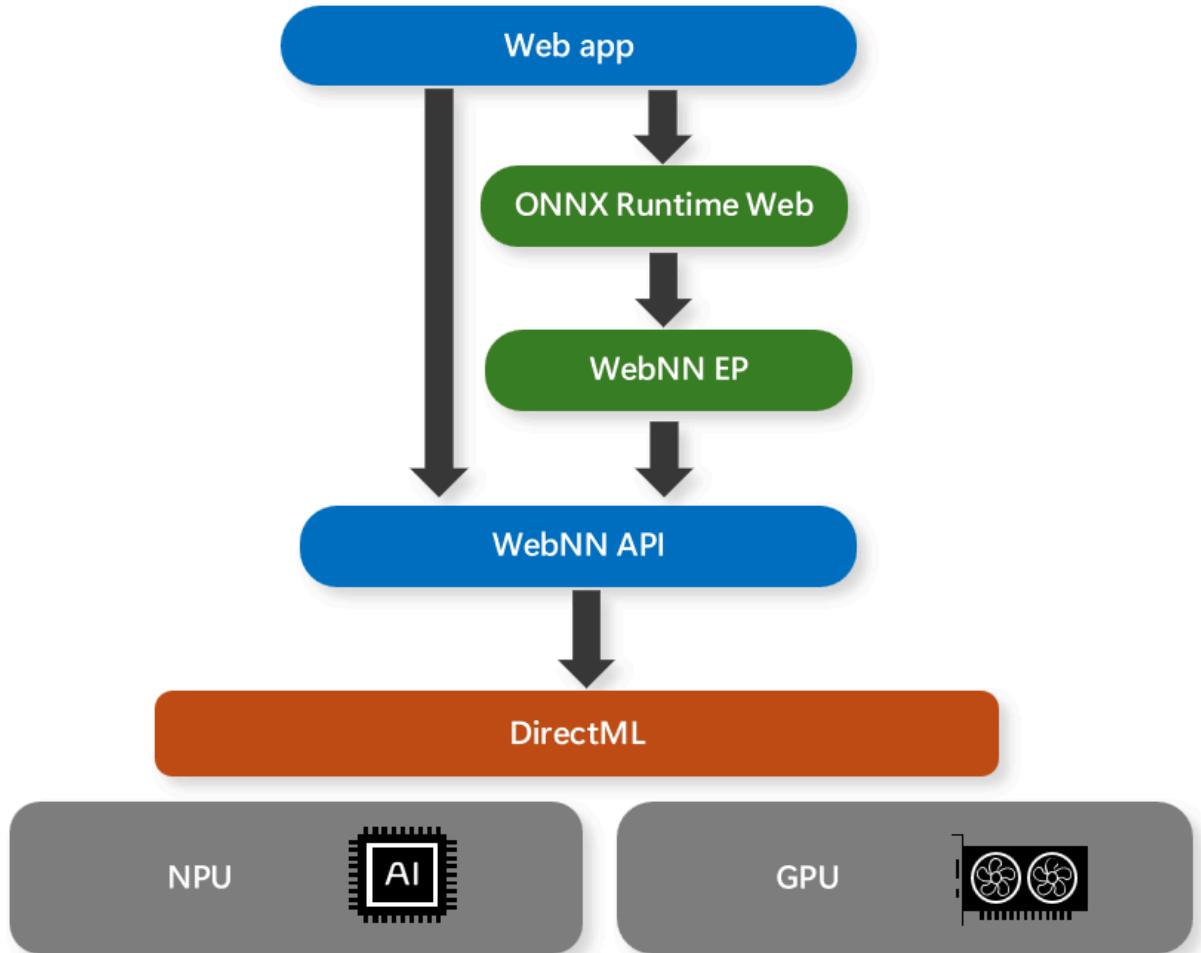
WebNN is designed as a backend API for web frameworks. For Windows, we recommend using [ONNX Runtime Web](#). This gives a familiar experience to using DirectML and ONNX Runtime natively so you can have a consistent experience deploying AI in ONNX format across web and native applications.

WebNN requirements

You can check information about your browser by navigating to `about://version` in your chromium browser's address bar.

[\[+\] Expand table](#)

Hardware	Web Browsers	Windows version	ONNX Runtime Web version	Driver Version
GPU	WebNN requires a Chromium browser*. Please use the most recent version of Microsoft Edge Beta.	Minimum version: Windows 11, version 21H2.	Minimum version: 1.18	Install the latest driver for your hardware.
NPU	WebNN requires a Chromium browser*. Please use the most recent version of Microsoft Edge Canary. See note below for how to disable the GPU blocklist.	Minimum version: Windows 11, version 21H2.	Minimum version: 1.18	Intel driver version: 32.0.100.2381. See FAQ for steps on how to update the driver.



ⓘ Note

Chromium based browsers can currently support WebNN, but will depend on the individual browser's implementation status.

ⓘ Note

For NPU support, launch edge from the command line with the following flag:

```
msedge.exe --disable_webnn_for_npu=0
```

Model support

GPU (Preview):

When running on GPUs, WebNN currently supports the following models:

- Stable Diffusion Turbo ↗
- Stable Diffusion 1.5 ↗

- [Whisper-base ↗](#)
- [MobileNetv2 ↗](#)
- [Segment Anything ↗](#)
- [ResNet ↗](#)
- [EfficientNet ↗](#)
- SqueezeNet

WebNN also works with custom models as long as operator support is sufficient. Check status of operators [here ↗](#).

NPU (Preview):

On Intel's® Core™ Ultra processors with Intel® AI Boost NPU, WebNN supports:

- [Whisper-base ↗](#)
- [MobileNetV2 ↗](#)
- [ResNet ↗](#)
- [EfficientNet ↗](#)

FAQ

How do I file an issue with WebNN?

For general issues with WebNN, please file an issue on our [WebNN Developer Preview GitHub ↗](#)

For issues with ONNX Runtime Web or the WebNN Execution Provider, go to the [ONNXRuntime Github ↗](#).

How do I debug issues with WebNN?

The [WebNN W3C Spec ↗](#) has information on error propagation, typically through DOM exceptions. The log at the end of about://gpu may also have helpful information. For further issues please file an issue as linked above.

Does WebNN support other operating systems?

Currently, WebNN best supports the Windows operating system. Versions for other operating systems are in progress.

What hardware back-ends are currently available? Are certain models only supported with specific hardware back-ends?

You can find information about operator support in WebNN at [Implementation Status of WebNN Operations | Web Machine Learning](#).

What are the steps to update the Intel driver for NPU Support (Preview)?

1. Locate the updated driver from [Intel's Driver Website](#).
2. Uncompress the ZIP file.
3. Press Win+R to open the Run dialog box.
4. Type devmgmt.msc into the text field.
5. Press Enter or click OK.
6. In the Device Manager, open the "Neural processors" node
7. Right click on the NPU who's driver you wish to update.
8. Select "Update Driver" from the context menu
9. Select "Browse my computer for drivers"
10. Select "Let me pick from a list of available drivers on my computer"
11. Press the "Have disk" button
12. Press the "Browse" button
13. Navigate to the place where you decompressed the aforementioned zip file.
14. Press OK.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

WebNN API Tutorial

Article • 02/10/2025

For an intro to WebNN, including information about operating system support, model support, and more, visit the [WebNN Overview](#).

This tutorial will show you how to use WebNN with ONNX Runtime Web to build an image classification system on the web that is hardware accelerated using on-device GPU. We will be leveraging the **MobileNetV2** model, which is an open-source model on [Hugging Face](#) used to classify images.

If you want to view and run the final code of this tutorial, you can find it on our [WebNN Developer Preview GitHub](#).

ⓘ Note

The WebNN API is a W3C Candidate Recommendation and is in early stages of a developer preview. Some functionality is limited. We have a list of current support and [implementation status](#).

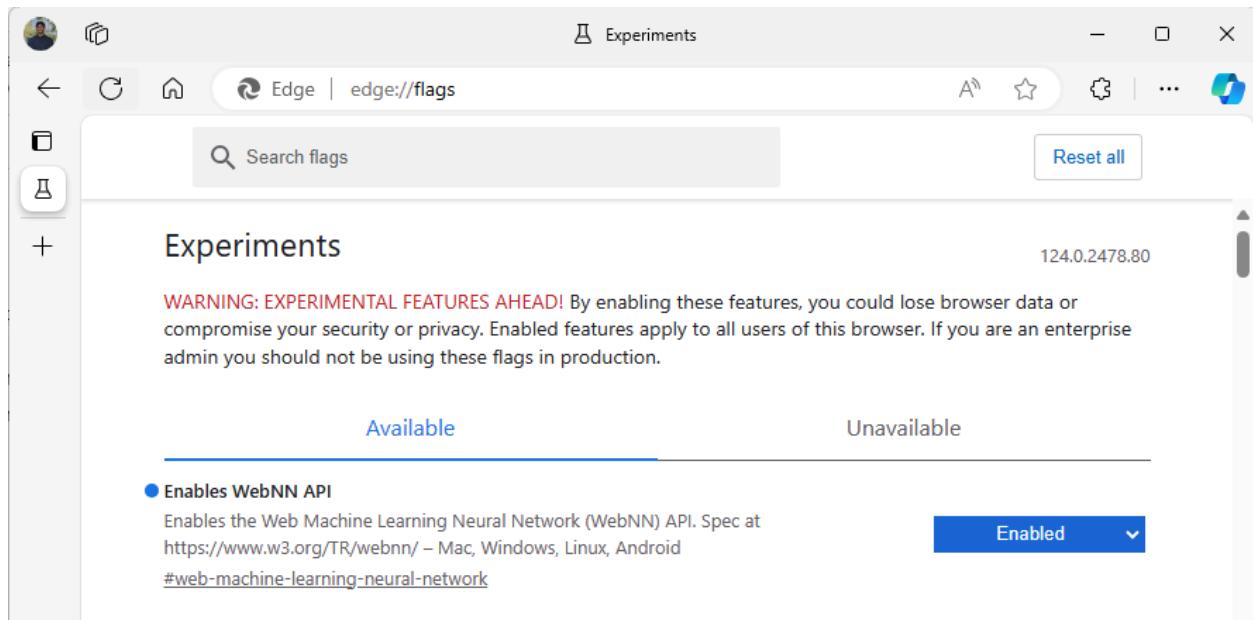
Requirements and set-up:

Setting Up Windows

Ensure you have the correct versions of Edge, Windows, and hardware drivers as detailed in the [WebNN Requirements section](#).

Setting Up Edge

1. Download and install [Microsoft Edge Dev](#).
2. Launch Edge Beta, and navigate to `about:flags` in the address bar.
3. Search for "WebNN API", click the dropdown, and set to 'Enabled'.
4. Restart Edge, as prompted.



Setting Up Developer Environment

1. Download and install [Visual Studio Code \(VSCode\)](#).
2. Launch VSCode.
3. Download and install the [Live Server extension for VSCode](#) within VSCode.
4. Select `File --> Open Folder`, and create a blank folder in your desired location.

Step 1: Initialize the web app

1. To begin, create a new `index.html` page. Add the following boilerplate code to your new page:

```
HTML

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Website</title>
  </head>
  <body>
    <main>
      <h1>Welcome to My Website</h1>
    </main>
  </body>
</html>
```

2. Verify the boilerplate code and developer setup worked by selecting the **Go Live** button at the bottom right hand side of VSCode. This should launch a local server in Edge Beta running the boilerplate code.
3. Now, create a new file called `main.js`. This will contain the javascript code for your app.
4. Next, create a subfolder off the root directory named `images`. Download and save any image within the folder. For this demo, we'll use the default name of `image.jpg`.
5. Download the **mobilenet** model from the [ONNX Model Zoo](#). For this tutorial, you'll be using the [mobilenet2-10.onnx](#) file. Save this model to the root folder of your web app.
6. Finally, download and save this [image classes file](#), `imagenetClasses.js`. This provides 1000 common classifications of images for your model to use.

Step 2: Add UI elements and parent function

1. Within the body of the `<main>` html tags you added in the previous step, replace the existing code with the following elements. These will create a button and display a default image.

HTML

```
<h1>Image Classification Demo!</h1>
<div></div>
<button onclick="classifyImage('./images/image.jpg')" type="button">Click
Me to Classify Image!</button>
<h1 id="outputText"> This image displayed is ... </h1>
```

2. Now, you'll add **ONNX Runtime Web** to your page, which is a JavaScript library you'll use to access the WebNN API. Within the body of the `<head>` html tags, add the following javascript source links.

HTML

```
<script src="./main.js"></script>
<script src="imagenetClasses.js"></script>
<script src="https://cdn.jsdelivr.net/npm/onnxruntime-web@1.18.0-
dev.20240311-5479124834/dist/ort.webgpu.min.js"></script>
```

3. Open your `main.js` file, and add the following code snippet.

JavaScript

```
async function classifyImage(pathToImage){  
  var imageTensor = await getImageTensorFromPath(pathToImage); // Convert  
image to a tensor  
  var predictions = await runModel(imageTensor); // Run inference on the  
tensor  
  console.log(predictions); // Print predictions to console  
  document.getElementById("outputText").innerHTML += predictions[0].name; //  
Display prediction in HTML  
}
```

Step 3: Pre-process data

1. The function you just added calls `getImageTensorFromPath`, another function you have to implement. You'll add it below, as well as another async function it calls to retrieve the image itself.

JavaScript

```
async function getImageTensorFromPath(path, width = 224, height = 224) {  
  var image = await loadImagefromPath(path, width, height); // 1. load the  
image  
  var imageTensor = imageDataToTensor(image); // 2. convert to tensor  
  return imageTensor; // 3. return the tensor  
}  
  
async function loadImagefromPath(path, resizedWidth, resizedHeight) {  
  var imageData = await Jimp.read(path).then(imageBuffer => { // Use Jimp  
to load the image and resize it.  
    return imageBuffer.resize(resizedWidth, resizedHeight);  
});  
  
  return imageData.bitmap;  
}
```

2. You also need to add the `imageDataToTensor` function that is referenced above, which will render the loaded image into a tensor format that will work with our ONNX model. This is a more involved function, though it might seem familiar if you've worked with similar image classification apps before. For an extended explanation, you can view [this ONNX tutorial ↗](#).

JavaScript

```
function imageDataToTensor(image) {  
  var imageBufferData = image.data;  
  let pixelCount = image.width * image.height;  
  const float32Data = new Float32Array(3 * pixelCount); // Allocate enough  
space for red/green/blue channels.
```

```

    // Loop through the image buffer, extracting the (R, G, B) channels,
    // rearranging from
    // packed channels to planar channels, and converting to floating point.
    for (let i = 0; i < pixelCount; i++) {
        float32Data[pixelCount * 0 + i] = imageBufferData[i * 4 + 0] / 255.0;
    }
    // Red
    float32Data[pixelCount * 1 + i] = imageBufferData[i * 4 + 1] / 255.0;
    // Green
    float32Data[pixelCount * 2 + i] = imageBufferData[i * 4 + 2] / 255.0;
    // Blue
    // Skip the unused alpha channel: imageBufferData[i * 4 + 3].
}
let dimensions = [1, 3, image.height, image.width];
const inputTensor = new ort.Tensor("float32", float32Data, dimensions);
return inputTensor;
}

```

Step 4: Call ONNX Runtime Web

1. You've now added all the functions needed to retrieve your image and render it as a tensor. Now, using the ONNX Runtime Web library that you loaded above, you'll run your model. Note that to use WebNN here, you simply specify

`executionProvider = "webnn"` - ONNX Runtime's support makes it very straightforward to enable WebNN.

JavaScript

```

async function runModel(preprocessedData) {
    // Set up environment.
    ort.env.wasm.numThreads = 1;
    ort.env.wasmsimd = true;
    // Uncomment for additional information in debug builds:
    // ort.env.wasm.proxy = true;
    // ort.env.logLevel = "verbose";
    // ort.env.debug = true;

    // Configure WebNN.
    const modelPath = "./mobilenetv2-10.onnx";
    const devicePreference = "gpu"; // Other options include "npu" and
    "cpu".
    const options = {
        executionProviders: [{ name: "webnn", deviceType: devicePreference,
        powerPreference: "default" }],
        freeDimensionOverrides: {"batch": 1, "channels": 3, "height": 224,
        "width": 224}
        // The key names in freeDimensionOverrides should map to the real
        input dim names in the model.
        // For example, if a model's only key is batch_size, you only need to
        set
    }
}
```

```

    // freeDimensionOverrides: {"batch_size": 1}
};

modelSession = await ort.InferenceSession.create(modelPath, options);

// Create feeds with the input name from model export and the
preprocessed data.
const feeds = {};
feeds[modelSession.inputNames[0]] = preprocessedData;
// Run the session inference.
const outputData = await modelSession.run(feeds);
// Get output results with the output name from the model export.
const output = outputData[modelSession.outputNames[0]];
// Get the softmax of the output data. The softmax transforms values to
be between 0 and 1.
var outputSoftmax = softmax(Array.prototype.slice.call(output.data));
// Get the top 5 results.
var results = imagenetClassesTopK(outputSoftmax, 5);

return results;
}

```

Step 5: Post-process data

- Finally, you'll add a `softmax` function, then add your final function to return the most likely image classification. The `softmax` transforms your values to be between 0 and 1, which is the probability form needed for this final classification.

First, add the following source files for helper libraries **Jimp** and **Lodash** in the head tag of `main.js`.

JavaScript

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/jimp/0.22.12/jimp.min.js"
integrity="sha512-
8xrUum7qKj8xbiUr0zDEJL5uLjpSIMxVevAM5pvBroaxJnxJGFsKaohQPmlzQP8rEoAxrAujWttT
nx3AMgGIww==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
<script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js">
</script>

```

Now, add these following functions to `main.js`.

JavaScript

```

// The softmax transforms values to be between 0 and 1.
function softmax(resultArray) {
    // Get the largest value in the array.
    const largestNumber = Math.max(...resultArray);

```

```

    // Apply the exponential function to each result item subtracted by the
    largest number, using reduction to get the
    // previous result number and the current number to sum all the
    exponentials results.
    const sumOfExp = resultArray
        .map(resultItem => Math.exp(resultItem - largestNumber))
        .reduce((prevNumber, currentNumber) => prevNumber + currentNumber);

    // Normalize the resultArray by dividing by the sum of all exponentials.
    // This normalization ensures that the sum of the components of the output
    vector is 1.
    return resultArray.map((resultValue, index) => {
        return Math.exp(resultValue - largestNumber) / sumOfExp
    });
}

function imagenetClassesTopK(classProbabilities, k = 5) {
    const probs = _.isTypedArray(classProbabilities)
        ? Array.prototype.slice.call(classProbabilities)
        : classProbabilities;

    const sorted = _.reverse(
        _.sortBy(
            probs.map((prob, index) => [prob, index]),
            probIndex => probIndex[0]
        )
    );

    const topK = _.take(sorted, k).map(probIndex => {
        const iClass = imagenetClasses[probIndex[1]]
        return {
            id: iClass[0],
            index: parseInt(probIndex[1].toString(), 10),
            name: iClass[1].replace(/\_/g, " "),
            probability: probIndex[0]
        }
    });
    return topK;
}

```

2. You've now added all the scripting needed to run image classification with WebNN in your basic web app. Using the Live Server extension for VS Code, you can now launch your basic webpage in-app to see the results of the classification for yourself.

Feedback

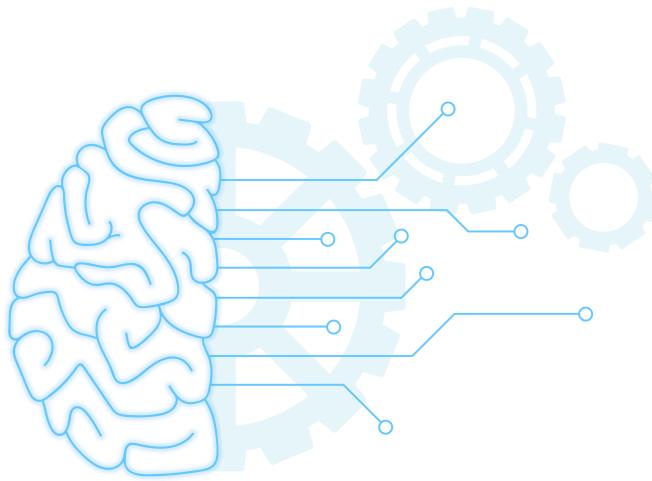
Was this page helpful?

 Yes

 No

GPU accelerated ML training

Article • 02/10/2025



This documentation covers setting up GPU accelerated machine learning (ML) training scenarios for the [Windows Subsystem for Linux](#) (WSL) and native Windows.

This functionality supports both professional and beginner scenarios. Below you'll find pointers to step-by-step guides on how to get your system set up depending on your level of expertise in ML, your GPU vendor, and the software library that you intend to use.

NVIDIA CUDA in WSL

If you're a professional data scientist who uses a native Linux environment day-to-day for inner-loop ML development and experimentation, and you have an NVIDIA GPU, then we recommend setting up [NVIDIA CUDA in WSL](#).

PyTorch with DirectML

To use PyTorch with a framework that works across the breadth of DirectX 12 capable GPUs, we recommend setting up the [PyTorch with DirectML](#) package. This package accelerates workflows on AMD, Intel, and NVIDIA GPUs.

If you're more familiar with a native Linux environment, then we recommend running [PyTorch with DirectML inside WSL](#).

If you're more familiar with Windows, then we recommend running [PyTorch with DirectML on native Windows](#).

TensorFlow with DirectML

ⓘ Important

This project is now discontinued, and isn't actively being worked on.

To use TensorFlow with a framework that works across the breadth of DirectX 12 capable GPUs, we recommend setting up the [TensorFlow with DirectML package](#). This package accelerates workflows on AMD, Intel, and NVIDIA GPUs.

Next steps

- [Enable NVIDIA CUDA in WSL](#)
- [Enable TensorFlow with DirectML](#)
- [Enable PyTorch with DirectML inside WSL](#)
- [Enable PyTorch with DirectML on native Windows](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Enable NVIDIA CUDA on WSL

Windows 11 and later updates of Windows 10 support running existing ML tools, libraries, and popular frameworks that use NVIDIA CUDA for GPU hardware acceleration inside a Windows Subsystem for Linux (WSL) instance. This includes PyTorch and TensorFlow as well as all the Docker and NVIDIA Container Toolkit support available in a native Linux environment.

Install Windows 11 or Windows 10, version 21H2

To use these features, you can download and install [Windows 11](#) or [Windows 10, version 21H2](#).

Install the GPU driver

Download and install the [NVIDIA CUDA enabled driver for WSL](#) to use with your existing CUDA ML workflows. For more info about which driver to install, see:

- [Getting Started with CUDA on WSL 2](#)
- [CUDA on Windows Subsystem for Linux \(WSL\)](#)

Install WSL

Once you've installed the above driver, ensure you [enable WSL](#) and [install a glibc-based distribution](#), such as Ubuntu or Debian. Ensure you have the latest kernel by selecting **Check for updates** in the **Windows Update** section of Windows Settings.

ⓘ Note

Ensure you have **Receive updates for other Microsoft products** enabled. You can find it in **Advanced options** within the **Windows Update** section of Windows Settings.

For these features, you need a kernel version of 5.10.43.3 or higher. You can check the version number by running the following command in PowerShell.

```
PowerShell
```

```
wsl cat /proc/version
```

Get started with NVIDIA CUDA

Now follow the instructions in the [NVIDIA CUDA on WSL User Guide](#) and you can start using your existing Linux workflows through [NVIDIA Docker](#), or by installing [PyTorch](#) or [TensorFlow](#) inside WSL.

Share feedback on NVIDIA's support via their [Community forum for CUDA on WSL](#).

Last updated on 09/18/2025

Enable GPU acceleration for TensorFlow 2 with tensorflow-directml-plugin

Article • 02/10/2025

ⓘ Important

This project is now discontinued, and isn't actively being worked on.

This release provides students, beginners, and professionals a way to run machine learning (ML) training on their existing DirectX 12-enabled hardware by using the DirectML Plugin for TensorFlow 2.

ⓘ Note

You can install `tensorflow-directml-plugin` by using Python x86-64 3.10. But `tensorflow-directml-plugin` isn't supported for version 3.11 and later.

Learn how to configure your device to run and train models with the GPU using `tensorflow-directml-plugin`.

STEP 1: Minimum (and maximum) system requirements

Before installing the TensorFlow-DirectML-Plugin, ensure your version of Windows or WSL supports TensorFlow-DirectML-Plugin.

Windows native

- Windows 10 Version 1709, 64-bit (Build 16299 or higher) or Windows 11 Version 21H2, 64-bit (Build 22000 or higher)
- Python x86-64 3.7, 3.8, 3.9, or 3.10. Version 3.10 is also the *maximum* supported version.
- One of the following supported GPUs:
 - AMD Radeon R5/R7/R9 2xx series or newer
 - Intel HD Graphics 5xx or newer
 - NVIDIA GeForce GTX 9xx series GPU or newer

Windows Subsystem for Linux

- Windows 10 Version 21H2, 64-bit (Build 20150 or higher) or Windows 11 Version 21H2, 64-bit (Build 22000 or higher)
- Python x86-64 3.7, 3.8, 3.9 or 3.10. Version 3.10 is also the *maximum* supported version.
- One of the following supported GPUs:
 - AMD Radeon R5/R7/R9 2xx series or newer, and [20.20.01.05 driver or newer ↗](#)
 - Intel HD Graphics 6xx or newer, and [28.20.100.8322 driver or newer ↗](#)
 - NVIDIA GeForce GTX 9xx series GPU or newer, and [460.20 driver or newer ↗](#)

Install the latest GPU driver

Ensure that you have the latest GPU driver installed for your hardware. Select **Check for updates** in the **Windows Update** section of the **Settings** app. If needed, pick up an install from your hardware vendor using the above links.

STEP 2: Configure your Windows environment

Windows native

The **TensorFlow-DirectML-Plugin** package on native Windows works starting with Windows 10, version 1709 (Build 16299 or higher). You can check your build version number by running `winver` via the **Run** command (Windows logo key + R).

Windows Subsystem for Linux

Once you've installed the above driver, ensure you [enable WSL](#) and [install a glibc-based distribution](#) (such as Ubuntu or Debian). For our testing, we used Ubuntu. Ensure you have the latest kernel by selecting **Check for updates** in the **Windows Update** section of the **Settings** app.

Note

Ensure you have the **Receive updates for other Microsoft products when you update Windows** option enabled. You can find it in **Advanced options** within the **Windows Update** section of the **Settings** app.

For these features, you need a kernel version of 5.10.43.3 or higher. You can check the version number by running the following command in PowerShell.

```
wsl cat /proc/version
```

STEP 3: Set up your environment

We recommend setting up a virtual Python environment inside Windows. There are many tools that you can use to set up a virtual Python environment—for these instructions, we'll use [Anaconda's Miniconda](#). The rest of this setup assumes that you use a Miniconda environment. [Learn more about using python environments](#)

Create an environment in Miniconda

Download and install the [Miniconda Windows installer](#) on your system. There is [additional guidance for setup](#) on Anaconda's site. Once Miniconda is installed, create an environment using Python named `tfdml_plugin`, and activate it through the following commands.

```
conda create --name tfdml_plugin python=3.9  
conda activate tfdml_plugin
```

ⓘ Note

tensorflow version >= 2.9 and python version >= 3.7 supported.

STEP 4: Install base TensorFlow

Download the base TensorFlow package. Currently the directml-plugin only works with `tensorflow-cpu==2.10` and not `tensorflow` or `tensorflow-gpu`.

```
pip install tensorflow-cpu==2.10
```

STEP 5: Install tensorflow-directml-plugin

Installing this package automatically enables the DirectML backend for existing scripts without any code changes.

```
pip install tensorflow-directml-plugin
```

⚠ Note

If your training scripts hardcode the device string to something other than "GPU", that might throw errors.

Alternatively, the package can be built from the source. [Instructions for building tensorflow-directml-plugin from source ↗](#).

TensorFlow with DirectML samples and feedback

Check out [our samples ↗](#) or use your existing model scripts. If you run into issues, or have feedback on the TensorFlow-DirectML-Plugin package, then please [connect with our team ↗](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

FAQ

How do I enable DirectML acceleration?

The DirectML device is enabled by default, assuming you have an appropriate DirectX 12 GPU available. TensorFlow operations will automatically be assigned to the DirectML device if possible.

If you're having trouble determining whether your model is running using DirectML acceleration or not, you can put `tf.debugging.set_log_device_placement(True)` as the first statement of your program and TensorFlow will print device placement information to the console.

How do I control device placement of specific operations?

As with any other device (see [TensorFlow Guide: Use a GPU](#)), you can use `tf.device()` to control which device to run on.

For running the TensorFlow 2 with DirectML backend using the TensorFlow-DirectML-Plugin, the device string is `'GPU'`, and will automatically override any other devices. To change the device name you can [build tensorflow-directml-plugin from source](#).

For TensorFlow-DirectML 1.15, the device string is `'DML'`.

Python

```
import tensorflow as tf

tf.debugging.set_log_device_placement(True)

tf.enable_eager_execution()

# Explicitly place tensors on the DirectML device

with tf.device('/DML:0'):

    a = tf.constant([1.0, 2.0, 3.0])

    b = tf.constant([4.0, 5.0, 6.0])
```

```
c = tf.add(a, b)
```

```
print(c)
```

```
Executing op Add in device /job:localhost/replica:0/task:0/device:DML:0
```

```
tf.Tensor([5. 7. 9.], shape=(3,), dtype=float32)
```

I have multiple GPUs. How do I select which one is used by DirectML?

There are a couple of different ways to do this, depending on whether you want to control it process-wide or per-session (or both).

If you want to control which devices are visible to TensorFlow process-wide, use the `DML_VISIBLE_DEVICES` environment variable. If you want to control it on a per-session basis, use `tf.GPUOptions.visible_device_list`.

How do I use the `DML_VISIBLE_DEVICES` environment variable to control which GPU(s) get used by DirectML?

TensorFlow with DirectML supports a `DML_VISIBLE_DEVICES` environment variable, which takes the form of a comma-separated list of device IDs (also known as "adapter indices".) When set, only the device IDs in that list will be visible to TensorFlow process-wide. Devices excluded using `DML_VISIBLE_DEVICES` will not appear in the list of physical devices available to TensorFlow.

Python

```
import tensorflow as tf
tf.debugging.set_log_device_placement(True)
tf.enable_eager_execution()

a = tf.constant([1.])
b = tf.constant([2.])
c = tf.add(a, b)
print(c)
```

Here is example output **without** `DML_VISIBLE_DEVICES` set:

```
DirectML device enumeration: found 2 compatible adapters.  
DirectML: creating device on adapter 0 (NVIDIA TITAN V)  
DirectML: creating device on adapter 1 (AMD Radeon RX 5700 XT)  
Executing op Add in device /job:localhost/replica:0/task:0/device:DML:0  
tf.Tensor([3.], shape=(1,), dtype=float32)
```

With `DML_VISIBLE_DEVICES="1"`:

```
DirectML device enumeration: found 1 compatible adapters.  
DirectML: creating device on adapter 0 (AMD Radeon RX 5700 XT)  
Executing op Add in device /job:localhost/replica:0/task:0/device:DML:0  
tf.Tensor([3.], shape=(1,), dtype=float32)
```

Note that by restricting the visible devices to only index 1 (AMD Radeon RX 5700 XT), TensorFlow now assigns an ID of 0 to this device, and makes it the default.

You may also re-order devices using this environment variable. For example, setting

`DML_VISIBLE_DEVICES="1,0"`:

```
DirectML device enumeration: found 2 compatible adapters.  
DirectML: creating device on adapter 0 (AMD Radeon RX 5700 XT)  
DirectML: creating device on adapter 1 (NVIDIA TITAN V)  
Executing op Add in device /job:localhost/replica:0/task:0/device:DML:0  
tf.Tensor([3.], shape=(1,), dtype=float32)
```

Notice that the two GPUs (NVIDIA TITAN V and AMD Radeon RX 5700 XT) have now switched places.

To prevent specific devices from being visible, you can supply an invalid ID (e.g. `-1`) in the comma-separated list. All device IDs after the invalid entry are ignored. You can also use this to disable DirectML acceleration altogether.

`DML_VISIBLE_DEVICES="-1"`:

```
DirectML device enumeration: found 0 compatible adapters.  
Executing op Add in device /job:localhost/replica:0/task:0/device:CPU:0  
tf.Tensor([3.], shape=(1,), dtype=float32)
```

How do I use the `visible_device_list` session option to control which GPU(s) DirectML uses to run the session?

Similar to `DML_VISIBLE_DEVICES`, you can also set a similar string to control visible devices at a session level. The `visible_device_list` attribute is available on the `GPUOptions` class when creating your TensorFlow session.

Python

```
import tensorflow as tf

a = tf.constant([1.])
b = tf.constant([2.])
c = tf.add(a, b)

gpu_config = tf.GPUOptions()
gpu_config.visible_device_list = "1"

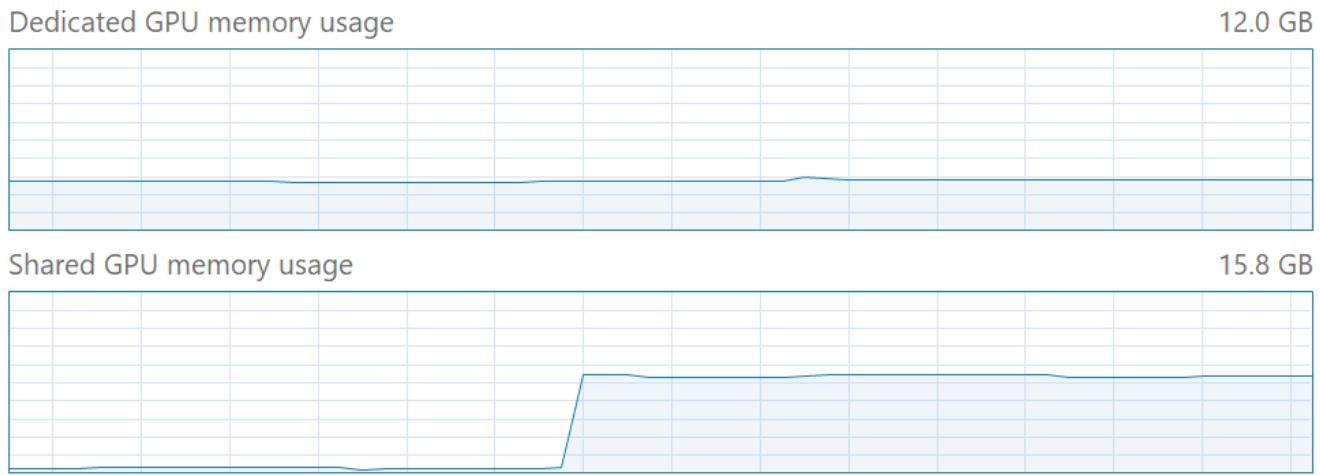
session = tf.Session(config=tf.ConfigProto(gpu_options=gpu_config))
print(session.run(c))
```

```
DirectML device enumeration: found 2 compatible adapters.
DirectML: creating device on adapter 1 (AMD Radeon RX 5700 XT)
[3.]
```

You can read the [TensorFlow GPUOptions API reference](#) for more details.

Why does tensorflow-directml have high shared GPU memory usage?

When training large models, you might notice high usage of shared GPU memory in Task Manager:



This is normal. TensorFlow-DirectML uses shared GPU memory as a staging area for upload and readback of tensor data to and from the GPU. Because of this, some increase in shared GPU memory utilization is expected.

Note that DirectML will always use dedicated GPU memory (for example, onboard VRAM) in preference to system memory, if available. However, some usage of shared GPU memory is still expected depending on how much staging memory DirectML requires.

Why doesn't DirectML utilize all of my available dedicated GPU memory?

Some devices, like the CUDA device, will by default reserve a large proportion of available dedicated GPU memory at startup.

The DirectML device, however, allocates memory on demand rather than reserving it up front. This behavior avoids starving other system processes of GPU memory, but can occasionally cause a slightly higher memory overhead.

If you prefer to reserve memory up front, then you can control that by setting the [TF_FORCE_GPU_ALLOW_GROWTH](#) environment variable to `false`.

For more information about the environment variables used by tensorflow-directml, see [Environment variables](#).

I'm seeing DXGI_ERROR_DEVICE_REMOVED or DEVICE_HUNG errors. How do I resolve these?

Please refer to the instructions on GitHub: [Troubleshooting GPU timeouts in tensorflow-directml](#).

Why am I getting device assignment or node colocation errors?

If you are seeing errors similar to the following:

```
tensorflow.python.framework.errors_impl.InvalidArgumentError: Cannot assign a device for operation
```

```
tensorflow/core/common_runtime/colocation_graph.cc:983] Failed to place the graph without changing the devices of some resources. Some of the operations (that had to be colocated with resource generating operations) are not supported on the resources' devices. Current candidate devices are [...]
```

This usually means that you're attempting to use an operator that's unsupported by the DirectML device, or a combination of operator and data type that's unsupported. For a full list of supported operators, see [Roadmap \(operators\)](#) on our wiki.

DirectML sample applications

Important

DirectML is in sustained engineering. DirectML continues to be supported, but new feature development has moved to Windows ML for Windows-based ONNX Runtime deployments. Windows ML provides the same ONNX Runtime APIs while dynamically selecting the best execution provider based on your hardware. See [What is Windows ML](#) to learn more.

For DirectML sample applications, including a sample of a minimal DirectML application, see [DirectML samples](#).

For a sample demonstrating how to use Olive—a powerful tool you can use to optimize DirectML performance—see [Stable diffusion optimization with DirectML](#).

For samples with the ONNX Generate() API for Generative AI models, please visit: [ONNX Runtime Generate\(\) API](#)

See also

- [Windows AI](#)

Last updated on 01/09/2026

DirectML reference

Article • 02/10/2025

This section covers Direct Machine Learning (DirectML) APIs declared in `DirectML.h`.

In this section

[] [Expand table](#)

Topic	Description
DirectML interfaces	The following interfaces, and their methods, are declared in <code>directml.h</code> .
DirectML functions	The following free functions are declared in <code>directml.h</code> .
DirectML structures	The following structures are declared in <code>directml.h</code> .
DirectML enumerations	The following enumerations are declared in <code>directml.h</code> .
DirectML constants	The following constants are declared in <code>directml.h</code> .

Related topics

- [Core reference](#)
- [Windows AI](#)
- [Direct3D 12 Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DirectML interfaces

Article • 02/10/2025

The following interfaces are declared in `DirectML.h`.

In this section

[] Expand table

Topic	Description
IDMLBindingTable	Creates a DirectML device for a given Direct3D 12 device.
IDMLCommandRecorder	Records dispatches of DirectML work into a Direct3D 12 command list.
IDMLCompiledOperator	Represents a compiled, efficient form of an operator suitable for execution on the GPU.
IDMLDebugDevice	Controls the DirectML debug layer.
IDMLDevice	Represents a DirectML device, which is used to create operators, binding tables, command recorders, and other objects.
IDMLDevice1	Represents a DirectML device, which is used to create operators, binding tables, command recorders, and other objects.
IDMLDeviceChild	An interface implemented by all objects created from the DirectML device.
IDMLDispatchable	Implemented by objects that can be recorded into a command list for dispatch on the GPU, using IDMLCommandRecorder::RecordDispatch .
IDMLObject	An interface from which IDMLDevice and IDMLDeviceChild inherit directly (and all other interfaces, indirectly). Consequently, it provides methods common to all DirectML interfaces, specifically methods to associate private data, and to annotate object names.
IDMLOperator	Represents a DirectML operator.
IDMLOperatorInitializer	Represents a specialized object whose purpose is to initialize compiled operators.
IDMLPageable	Implemented by objects that can be evicted from GPU memory, and hence that can be supplied to IDMLDevice::Evict and IDMLDevice::MakeResident .

Related topics

- DirectML reference
 - Windows AI
 - Core reference
 - Direct3D 12 Reference
-

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLBindingTable interface (directml.h)

Article02/16/2023

Wraps a range of an application-managed descriptor heap, and is used by DirectML to create bindings for resources. To create this object, call [IDMLDevice::CreateBindingTable](#). The **IDMLBindingTable** interface inherits from [IDMLDeviceChild](#).

The binding table is created over a range of CPU and GPU descriptor handles. When an **IDMLBindingTable::Bind*** method is called, DirectML writes one or more descriptors into the range of CPU descriptors. When you use the binding table during a call to [IDMLCommandRecorder::RecordDispatch](#), DirectML binds the corresponding GPU descriptors to the pipeline.

The CPU and GPU descriptor handles aren't required to point to the same entries in a descriptor heap, however it is then your application's responsibility to ensure that the entire descriptor range referred to by the CPU descriptor handle is copied into the range referred to by the GPU descriptor handle prior to execution using this binding table.

It is your application's responsibility to perform correct synchronization between the CPU and GPU work that uses this binding table. For example, you must take care not to overwrite the bindings created by the binding table (for example, by calling **Bind*** again on the binding table, or by overwriting the descriptor heap manually) until all work using the binding table has completed execution on the GPU. In addition, since the binding table doesn't maintain a reference on the descriptor heap it writes into, you must not release the backing shader-visible descriptor heap until all work using that binding table has completed execution on the GPU.

The binding table is associated with exactly one dispatchable object (an operator initializer, or a compiled operator), and represents the bindings for that particular object. You can reuse a binding table by calling [IDMLBindingTable::Reset](#), however. Note that since the binding table doesn't own the descriptor heap itself, it is safe to call **Reset** and reuse the binding table for a different dispatchable object even before any outstanding executions have completed on the GPU.

The binding table doesn't keep strong references on any resources bound using it—your application must ensure that resources are not deleted while still in use by the GPU.

This object is not thread safe—your application must not call methods on the binding table simultaneously from different threads without synchronization.

Inheritance

The [IDMLBindingTable](#) interface inherits from the [IDMLDeviceChild](#) interface.

Methods

The [IDMLBindingTable](#) interface has these methods.

[+] [Expand table](#)

IDMLBindingTable::BindInputs	Binds a set of resources as input tensors.
IDMLBindingTable::BindOutputs	Binds a set of resources as output tensors.
IDMLBindingTable::BindPersistentResource	Binds a buffer as a persistent resource. You can determine the required size of this buffer range by calling IDMLDispatchable::GetBindingProperties .
IDMLBindingTable::BindTemporaryResource	Binds a buffer to use as temporary scratch memory. You can determine the required size of this buffer range by calling IDMLDispatchable::GetBindingProperties .
IDMLBindingTable::Reset	Resets the binding table to wrap a new range of descriptors, potentially for a different operator or initializer. This allows dynamic reuse of the binding table.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[Binding in DirectML](#)

[IDMLDeviceChild](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLBindingTable::BindInputs method (directml.h)

Article 01/21/2022

Binds a set of resources as input tensors.

If binding for a compiled operator, the number of bindings must exactly match the number of inputs of the operator, including optional tensors. This can be determined from the operator description used to create the operator. If too many or too few bindings are provided, device removal will occur. For optional tensors, you may use [DML_BINDING_TYPE_NONE](#) to specify 'no binding'. Otherwise, the binding type must match the tensor type when the operator was created.

For operator initializers, input bindings are expected to be of type [DML_BINDING_TYPE_BUFFER_ARRAY](#) with one input binding per operator to initialize, supplied in the order that you specified the operators during creation or reset of the initializer. Each buffer array should have a size equal to the number of inputs of its corresponding operator to initialize. Input tensors that had the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set should be bound during initialize—otherwise, nothing should be bound for that tensor. If there is nothing to be bound as input for initialization of an operator (that is, there are no tensors with the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set) then you may supply `nullptr` or an empty [DML_BUFFER_ARRAY_BINDING](#) to indicate 'no binding'.

To unbind all input resources, supply a *rangeCount* of 0, and a value of `nullptr` for *bindings*.

If an input tensor has the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set, it may only be bound when executing an operator initializer. Otherwise, if the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag is not set, the opposite is true—the input tensor must not be bound when executing the initializer, but must be bound when executing the operator itself.

All buffers being bound as input must have heap type [D3D12_HEAP_TYPE_DEFAULT](#), except when the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag is set. If the [DML_TENSOR_FLAG OWNED_BY_DML](#) is set for a tensor that is being bound as input for an initializer, the buffer's heap type may be either [D3D12_HEAP_TYPE_DEFAULT](#) or [D3D12_HEAP_TYPE_UPLOAD](#).

Multiple bindings are permitted to reference the same [ID3D12Resource](#) in some cases; however, you should take care when an operator simultaneously reads and writes to the

same region of a resource. A potential binding hazard exists when: a pair of bindings reference the same **ID3D12Resource**, and at least one of the bindings is involved in writing, and the buffer regions intersect (overlap by at least one byte). Binding hazards are validated using the following rules, as of DirectML 1.7.0:

- When binding for initialization, an input binding can never reference the same resource as the output binding—inputs are copied into the output resource (the future persistent resource for execution), and copying might require a resource state transition.
- When binding for execution, an input binding may reference the same resource as an output binding; however, the respective binding ranges can intersect only if the regions are identical *and* the operator supports in-place execution.
- If present, a persistent binding must not intersect with any output binding or temporary binding.
- If present, a temporary binding must not intersect any input binding, output binding, or persistent binding.

The above rules assume that two resources don't alias the same region of a heap, so extra caution is required when using placed or reserved resources.

Syntax

C++

```
void BindInputs(  
    UINT bindingCount,  
    [in, optional] const DML_BINDING_DESC *bindings  
);
```

Parameters

`bindingCount`

Type: **UINT**

This parameter determines the size of the *bindings* array (if provided).

[in, optional] `bindings`

Type: **const DML_BINDING_DESC***

An optional pointer to a constant array of **DML_BINDING_DESC** containing descriptions of the tensor resources to bind.

Return value

None

Remarks

Binding hazard examples

In the examples below, the rectangles represent a buffer resource referenced by at least one binding in a binding table. Each row indicates the range of bytes potentially read (R) or written (W) by the named binding. All examples assume that resources don't share the same physical memory.

Example 1

This example shows an input and output binding referencing different resources. A pair of bindings that reference different resources is never a hazard, so this is always a valid binding.

	Resource A	Resource B
Input 0:	RRRRRRRRRRRRRR	
Output 0:		WWWWWWWWWWWWWW

Example 2

This example shows an input and output binding referencing different regions of the same resource. A pair of bindings with non-overlapping regions of the same resource is not a hazard when binding for execution. This is a valid binding when binding for execution, but it's invalid when binding for initialization.

	Resource A
Input 0:	RRRRRRR
Output 0:	WWWWWWWW

Example 3

This example shows two inputs bindings that overlap ranges. A pair of read-only bindings (input bindings and persistent bindings) can never be a hazard, regardless of any buffer-region intersection, so this is always a valid binding.

	Resource A	Resource B
Input 0:	RRRRRRRRRR	
Input 1:	RRRRRRRRRR	
Output 0:		WWWWWWWWWWWWWWWWW
	+-----+	+-----+

Example 4

This example shows an input and output binding with identical regions. This binding is valid if and only if the operator being bound supports in-place execution and the binding is for execution.

	Resource A
Input 0:	RRRRRRRRRRRRRRR
Output 0:	WWWWWWWWWWWWWWW
	+-----+

Example 5

The following binding is never valid, regardless of the operator's in-place execution support, because it involves a pair of bindings with overlapping regions that are not identical.

	Resource A
Input 0:	RRRRRRRRRRRR
Output 0:	WWWWWWWWWWWWW
	+-----+

Requirements

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLBindingTable::BindOutputs method (directml.h)

Article 02/22/2024

Binds a set of resources as output tensors.

If binding for a compiled operator, the number of bindings must exactly match the number of inputs of the operator, including optional tensors. This can be determined from the operator description used to create the operator. If too many or too few bindings are provided, device removal will occur. For optional tensors, you may use [DML_BINDING_TYPE_NONE](#) to specify 'no binding'. Otherwise, the binding type must match the tensor type when the operator was created.

For operator initializers, the output bindings are the persistent resources of each operator, supplied in the order the operators were given when creating or resetting the initializer. If a particular operator does not require a persistent resource, you should prove an empty binding in that slot.

To unbind all input resources, supply a *rangeCount* of 0, and a value of `nullptr` for *bindings*.

The writable areas of two output tensors must not overlap with one another. The 'writable area' of an output buffer being bound is defined as being the start offset of the buffer range, up to the *TotalTensorSizeInBytes* as specified in the tensors description.

All buffers being bound as output must have heap type [D3D12_HEAP_TYPE_DEFAULT](#).

Syntax

C++

```
void BindOutputs(
    UINT bindingCount,
    [in, optional] const DML_BINDING_DESC *bindings
);
```

Parameters

`bindingCount`

Type: [UINT](#)

This parameter determines the size of the *bindings* array (if provided).

[in, optional] `bindings`

Type: `const DML_BINDING_DESC*`

An optional pointer to a constant array of `DML_BINDING_DESC` containing descriptions of the tensor resources to bind.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLBindingTable::BindPersistentResource method (directml.h)

Article 10/13/2021

Binds a buffer as a persistent resource. You can determine the required size of this buffer range by calling [IDMLDispatchable::GetBindingProperties](#).

If the binding properties for the operator specify a size of zero for the persistent resource, then you may supply `nullptr` to this method (which indicates no resource to bind). Otherwise, a binding of type `DML_BINDING_TYPE_BUFFER` must be supplied that is at least as large as the required `PersistentResourceSize` returned by [IDMLDispatchable::GetBindingProperties](#).

Unlike the temporary resource, the persistent resource's contents and lifetime must persist as long as the compiled operator does. That is, if an operator requires a persistent resource, then your application must supply it during initialization and subsequently also supply it to all future executes of the operator without modifying its contents.

The persistent resource is typically used by DirectML to store lookup tables or other long-lived data that is computed during initialization of an operator and reused on future executions of that operator.

As the persistent resource's data is opaque, once initialized it cannot be copied or moved to another buffer.

The persistent resource is only written to during initialization of an operator and is thereafter immutable; all subsequent executions are guaranteed not to write to the persistent resource.

The supplied buffer range to be bound as the persistent buffer must have its start offset aligned to `DML_PERSISTENT_BUFFER_ALIGNMENT`. The type of the heap underlying the buffer must be `D3D12_HEAP_TYPE_DEFAULT`.

Syntax

C++

```
void BindPersistentResource(  
    [in, optional] const DML_BINDING_DESC *binding  
);
```

Parameters

[in, optional] binding

Type: [const DML_BINDING_DESC*](#)

An optional pointer to a [DML_BINDING_DESC](#) containing the description of a tensor resource to bind.

Return value

None

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLBindingTable::BindTemporaryResource method (directml.h)

Article 02/22/2024

Binds a buffer to use as temporary scratch memory. You can determine the required size of this buffer range by calling [IDMLDispatchable::GetBindingProperties](#).

If the binding properties for the [IDMLDispatchable](#) specify a size of zero for the temporary resource, then you may supply `nullptr` to this method (which indicates no resource to bind). Otherwise, a binding of type [DML_BINDING_TYPE_BUFFER](#) must be supplied that is at least as large as the required [TemporaryResourceSize](#) returned by [IDMLDispatchable::GetBindingProperties](#).

The temporary resource is typically used as scratch memory during execution of an operator. The contents of a temporary resource need not be defined prior to execution. For example, DirectML doesn't require that you zero the contents of the temporary resource prior to binding or executing an operator.

You don't need to preserve the contents of the temporary buffer, and your application is free to overwrite or reuse its contents as soon as execution of an operator or initializer completes on the GPU. This is in contrast to a persistent resource, whose contents must be preserved and lifetime extended for the lifetime of the operator.

The supplied buffer range to be bound as the temporary buffer must have its start offset aligned to [DML_TEMPORARY_BUFFER_ALIGNMENT](#). The type of the heap underlying the buffer must be [D3D12_HEAP_TYPE_DEFAULT](#).

Syntax

C++

```
void BindTemporaryResource(  
    [in, optional] const DML_BINDING_DESC *binding  
);
```

Parameters

[in, optional] binding

Type: [const DML_BINDING_DESC*](#)

An optional pointer to a [DML_BINDING_DESC](#) containing the description of a tensor resource to bind.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLBindingTable::Reset method (directml.h)

Article 02/22/2024

Resets the binding table to wrap a new range of descriptors, potentially for a different operator or initializer. This allows dynamic reuse of the binding table.

Resetting a binding table doesn't modify any previous bindings created by the table. Because of this, it is safe to reset the binding table immediately after supplying it to [IDMLCommandRecorder::RecordDispatch](#), even if that work has not yet completed execution on the GPU, so long as the underlying descriptors remain valid.

See [IDMLDevice::CreateBindingTable](#) for more information on the parameters supplied to this method.

Syntax

C++

```
HRESULT Reset(  
    [in, optional] const DML_BINDING_TABLE_DESC *desc  
);
```

Parameters

[in, optional] desc

Type: [const DML_BINDING_TABLE_DESC*](#)

An optional pointer to a [DML_BINDING_TABLE_DESC](#) containing the binding table parameters. This may be `nullptr`, indicating an empty binding table.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLCommandRecorder interface (directml.h)

Article 02/22/2024

Records dispatches of DirectML work into a Direct3D 12 command list. The **IDMLCommandRecorder** interface inherits from [IDMLDeviceChild](#).

The command recorder is a stateless object whose purpose is to record commands into a Direct3D 12 command list. DirectML doesn't create command lists, command allocators, nor command queues; nor does it directly submit any work for execution on the GPU. Instead, your application manages its own command lists and queues, and it uses the **IDMLCommandRecorder** to record work into its existing command lists. You're then responsible for executing the command list on a queue of your choice.

This object is thread-safe.

Inheritance

The **IDMLCommandRecorder** interface inherits from the [IDMLDeviceChild](#) interface.

Methods

The **IDMLCommandRecorder** interface has these methods.

[+] Expand table

[IDMLCommandRecorder::RecordDispatch](#)

Records execution of a dispatchable object (an operator initializer, or a compiled operator) onto a command list.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	directml.h

See also

[IDMLDeviceChild](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IDMLCommandRecorder::RecordDispatch method (directml.h)

Article 10/05/2021

Records execution of a dispatchable object (an operator initializer, or a compiled operator) onto a command list.

This method doesn't submit the execution to the GPU; it merely records it onto the command list. You are responsible for closing the command list and submitting it to the Direct3D 12 command queue.

Prior to execution of this call on the GPU, all resources bound must be in the [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#) state, or a state implicitly promotable to [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#), such as

[D3D12_RESOURCE_STATE_COMMON](#). After this call completes, the resources remain in the [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#) state. The only exception to this is for upload heaps bound when executing an operator initializer and while one or more tensors has the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set. In that case, any upload heaps bound for input must be in the [D3D12_RESOURCE_STATE_GENERIC_READ](#) state and will remain in that state, as required by all upload heaps.

This method resets the following state on the command list.

- Compute root signature
- Pipeline state

No other command list state is modified.

Although this method takes a binding table representing the resources to bind to the pipeline, it doesn't set the descriptor heaps containing the descriptors themselves. Therefore, your application is responsible for calling [ID3D12GraphicsCommandList::SetDescriptorHeaps](#) to bind the correct descriptor heaps to the pipeline.

If [DML_EXECUTION_FLAG DESCRIPTORS_VOLATILE](#) was not set when compiling the operator, then all bindings must be set on the binding table before [RecordDispatch](#) is called, otherwise the behavior is undefined. Otherwise, if the [_DESCRIPTORS_VOLATILE](#) flag is set, binding of resources may be deferred until the Direct3D 12 command list is submitted to the command queue for execution.

This method acts logically like a call to [ID3D12GraphicsCommandList::Dispatch](#). As such, unordered access view (UAV) barriers are necessary to ensure correct ordering if there

are data dependencies between dispatches. This method does not insert UAV barriers on input nor output resources. Your application must ensure that the correct UAV barriers are performed on any inputs if their contents depend on an upstream dispatch, and on any outputs if there are downstream dispatches that depend on those outputs.

This method doesn't hold references to any of the interfaces passed in. It is your responsibility to ensure that the [IDMLDispatchable](#) object is not released until all dispatches using it have completed execution on the GPU.

Syntax

C++

```
void RecordDispatch(  
    ID3D12CommandList *commandList,  
    IDMLDispatchable  *dispatchable,  
    IDMLBindingTable *bindings  
) ;
```

Parameters

`commandList`

Type: [ID3D12CommandList*](#)

A pointer to an [ID3D12CommandList](#) interface representing the command list to record the execution into. The command list must be open and must have type [D3D12_COMMAND_LIST_TYPE_DIRECT](#) or [D3D12_COMMAND_LIST_TYPE_COMPUTE](#).

`dispatchable`

Type: [IDMLDispatchable*](#)

A pointer to an [IDMLDispatchable](#) interface representing the object (an operator initializer, or a compiled operator) whose execution will be recorded into the command list.

`bindings`

Type: [IDMLBindingTable*](#)

A pointer to an [IDMLBindingTable](#) interface representing the bindings to use for executing the dispatchable object. If the

`DML_EXECUTION_FLAG_DESCRIPTOR_VOLATILE` flag was not set, then you must fill out all required bindings, otherwise an error will result.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLCommandRecorder](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLCompiledOperator interface (directml.h)

Article 02/22/2024

Represents a compiled, efficient form of an operator suitable for execution on the GPU. To create this object, call [IDMLDevice::CompileOperator](#). The **IDMLCompiledOperator** interface inherits from [IDMLDispatchable](#).

Unlike [IDMLOperator](#), compiled operators are "baked", and can be executed directly by the GPU. After an operator is compiled, you must initialize it exactly once before it can be executed. It's an error to initialize an operator more than once. Operator initializers are used to initialize compiled operators. You can use

[IDMLCommandRecorder::RecordDispatch](#) to record the dispatch of an operator initializer which, when executed on the GPU, will initialize one or more operators.

In addition to input and output tensors, operators may require additional memory for execution. This additional memory must be provided by your application in the form of temporary and persistent resources.

A temporary resource is scratch memory that is only used during the execution of the operator, and doesn't need to persist after the call to

[IDMLCommandRecorder::RecordDispatch](#) completes on the GPU. This means that your application may release or overwrite the temporary resource in between dispatches of the compiled operator. In contrast, the persistent resource must live at least until the last execute of the operator has completed on the GPU. Additionally, the contents of the persistent resource are opaque and must be preserved between executions of the operator.

The size of the temporary and persistent resources varies per operator. Call [IDMLDispatchable::GetBindingProperties](#) to query the required size, in bytes, of the persistent and temporary resources for this compiled operator. See [IDMLBindingTable::BindTemporaryResource](#) and [IDMLBindingTable::BindPersistentResource](#) for more information on binding temporary and persistent resources.

All methods on this interface are thread-safe.

Inheritance

The **IDMLCompiledOperator** interface inherits from the [IDMLDispatchable](#) interface.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[Binding in DirectML](#)

[IDMLDispatchable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IDMLDebugDevice interface (directml.h)

Article 02/22/2024

Controls the DirectML debug layers. The **IDMLDebugDevice** interface inherits from the **IUnknown** interface.

This interface may be retrieved from the **IDMLDevice** via **QueryInterface**. This interface is available only if the DirectML device was created with the **DML_CREATE_DEVICE_FLAG_DEBUG** flag. DirectML sends messages to the **ID3D12InfoQueue** associated with the **ID3D12Device** passed in at **DMLCreateDevice**; the Direct3D 12 info queue can be retrieved via **QueryInterface** on the **ID3D12Device**.

This object is thread-safe.

Inheritance

The **IDMLDebugDevice** interface inherits from the **IUnknown** interface.

Methods

The **IDMLDebugDevice** interface has these methods.

[+] Expand table

[IDMLDebugDevice::SetMuteDebugOutput](#)

Determine whether to mute DirectML from sending messages to the **ID3D12InfoQueue**.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDebugDevice::SetMuteDebugOutput method (directml.h)

Article 02/22/2024

Determine whether to mute DirectML from sending messages to the [ID3D12InfoQueue](#).

Syntax

C++

```
void SetMuteDebugOutput(  
    BOOL mute  
>;
```

Parameters

mute

Type: [BOOL](#)

If **TRUE**, DirectML is muted, and it will not send messages to the [ID3D12InfoQueue](#). If **FALSE**, DirectML is not muted, and it will send messages to the [ID3D12InfoQueue](#). The default value is **FALSE**.

Return value

None

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib

Requirement	Value
DLL	DirectML.dll

See also

[IDMLDebugDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IDMLDevice interface (directml.h)

Article 02/16/2023

Represents a DirectML device, which is used to create operators, binding tables, command recorders, and other objects. The **IDMLDevice** interface inherits from [IDMLObject](#).

A DirectML device is always associated with exactly one underlying Direct3D 12 device. All objects created by the DirectML device maintain a strong reference to their parent device. Unlike the Direct3D 12 device, the DML device is not a singleton. Therefore, it's possible to create multiple DirectML devices over the same Direct3D 12 device. However, this isn't recommended as the DirectML device has no mutable state, so there's little advantage to creating multiple DML devices over the same Direct3D 12 device.

This object is thread-safe.

Inheritance

The **IDMLDevice** interface inherits from the [IDMLObject](#) interface.

Methods

The **IDMLDevice** interface has these methods.

[+] [Expand table](#)

IDMLDevice::CheckFeatureSupport
Gets information about the optional features and capabilities that are supported by the DirectML device.
IDMLDevice::CompileOperator
Compiles an operator into an object that can be dispatched to the GPU.
IDMLDevice::CreateBindingTable
Creates a binding table, which is an object that can be used to bind resources (such as tensors) to the pipeline.

[IDMLDevice::CreateCommandRecorder](#)

Creates a DirectML command recorder.

[IDMLDevice::CreateOperator](#)

Creates a DirectML operator.

[IDMLDevice::CreateOperatorInitializer](#)

Creates an object that can be used to initialize compiled operators.

[IDMLDevice::Evict](#)

Evicts one or more pageable objects from GPU memory. Also see [IDMLDevice::MakeResident](#).

[IDMLDevice::GetDeviceRemovedReason](#)

Retrieves the reason that the DirectML device was removed.

[IDMLDevice::GetParentDevice](#)

Retrieves the Direct3D 12 device that was used to create this DirectML device.

[IDMLDevice::MakeResident](#)

Causes one or more pageable objects to become resident in GPU memory. Also see [IDMLDevice::Evict](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[IDMLObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::CheckFeatureSupport method (directml.h)

Article02/22/2024

Gets information about the optional features and capabilities that are supported by the DirectML device.

Syntax

C++

```
HRESULT CheckFeatureSupport(  
    DML_FEATURE feature,  
    UINT      featureQueryDataSize,  
    [in, optional] const void *featureQueryData,  
    UINT      featureSupportDataSize,  
    [out]      void      *featureSupportData  
) ;
```

Parameters

`feature`

Type: [DML_FEATURE](#)

A constant from the [DML_FEATURE](#) enumeration describing the feature(s) that you want to query for support.

`featureQueryDataSize`

Type: [UINT](#)

The size of the structure pointed to by the *featureQueryData* parameter, if provided, otherwise 0.

`[in, optional] featureQueryData`

Type: [const void*](#)

An optional pointer to a query structure that corresponds to the value of the *feature* parameter. To determine the corresponding query type for each constant, see [DML_FEATURE](#).

`featureSupportDataSize`

Type: [UINT](#)

The size of the structure pointed to by the *featureSupportData* parameter.

`[out] featureSupportData`

Type: [void*](#)

A pointer to a support data structure that corresponds to the value of the *feature* parameter. To determine the corresponding support data type for each constant, see [DML_FEATURE](#).

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns [DXGI_ERROR_UNSUPPORTED](#) if the [DML_FEATURE](#) is unrecognized or unsupported, and [E_INVALIDARG](#) if the parameters are incorrect.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

IDMLDevice::CompileOperator method (directml.h)

Article 02/22/2024

Compiles an operator into an object that can be dispatched to the GPU.

A compiled operator represents the efficient, baked form of an operator suitable for execution on the GPU. A compiled operator holds state (such as shaders and other objects) required for execution. Because a compiled operator implements the [IDMLPageable](#) interface, you're able to evict one from GPU memory if you wish. See [IDMLDevice::Evict](#) and [IDMLDevice::MakeResident](#) for more info.

The compiled operator maintains a strong reference to the supplied [IDMLOperator](#) pointer.

Syntax

C++

```
HRESULT CompileOperator(
    IDMLOperator      *op,
    DML_EXECUTION_FLAGS flags,
    REFIID            riid,
    [out] void        **ppv
);
```

Parameters

op

Type: [IDMLOperator*](#)

The operator (created with [IDMLDevice::CreateOperator](#)) to compile.

flags

Type: [DML_EXECUTION_FLAGS](#)

Any flags to control the execution of this operator.

riid

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in *ppv*. This is expected to be the GUID of [IDMLCompiledOperator](#).

[out] *ppv*

Type: **void****

A pointer to a memory block that receives a pointer to the compiled operator. This is the address of a pointer to an [IDMLCompiledOperator](#), representing the compiled operator created.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

Feedback

Was this page helpful?

 Yes

 No

IDMLDevice::CreateBindingTable method (directml.h)

Article 10/13/2021

Creates a binding table, which is an object that can be used to bind resources (such as tensors) to the pipeline.

The binding table wraps a range of an application-managed descriptor heap using the provided descriptor handles and count. Binding tables are used by DirectML to manage the binding of resources by writing descriptors into the descriptor heap at the offset specified by the **CPUDescriptorHandle**, and binding those descriptors to the pipeline using the descriptors at the offset specified by the **GPUDescriptorHandle**. The order in which DirectML writes descriptors into the heap is unspecified, so your application must take care not to overwrite the descriptors wrapped by the binding table.

The supplied CPU and GPU descriptor handles may come from different heaps, however it is then your application's responsibility to ensure that the entire descriptor range referred to by the CPU descriptor handle is copied into the range referred to by the GPU descriptor handle prior to execution using this binding table.

The descriptor heap from which the handles are supplied must have type [D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV](#). Additionally, the heap referred to by the **GPUDescriptorHandle** must be a shader-visible descriptor heap.

You must not delete the heap referred to by the GPU descriptor handle until all work referencing it has completed execution on the GPU. You may, however, reset or release the binding table itself as soon as the dispatch has been recorded into the command list. Similar to the relationship between [ID3D12CommandList](#) and [ID3D12CommandAllocator](#), the [IDMLBindingTable](#) doesn't own the underlying memory referenced by the descriptor handles. Rather, the [ID3D12DescriptorHeap](#) does. Therefore, you're permitted to reset or release a DirectML binding table before work using the binding table has completed execution on the GPU.

Syntax

C++

```
HRESULT CreateBindingTable(
    [in, optional] const DML_BINDING_TABLE_DESC *desc,
    REFIID riid,
```

```
[out] void **ppv  
);
```

Parameters

[in, optional] desc

Type: **const DML_BINDING_TABLE_DESC***

An optional pointer to a [DML_BINDING_TABLE_DESC](#) containing the binding table parameters. This may be `nullptr`, indicating an empty binding table.

riid

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in `ppv`. This is expected to be the GUID of [IDMLBindingTable](#).

[out] ppv

Type: **void****

A pointer to a memory block that receives a pointer to the binding table. This is the address of a pointer to an [IDMLBindingTable](#), representing the binding table created.

Return value

Type: **HRESULT**

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib

Requirement	Value
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::CreateCommandRecorder method (directml.h)

Article02/22/2024

Creates a DirectML command recorder.

A command recorder allows your application to record the initialization and execution of compiled operators into existing Direct3D 12 command lists. The command recorder is a stateless object: it does not own command lists or operators, nor does it execute any GPU work. Instead, it merely records the commands necessary for dispatching initialization or execution into an application-supplied command list. Your application is then responsible for submitting the execution of that command list to the Direct3D 12 command queue.

Syntax

C++

```
HRESULT CreateCommandRecorder(
    REFIID riid,
    [out] void    **ppv
);
```

Parameters

riid

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in *ppv*. This is expected to be the GUID of [IDMLCommandRecorder](#).

[out] ppv

Type: [void**](#)

A pointer to a memory block that receives a pointer to the command recorder. This is the address of a pointer to an [IDMLCommandRecorder](#), representing the command recorder created.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::CreateOperator method (directml.h)

Article 10/13/2021

Creates a DirectML operator.

In DirectML, an operator represents an abstract bundle of functionality, which can be compiled into a form suitable for execution on the GPU. Operator objects cannot be executed directly; they must first be compiled into an [IDMLCompiledOperator](#).

Syntax

C++

```
HRESULT CreateOperator(
    const DML_OPERATOR_DESC *desc,
    REFIID                   riid,
    [out] void                **ppv
);
```

Parameters

`desc`

Type: [const DML_OPERATOR_DESC*](#)

The description of the operator to be created.

`riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in `ppv`. This is expected to be the GUID of [IDMLOperator](#).

`[out] ppv`

Type: [void**](#)

A pointer to a memory block that receives a pointer to the operator. This is the address of a pointer to an [IDMLOperator](#), representing the operator created.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

[IDMLDevice::CompileOperator](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::CreateOperatorInitializer method (directml.h)

Article 10/13/2021

Creates an object that can be used to initialize compiled operators.

Once compiled, an operator must be initialized exactly once on the GPU before it can be executed. The operator initializer holds the state necessary for initialization of one or more target compiled operators.

Once instantiated, dispatch of the operator initializer can be recorded in a command list via [IDMLCommandRecorder::RecordDispatch](#). After execution completes on the GPU, all compiled operators that are targets of the initializer enter the initialized state.

An operator initializer can be reused to initialize different sets of compiled operators.

See [IDMLOperatorInitializer::Reset](#) for more info.

An operator initializer can be created with no target operators. Executing such an initializer is a no-op. Creating an operator initializer with no target operators may be useful if you wish to create an initializer up-front, but don't yet know which operators it will be used to initialize. [IDMLOperatorInitializer::Reset](#) can be used to reset which operators to target.

Syntax

C++

```
HRESULT CreateOperatorInitializer(
    UINT          operatorCount,
    [in, optional] IDMLCompiledOperator * const *operators,
    REFIID        riid,
    [out]         void      **ppv
);
```

Parameters

operatorCount

Type: [UINT](#)

This parameter determines the number of elements in the array passed in the *operators* parameter.

[in, optional] operators

Type: [IDMLCompiledOperator](#)*

An optional pointer to a constant array of [IDMLCompiledOperator](#) pointers containing the set of operators that this initializer will target. Upon execution of the initializer, the target operators become initialized. This array may be null or empty, indicating that the initializer has no target operators.

riid

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in *ppv*. This is expected to be the GUID of [IDMLOperatorInitializer](#).

[out] ppv

Type: [void](#)**

A pointer to a memory block that receives a pointer to the operator initializer. This is the address of a pointer to an [IDMLOperatorInitializer](#), representing the operator initializer created.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib

Requirement	Value
DLL	DirectML.dll

See also

[IDMLDevice](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::Evict method (directml.h)

Article 02/22/2024

Evicts one or more pageable objects from GPU memory. Also see [IDMLDevice::MakeResident](#).

Syntax

C++

```
HRESULT Evict(
    UINT          count,
    [in] IDMLPageable * const *ppObjects
);
```

Parameters

count

Type: [UINT](#)

This parameter determines the number of elements in the array passed in the *ppObjects* parameter.

[in] ppObjects

Type: [IDMLPageable*](#)

A pointer to a constant array of [IDMLPageable](#) pointers containing the pageable objects to evict from GPU memory.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

[IDMLDevice::MakeResident](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::GetDeviceRemovedReason method (directml.h)

Article 02/22/2024

Retrieves the reason that the DirectML device was removed.

Syntax

C++

```
HRESULT GetDeviceRemovedReason();
```

Return value

Type: [HRESULT](#)

An [HRESULT](#) containing the reason that the device was removed, or [S_OK](#) if the device has not been removed.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::GetParentDevice method (directml.h)

Article 02/22/2024

Retrieves the Direct3D 12 device that was used to create this DirectML device.

Syntax

C++

```
HRESULT GetParentDevice(
    REFIID riid,
    [out] void    **ppv
);
```

Parameters

riid

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in *ppv*. This is expected to be the GUID of [ID3D12Device](#).

[out] ppv

Type: [void**](#)

A pointer to a memory block that receives a pointer to the device. This is the address of a pointer to an [ID3D12Device](#), representing the device.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Requirements

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice::MakeResident method (directml.h)

Article 02/22/2024

Causes one or more pageable objects to become resident in GPU memory. Also see [IDMLDevice::Evict](#).

Syntax

C++

```
HRESULT MakeResident(
    UINT          count,
    [in] IDMLPageable * const *ppObjects
);
```

Parameters

`count`

Type: `UINT`

This parameter determines the number of elements in the array passed in the `ppObjects` parameter.

`[in] ppObjects`

Type: [IDMLPageable*](#)

A pointer to a constant array of [IDMLPageable](#) pointers containing the pageable objects to make resident in GPU memory.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

[IDMLDevice::Evict](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice1 interface (directml.h)

Article 02/16/2023

Represents a DirectML device, which is used to create operators, binding tables, command recorders, and other objects. The [IDMLDevice1](#) interface inherits from [IDMLDevice](#).

A DirectML device is always associated with exactly one underlying Direct3D 12 device. All objects created by the DirectML device maintain a strong reference to their parent device. Unlike the Direct3D 12 device, the DML device is not a singleton. Therefore, it's possible to create multiple DirectML devices over the same Direct3D 12 device. However, this isn't recommended as the DirectML device has no mutable state, so there's little advantage to creating multiple DML devices over the same Direct3D 12 device.

This object is thread-safe.

Availability

This API was introduced in DirectML version [1.1.0](#).

Tensor constraints

Target Platform: Windows

Inheritance

The [IDMLDevice1](#) interface inherits from the [IDMLDevice](#) interface.

Methods

The [IDMLDevice1](#) interface has these methods.

[+] [Expand table](#)

[IDMLDevice1::CompileGraph](#)

Compiles a graph of DirectML operators into an object that can be dispatched to the GPU.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	directml.h

See also

[IDMLDevice interface](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDevice1::CompileGraph method (directml.h)

Article 10/20/2021

Compiles a graph of DirectML operators into an object that can be dispatched to the GPU.

A compiled operator represents the efficient, baked form of an operator suitable for execution on the GPU. A compiled operator holds state (such as shaders and other objects) required for execution. Because a compiled operator implements the [IDMLPageable](#) interface, you're able to evict one from GPU memory if you wish. See [IDMLDevice1::Evict](#) and [IDMLDevice1::MakeResident](#) for more info.

The compiled operator doesn't use nor reference the [IDMLOperator](#) objects supplied within the graph description after this method returns.

Syntax

C++

```
HRESULT CompileGraph(  
    const DML_GRAPH_DESC *desc,  
    DML_EXECUTION_FLAGS flags,  
    REFIID riid,  
    [out] void **ppv  
>;
```

Parameters

`desc`

Type: [DML_GRAPH_DESC*](#)

A description of the graph to compile. See [DML_GRAPH_DESC](#).

`flags`

Type: [DML_EXECUTION_FLAGS](#)

Any flags to control the execution of this operator.

`riid`

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in *ppv*. This is expected to be the GUID of [IDMLCompiledOperator](#).

[out] *ppv*

Type: **void****

A pointer to a memory block that receives a pointer to the compiled operator. This is the address of a pointer to an [IDMLCompiledOperator](#), representing the compiled operator created.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Remarks

The DirectML operator graph API provides an abstract way to use DirectML efficiently across diverse hardware. DirectML applies tensor-level optimizations such as choosing the most efficient tensor layout based on the adapter being used. It also applies optimizations such as the removal of Join or Split operators.

We recommend that you apply high-level optimizations before building a DirectML graph. For example, fusing Convolution operators with BatchNorm, constant folding, and common subexpression elimination. The optimizations within DirectML's graph optimizer are intended to complement such device-independent optimizations, which are typically handled generically by machine learning frameworks.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows

Requirement	Value
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDeviceChild interface (directml.h)

Article 02/22/2024

An interface implemented by all objects created from the DirectML device. The **IDMLDeviceChild** interface inherits from [IDMLObject](#).

Inheritance

The **IDMLDeviceChild** interface inherits from the [IDMLObject](#) interface.

Methods

The **IDMLDeviceChild** interface has these methods.

[+] [Expand table](#)

IDMLDeviceChild::GetDevice
Retrieves the DirectML device that was used to create this object.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[IDMLObject](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

IDMLDeviceChild::GetDevice method (directml.h)

Article 02/22/2024

Retrieves the DirectML device that was used to create this object.

Syntax

C++

```
HRESULT GetDevice(  
    REFIID riid,  
    [out] void    **ppv  
>;
```

Parameters

`riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in `ppv`. This is expected to be the GUID of [IDMLDevice](#).

`[out] ppv`

Type: `void**`

A pointer to a memory block that receives a pointer to the DirectML device. This is the address of a pointer to an [IDMLDevice](#), representing the DirectML device.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDeviceChild](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDispatchable interface (directml.h)

Article 02/22/2024

Implemented by objects that can be recorded into a command list for dispatch on the GPU, using [IDMLCommandRecorder::RecordDispatch](#). The **IDMLDispatchable** interface inherits from [IDMLPageable](#).

This interface is implemented by [IDMLCompiledOperator](#) and [IDMLOperatorInitializer](#).

Inheritance

The **IDMLDispatchable** interface inherits from the [IDMLPageable](#) interface.

Methods

The **IDMLDispatchable** interface has these methods.

[] [Expand table](#)

IDMLDispatchable::GetBindingProperties
Retrieves the binding properties for a dispatchable object (an operator initializer, or a compiled operator).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[IDMLPageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLDispatchable::GetBindingProperties method (directml.h)

Article 10/05/2021

Retrieves the binding properties for a dispatchable object (an operator initializer, or a compiled operator). The binding properties value contains the required size of the binding table in descriptors, as well as the required size in bytes of the temporary and persistent resources required to execute this object.

When called on an operator initializer, the binding properties of the object may be different if retrieved both before and after a call to [IDMLOperatorInitializer::Reset](#).

Syntax

C++

```
DML_BINDING_PROPERTIES GetBindingProperties();
```

Return value

Type: [DML_BINDING_PROPERTIES](#)

A [DML_BINDING_PROPERTIES](#) value containing binding properties.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLObject interface (directml.h)

Article 02/22/2024

An interface from which [IDMLDevice](#) and [IDMLDeviceChild](#) inherit directly (and all other interfaces, indirectly). Consequently, it provides methods common to all DirectML interfaces, specifically methods to associate private data, and to annotate object names. The [IDMLObject](#) interface inherits from the [IUnknown](#) interface.

Inheritance

The [IDMLObject](#) interface inherits from the [IUnknown](#) interface.

Methods

The [IDMLObject](#) interface has these methods.

[+] Expand table

IDMLObject::GetPrivateData
Gets application-defined data from a DirectML device object.
IDMLObject::SetName
Associates a name with the DirectML device object. This name is for use in debug diagnostics and tools.
IDMLObject::SetPrivateData
Sets application-defined data to a DirectML device object, and associates that data with an application-defined GUID.
IDMLObject::SetPrivateDataInterface
Associates an IUnknown-derived interface with the DirectML device object, and associates that interface with an application-defined GUID.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLObject::GetPrivateData method (directml.h)

Article 10/13/2021

Gets application-defined data from a DirectML device object. This method is thread-safe.

Syntax

C++

```
HRESULT GetPrivateData(  
    [in]          REFGUID guid,  
    [in, out]      UINT    *dataSize,  
    [out, optional] void   *data  
) ;
```

Parameters

[in] guid

Type: [REFGUID](#)

The GUID that is associated with the data.

[in, out] dataSize

Type: [UINT*](#)

A pointer to a variable that on input contains the size, in bytes, of the buffer that *data* points to, and on output contains the size, in bytes, of the amount of data that [GetPrivateData](#) retrieved.

[out, optional] data

Type: [void*](#)

A pointer to a memory block that receives the data from the device object if *dataSize* points to a value that specifies a buffer large enough to hold the data.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Remarks

If the data returned is a pointer to an [IUnknown](#) (or derived interface) that was previously set by [SetPrivateDataInterface](#), then that interface will have its reference count incremented before the private data is returned.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IDMLObject::SetName method (directml.h)

Article 02/22/2024

Associates a name with the DirectML device object. This name is for use in debug diagnostics and tools. This method is thread-safe.

Syntax

C++

```
HRESULT SetName(  
    PCWSTR name  
);
```

Parameters

name

Type: [PCWSTR](#)

A NULL-terminated **UNICODE** string that contains the name to associate with the DirectML device object.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h

Requirement	Value
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLObject::SetPrivateData method (directml.h)

Article 02/22/2024

Sets application-defined data to a DirectML device object, and associates that data with an application-defined **GUID**. This method is thread-safe.

Syntax

C++

```
HRESULT SetPrivateData(
    REFGUID guid,
    [in]        UINT   dataSize,
    [in, optional] const void *data
);
```

Parameters

guid

Type: [REFGUID](#)

The **GUID** to associate with the data.

[in] dataSize

Type: [UINT](#)

The size in bytes of the data.

[in, optional] data

Type: [const void*](#)

A pointer to a memory block that contains the data to be stored with this DirectML device object. If *data* is **NULL**, then *dataSize* must be 0, and any data that was previously associated with the **GUID** specified in *guid* will be destroyed.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	<code>directml.h</code>
Library	<code>DirectML.lib</code>
DLL	<code>DirectML.dll</code>

See also

[IDMLObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLObject::SetPrivateDataInterface method (directml.h)

Article 02/22/2024

Associates an [IUnknown](#)-derived interface with the DirectML device object, and associates that interface with an application-defined **GUID**. This method is thread-safe.

Syntax

C++

```
HRESULT SetPrivateDataInterface(
    [in]           REFGUID guid,
    [in, optional] IUnknown *data
);
```

Parameters

`[in] guid`

Type: [REFGUID](#)

The **GUID** to associate with the interface.

`[in, optional] data`

Type: [const IUnknown*](#)

A pointer to the [IUnknown](#)-derived interface to be associated with the device object.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLOperator interface (directml.h)

Article02/22/2024

Represents a DirectML operator. The **IDMLOperator** interface inherits from [IDMLDeviceChild](#).

Inheritance

The **IDMLOperator** interface inherits from the [IDMLDeviceChild](#) interface.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[IDMLDeviceChild](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLOperatorInitializer interface (directml.h)

Article 02/22/2024

Represents a specialized object whose purpose is to initialize compiled operators. To create an instance of this object, call [IDMLDevice::CreateOperatorInitializer](#). The **IDMLOperatorInitializer** interface inherits from [IDMLDispatchable](#).

An operator initializer is associated with one or more compiled operators, which are the targets for initialization. You can record operator initialization onto a command list using [IDMLCommandRecorder::RecordDispatch](#). When the initialization completes execution on the GPU, all of the target operators enter the initialized state. You must initialize all operators exactly once before they can be executed.

Inheritance

The **IDMLOperatorInitializer** interface inherits from the [IDMLDispatchable](#) interface.

Methods

The **IDMLOperatorInitializer** interface has these methods.

[+] [Expand table](#)

[IDMLOperatorInitializer::Reset](#)

Resets the initializer to handle initialization of a new set of operators.

Remarks

Operator initializers are reusable: once an instance has been used to initialize a set of operators, you can reset it with a different set of compiled operators as targets.

When executing an initializer, the expected bindings are as follows:

- Inputs should be one buffer array binding for each target operator, in the order that you originally specified the operators when creating or resetting the initializer. Each buffer array binding itself should have a size equal to the inputs of its

respective operator. Alternatively, you may specify NONE for a binding to bind no inputs for initialization of that target operator.

- Outputs should be the persistent resources for each target operator, in the order that you originally specified the operators when creating or resetting the initializer.
- As with any dispatchable object (an operator initializer, or a compiled operator), the initializer may require a temporary resource. Call [IDMLDispatchable::GetBindingProperties](#) to determine the required size of the temporary resource.
- Operator initializers don't ever require persistent resources. Therefore, calling [IDMLDispatchable::GetBindingProperties](#) on an operator initializer always returns a **PersistentResourceSize** of 0.

The operator initializer itself doesn't need to be initialized—GPU initialization only applies to compiled operators.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[Binding in DirectML](#)

[IDMLDispatchable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLOperatorInitializer::Reset method (directml.h)

Article 02/22/2024

Resets the initializer to handle initialization of a new set of operators.

You may use an initializer only to initialize a fixed set of operators, which are provided either during creation ([IDMLDevice::CreateOperatorInitializer](#)), or when the initializer is reset. Resetting the initializer allows your application to reuse an existing initializer object to initialize a new set of operators.

You must not call **Reset** until all outstanding work using the initializer has completed execution on the GPU.

This method is not thread-safe.

Syntax

C++

```
HRESULT Reset(  
    UINT operatorCount,  
    [in, optional] IDMLCompiledOperator * const *operators  
>;
```

Parameters

`operatorCount`

Type: **UINT**

This parameter determines the number of elements in the array passed in the *operators* parameter.

`[in, optional] operators`

Type: **IDMLCompiledOperator***

An optional pointer to a constant array of [IDMLCompiledOperator](#) pointers containing the operators that the initializer should initialize.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLOperatorInitializer](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IDMLPageable interface (directml.h)

Article 02/22/2024

Implemented by objects that can be evicted from GPU memory, and hence that can be supplied to [IDMLDevice::Evict](#) and [IDMLDevice::MakeResident](#). The [IDMLOperator](#) interface inherits from [IDMLDeviceChild](#).

Inheritance

The [IDMLPageable](#) interface inherits from the [IDMLDeviceChild](#) interface.

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	directml.h

See also

[IDMLDeviceChild](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DirectML functions

Article • 02/10/2025

The following functions are declared in `DirectML.h`.

In this section

[] Expand table

Topic	Description
DMLCreateDevice	Creates a DirectML device for a given Direct3D 12 device.
DMLCreateDevice1	Creates a DirectML device for a given Direct3D 12 device.

Related topics

- [DirectML reference](#)
- [Windows AI](#)
- [Core reference](#)
- [Direct3D 12 Reference](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DMLCreateDevice function (directml.h)

Article12/02/2022

Creates a DirectML device for a given Direct3D 12 device.

Syntax

C++

```
HRESULT DMLCreateDevice(
    ID3D12Device             *d3d12Device,
    DML_CREATE_DEVICE_FLAGS   flags,
    REFIID                   riid,
    void                     **ppv
);
```

Parameters

d3d12Device

Type: [ID3D12Device*](#)

A pointer to an [ID3D12Device](#) representing the Direct3D 12 device to create the DirectML device over. DirectML supports any D3D feature level, and Direct3D 12 devices created on any adapter, including WARP. However, not all features in DirectML may be available depending on the capabilities of the Direct3D 12 device. See [IDMLDevice::CheckFeatureSupport](#) for more info.

If the call to **DMLCreateDevice** is successful, then the DirectML device maintains a strong reference to the supplied Direct3D 12 device.

flags

Type: [DML_CREATE_DEVICE_FLAGS](#)

A [DML_CREATE_DEVICE_FLAGS](#) value specifying additional device creation options.

riid

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in *device*. This is expected to be the GUID of [IDMLDevice](#).

ppv

Type: _COM_Outptr_opt_ **void****

A pointer to a memory block that receives a pointer to the device. This is the address of a pointer to an [IDMLDevice](#), representing the DirectML device created.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Remarks

A newer version of this function, [DMLCreateDevice1](#), was introduced in DirectML version 1.1.0. **DMLCreateDevice** is equivalent to calling [DMLCreateDevice1](#) and supplying a *minimumFeatureLevel* of [DML_FEATURE_LEVEL_1_0](#).

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

- [DMLCreateDevice1 function](#)
- [DirectML version history](#)
- [Using the DirectML debug layer](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DMLCreateDevice1 function (directml.h)

Article12/02/2022

Creates a DirectML device for a given Direct3D 12 device.

Syntax

C++

```
HRESULT DMLCreateDevice1(
    ID3D12Device             *d3d12Device,
    DML_CREATE_DEVICE_FLAGS   flags,
    DML_FEATURE_LEVEL         minimumFeatureLevel,
    REFIID                   riid,
    void                     **ppv
);
```

Parameters

d3d12Device

Type: [ID3D12Device*](#)

A pointer to an [ID3D12Device](#) representing the Direct3D 12 device to create the DirectML device over. DirectML supports any D3D feature level, and Direct3D 12 devices created on any adapter, including WARP. However, not all features in DirectML may be available depending on the capabilities of the Direct3D 12 device. See [IDMLDevice::CheckFeatureSupport](#) for more info.

If the call to **DMLCreateDevice1** is successful, then the DirectML device maintains a strong reference to the supplied Direct3D 12 device.

flags

Type: [DML_CREATE_DEVICE_FLAGS](#)

A [DML_CREATE_DEVICE_FLAGS](#) value specifying additional device creation options.

minimumFeatureLevel

Type: [DML_FEATURE_LEVEL](#)

A [DML_FEATURE_LEVEL](#) value specifying the minimum feature level support required.

This parameter can be useful for callers that require a specific version of DirectML, but who may find themselves calling into an older version of DirectML. This can happen, for example, when the user runs your application on an older version of Windows 10.

Applications that explicitly depend on functionality introduced in a particular feature level can specify it as a *minimumFeatureLevel*. This will guarantee that **DMLCreateDevice1** won't succeed unless the underlying implementation is *at least* as capable as this requested minimum feature level.

Note that as this parameter specifies a *minimum* feature level, the underlying DirectML device may in fact support a higher feature level than the requested minimum. Once device creation succeeds, you can query all of the feature levels supported by this device using [IDMLDevice::CheckFeatureSupport](#).

`riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in *device*. This is expected to be the GUID of [IDMLDevice](#).

`ppv`

Type: [_COM_Outptr_opt_ void**](#)

A pointer to a memory block that receives a pointer to the device. This is the address of a pointer to an [IDMLDevice](#), representing the DirectML device created.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

If this version of DirectML doesn't support the *minimumFeatureLevel* requested, then this function will return [DXGI_ERROR_UNSUPPORTED](#).

Remarks

A newer version of this function, [DMLCreateDevice1](#), was introduced in DirectML version 1.1.0. **DMLCreateDevice1** is equivalent to calling **DMLCreateDevice1** and supplying a *minimumFeatureLevel* of [DML_FEATURE_LEVEL_1_0](#).

Availability

This API was introduced in DirectML version 1.1.0.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

- [DML_FEATURE_LEVEL enumeration](#)
- [DirectML version history](#)
- [DirectML feature level history](#)
- [Using the DirectML debug layer](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DirectML structures

Article • 02/14/2025

The following structures are declared in `DirectML.h`.

In this section

[+] Expand table

Topic and description
DML_ACTIVATION CELU_OPERATOR_DESC . Performs the continuously differentiable exponential linear unit (CELU) activation function on every element in <i>InputTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ACTIVATION ELU_OPERATOR_DESC . Describes a DirectML operator that performs an exponential linear unit (ELU) activation function on every element in the input.
DML_ACTIVATION GELU_OPERATOR_DESC . Performs the gaussian error linear unit (GELU) activation function on every element in <i>InputTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ACTIVATION HARD_SWISH_OPERATOR_DESC . Performs a hard swish activation function on every element in <i>InputTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ACTIVATION HARDMAX_OPERATOR_DESC . Describes a DirectML activation operator that performs a hardmax function on the input.
DML_ACTIVATION HARDMAX1_OPERATOR_DESC . Performs a hardmax function on each element of <i>InputTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ACTIVATION HARD_SIGMOID_OPERATOR_DESC . Describes a DirectML activation operator that performs a hard sigmoid function on every element in the input.
DML_ACTIVATION IDENTITY_OPERATOR_DESC . Describes a DirectML activation operator that performs the identity function.
DML_ACTIVATION LEAKY RELU_OPERATOR_DESC . Describes a DirectML operator that performs a leaky rectified linear unit (ReLU) activation function on every element in the input.
DML_ACTIVATION LINEAR_OPERATOR_DESC . Describes a DirectML operator that performs a linear activation function on every element in the input.
DML_ACTIVATION LOG_SOFTMAX_OPERATOR_DESC . Describes a DirectML operator that performs a log-of-softmax activation function on the input.
DML_ACTIVATION LOG_SOFTMAX1_OPERATOR_DESC . Performs a natural log-of-softmax activation function on each element of <i>InputTensor</i> , placing the result into the corresponding

Topic and description
element of <i>OutputTensor</i> .
DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC . Describes a DirectML operator that performs a parameterized rectified linear unit (ReLU) activation function on every element in the input.
DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC . Describes a DirectML operator that performs a parametric softplus activation function on every element in the input.
DML_ACTIVATION_RELU_GRAD_OPERATOR_DESC . Computes backpropagation gradients for a rectified linear unit (ReLU).
DML_ACTIVATION_RELU_OPERATOR_DESC . Describes a DirectML operator that performs a rectified linear unit (ReLU) activation function on every element in the input.
DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC . Describes a DirectML operator that performs a scaled exponential linear unit (ELU) activation function on every element in the input.
DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC . Describes a DirectML operator that performs a scaled hyperbolic tangent activation function on every element in the input.
DML_ACTIVATION_SHRINK_OPERATOR_DESC . Describes a DirectML operator that performs an elementwise shrink activation function on the input.
DML_ACTIVATION_SIGMOID_OPERATOR_DESC . Describes a DirectML operator that performs a sigmoid activation function on every element in the input.
DML_ACTIVATION_SOFTMAX_OPERATOR_DESC . Describes a DirectML operator that performs a softmax activation function on the input.
DML_ACTIVATION_SOFTMAX1_OPERATOR_DESC . Performs a softmax activation function on <i>InputTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC . Describes a DirectML operator that performs a softplus activation function on every element in the input.
DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC . Describes a DirectML operator that performs a softsign activation function on every element in the input.
DML_ACTIVATION_SWISH_OPERATOR_DESC . Performs a swish activation function on every element in <i>InputTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ACTIVATION_TANH_OPERATOR_DESC . Describes a DirectML operator that performs a hyperbolic tangent activation function on every element in the input.
DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC . Describes a DirectML operator that performs a thresholded rectified linear unit (ReLU) activation function on every element in the input.
DML_ADAM_OPTIMIZER_OPERATOR_DESC . Computes updated weights (parameters) using the supplied gradients, based on the Adam (ADAptive Moment estimation) algorithm. This operator is

Topic and description
an optimizer, and is typically used in the weight update step of a training loop to perform gradient descent.
DML_AVERAGE_POOLING_OPERATOR_DESC . Describes a DirectML operator that performs an average pooling function on the input.
DML_AVERAGE_POOLING1_OPERATOR_DESC . Averages values across the elements within the sliding window over the input tensor.
DML_ARGMAX_OPERATOR_DESC . Outputs the indices of the maximum-valued elements within one or more dimensions of the input tensor.
DML_ARGMIN_OPERATOR_DESC . Outputs the indices of the minimum-valued elements within one or more dimensions of the input tensor.
DML_AVERAGE_POOLING_GRAD_OPERATOR_DESC . Computes backpropagation gradients for average pooling (see DML_AVERAGE_POOLING_OPERATOR_DESC).
DML_BATCH_NORMALIZATION_GRAD_OPERATOR_DESC . Computes backpropagation gradients for batch normalization.
DML_BATCH_NORMALIZATION_OPERATOR_DESC . Describes a DirectML operator that performs a batch normalization function on the input.
DML_BATCH_NORMALIZATION_TRAINING_GRAD_OPERATOR_DESC . Computes backpropagation gradients for batch normalization training.
DML_BATCH_NORMALIZATION_TRAINING_OPERATOR_DESC . Performs a batch normalization on the input.
DML_BINDING_DESC . Contains the description of a binding so that you can add it to the binding table via a call to one of the IDMLBindingTable methods.
DML_BINDING_PROPERTIES . Contains information about the binding requirements of a particular compiled operator, or operator initializer.
DML_BINDING_TABLE_DESC . Specifies parameters to IDMLDevice::CreateBindingTable and IDMLBindingTable::Reset .
DML_BUFFER_ARRAY_BINDING . Specifies a resource binding that is an array of individual buffer bindings.
DML_BUFFER_BINDING . Specifies a resource binding described by a range of bytes in a Direct3D 12 buffer, represented by an offset and size into an ID3D12Resource .
DML_BUFFER_TENSOR_DESC . Describes a tensor that will be stored in a Direct3D 12 buffer resource.
DML_CAST_OPERATOR_DESC . Describes a DirectML data reorganization operator that performs the cast function $f(x) = \text{cast}(x)$, casting each element in the input to the data type of the output

Topic and description
tensor, and storing the result in the corresponding element in the output.
DML_CONVOLUTION_INTEGER_OPERATOR_DESC . Performs a convolution of the <i>FilterTensor</i> with the <i>InputTensor</i> . This operator performs forward convolution on integer data.
DML_CONVOLUTION_OPERATOR_DESC . Describes a DirectML matrix multiplication operator that performs a convolution function on the input.
DML_CUMULATIVE_PRODUCT_OPERATOR_DESC . Multiplies the elements of a tensor along an axis, writing the running tally of the product into the output tensor.
DML_CUMULATIVE_SUMMATION_OPERATOR_DESC . Sums the elements of a tensor along an axis, writing the running tally of the summation into the output tensor.
DML_DEPTH_TO_SPACE_OPERATOR_DESC . Describes a DirectML data reorganization operator that rearranges (permutes) data from depth into blocks of spatial data.
DML_DEPTH_TO_SPACE1_OPERATOR_DESC . Rearranges (permutes) data from depth into blocks of spatial data. The operator outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the height and width dimensions.
DML_DEQUANTIZE_OPERATOR_DESC . TBD.
DML_DIAGONAL_MATRIX_OPERATOR_DESC . Describes a DirectML math operator that generates an identity-like matrix with ones on the major diagonal and zeros everywhere else.
DML_DIAGONAL_MATRIX1_OPERATOR_DESC . Generates an identity-like matrix with ones (or other explicit value) along the given diagonal span, with other elements being filled with either the input values or zeros (if no <i>InputTensor</i> is passed).
DML_DYNAMIC_QUANTIZE_LINEAR_OPERATOR_DESC . Calculates the quantization scale and zero point values necessary to quantize the <i>InputTensor</i> , then applies that quantization, writing the result to <i>OutputTensor</i> .
DML_ELEMENT_WISE_ABS_OPERATOR_DESC . Describes a DirectML math operator that performs the element-wise absolute value function $f(x) = \text{abs}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ACOS_OPERATOR_DESC . Describes a DirectML trigonometric operator that performs the element-wise arccosine function $f(x) = \text{acos}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC . Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic cosine function $f(x) = \log(x + \sqrt{x * x - 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ADD_OPERATOR_DESC . Describes a DirectML math operator that performs the function of adding every element in <i>ATensor</i> to its corresponding element in <i>BTensor</i> .
DML_ELEMENT_WISE_ADD1_OPERATOR_DESC . Describes a DirectML math operator that performs the function of adding every element in <i>ATensor</i> to its corresponding element in <i>BTensor</i> ,

Topic and description
f(a, b) = a + b, with the option for fused activation.
DML_ELEMENT_WISE_ASIN_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise arcsine function $f(x) = \text{asin}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ASINH_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic sine function $f(x) = \log(x + \sqrt{x * x + 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ATAN_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise arctangent function $f(x) = \text{atan}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ATANH_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = (\log((1 + x) / (1 - x)) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ATAN_YX_OPERATOR_DESC. Computes the 2-argument arctangent for each element of <i>ATensor</i> and <i>BTensor</i> , where <i>ATensor</i> is the <i>Y-axis</i> and <i>BTensor</i> is the <i>X-axis</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_BIT_AND_OPERATOR_DESC. Computes the bitwise AND between each corresponding element of the input tensors, and writes the result into the output tensor.
DML_ELEMENT_WISE_BIT_COUNT_OPERATOR_DESC. Computes the bitwise NOT for each element of the input tensor, and writes the result into the output tensor.
DML_ELEMENT_WISE_BIT_NOT_OPERATOR_DESC. Computes the bitwise population count (the number of bits set to 1) for each element of the input tensor, and writes the result into the output tensor.
DML_ELEMENT_WISE_BIT_OR_OPERATOR_DESC. Computes the bitwise OR between each corresponding element of the input tensors, and writes the result into the output tensor.
DML_ELEMENT_WISE_BIT_SHIFT_LEFT_OPERATOR_DESC. Performs a logical left shift of each element of <i>ATensor</i> by a number of bits given by the corresponding element of <i>BTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_BIT_SHIFT_RIGHT_OPERATOR_DESC. Performs a logical right shift of each element of <i>ATensor</i> by a number of bits given by the corresponding element of <i>BTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_BIT_XOR_OPERATOR_DESC. Computes the bitwise XOR (eXclusive OR) between each corresponding element of the input tensors, and writes the result into the output tensor.
DML_ELEMENT_WISE_CEIL_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise ceiling function $f(x) = \text{ceil}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

Topic and description
DML_ELEMENT_WISE_CLIP_GRAD_OPERATOR_DESC. Computes backpropagation gradients for element-wise clip.
DML_ELEMENT_WISE_CLIP_GRAD1_OPERATOR_DESC. Computes backpropagation gradients for DML_ELEMENT_WISE_CLIP1_OPERATOR_DESC.
DML_ELEMENT_WISE_CLIP_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise clip function $f(x) = \text{clamp}(x * \text{scale} + \text{bias}, \text{minValue}, \text{maxValue})$, where the scale and bias terms are optional, and where $\text{clamp}(x) = \min(\text{maxValue}, \max(\text{minValue}, x))$.
DML_ELEMENT_WISE_CLIP1_OPERATOR_DESC. Performs a clamping (or limiting) operation for each element of <i>InputTensor</i> , placing the result into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC. Describes a DirectML operator that performs the element-wise constant power function $f(x) = \text{pow}(x * \text{scale} + \text{bias}, \text{exponent})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_COS_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise cosine function $f(x) = \cos(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_COSH_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise hyperbolic cosine function $f(x) = ((e^x + e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC. Describes a DirectML operator that performs the linear dequantize function on every element in <i>InputTensor</i> with respect to its corresponding element in <i>ScaleTensor</i> and <i>ZeroPointTensor</i> .
DML_ELEMENT_WISE_DIFFERENCE_SQUARE_OPERATOR_DESC. Subtracts each element of <i>BTensor</i> from the corresponding element of <i>ATensor</i> , multiplies the result by itself, and places the result into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC. Describes a DirectML math operator that performs the function of dividing every element in <i>ATensor</i> by its corresponding element in <i>BTensor</i> .
DML_ELEMENT_WISE_ERF_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise natural exponential function $f(x) = \exp(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_EXP_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise natural exponential function $f(x) = \exp(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise floor function $f(x) = \text{floor}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

Topic and description
DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC. Describes a DirectML generic operator that performs the element-wise identity function $f(x) = x * \text{scale} + \text{bias}$.
DML_ELEMENT_WISE_IF_OPERATOR_DESC. Describes a DirectML math operator that essentially performs a ternary <code>if</code> statement.
DML_ELEMENT_WISE_IS_INFINITY_OPERATOR_DESC. Checks each element of <i>InputTensor</i> for IEEE-754 -inf, inf, or both, depending on the given <i>InfinityMode</i> , and places the result (1 for true, 0 for false) into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC. Describes a DirectML math operator that determines, elementwise, whether the input is NaN.
DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC. Describes a DirectML math operator that performs a logical AND function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC. Describes a DirectML math operator that performs a logical equality function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC. Describes a DirectML math operator that performs a logical greater-than function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL_OPERATOR_DESC. Performs a logical <i>greater than or equal to</i> on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC. Describes a DirectML math operator that performs a logical less-than function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL_OPERATOR_DESC. Performs a logical <i>less than or equal to</i> on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of <i>OutputTensor</i> .
DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC. Describes a DirectML math operator that performs a logical NOT function on every element in the input.
DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC. Describes a DirectML math operator that performs a logical OR function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC. Describes a DirectML math operator that performs a logical exclusive OR (XOR) function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .

Topic and description
DML_ELEMENT_WISE_LOG_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise natural logarithm function $f(x) = \log(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_MAX_OPERATOR_DESC. Describes a DirectML math reduction operator that performs a maximum function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_MEAN_OPERATOR_DESC. Describes a DirectML math reduction operator that performs an arithmetic mean function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_MIN_OPERATOR_DESC. Describes a DirectML math reduction operator that performs a minimum function between every element in <code>ATensor</code> and its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_MODULUS_FLOOR_OPERATOR_DESC. Computes the modulus, with the same results as the Python modulus, for each pair of corresponding elements from the input tensors, placing the result into the corresponding element of <code>OutputTensor</code> .
DML_ELEMENT_WISE_MODULUS_TRUNCATE_OPERATOR_DESC. Computes the C modulus operator for each pair of corresponding elements of the input tensors, placing the result into the corresponding element of <code>OutputTensor</code> .
DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC. Describes a DirectML math operator that performs the function of multiplying every element in <code>ATensor</code> by its corresponding element in <code>BTensor</code> .
DML_ELEMENT_WISE_NEGATE_OPERATOR_DESC. Negates each element of <code>InputTensor</code> , storing the result into the corresponding element of <code>OutputTensor</code> .
DML_ELEMENT_WISE_POW_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise power function $f(x, \text{exponent}) = \text{pow}(x * \text{scale} + \text{bias}, \text{exponent})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC. Describes a DirectML operator that performs the linear quantize function on every element in <code>InputTensor</code> with respect to its corresponding element in <code>ScaleTensor</code> and <code>ZeroPointTensor</code> .
DML_ELEMENT_WISE_QUANTIZED_LINEAR_ADD_OPERATOR_DESC. Adds every element in <code>ATensor</code> to its corresponding element in <code>BTensor</code> , placing the result into the corresponding element of <code>OutputTensor</code> .
DML_ELEMENT_WISE_RECIP_OPERATOR_DESC. Describes a DirectML math operator that performs a reciprocal function on every element in the input.
DML_ELEMENT_WISE_ROUND_OPERATOR_DESC. Rounds each element of <code>InputTensor</code> to an integer value, placing the result into the corresponding element of <code>OutputTensor</code> .

Topic and description
DML_ELEMENT_WISE_SIGN_OPERATOR_DESC. Describes a DirectML operator that performs an elementwise shrink activation function on the input.
DML_ELEMENT_WISE_SIN_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise sine function $f(x) = \sin(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SINH_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise hyperbolic sine function $f(x) = ((e^x - e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SQRT_OPERATOR_DESC. Describes a DirectML math operator that performs a square root function on every element in the input.
DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC. Describes a DirectML math operator that performs the function of subtracting every element in <code>BTensor</code> from its corresponding element in <code>ATensor</code> .
DML_ELEMENT_WISE_TAN_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise tangent function $f(x) = \tan(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_TANH_OPERATOR_DESC. Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = \tanh(x) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC. Describes a DirectML math operator that performs the element-wise threshold function $f(x) = \max(x * \text{scale} + \text{bias}, \text{min})$, where the scale and bias terms are optional.
DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT. Provides detail about whether a DirectML device supports a particular data type within tensors.
DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT. Used to query a DirectML device for its support for a particular data type within tensors.
DML_FILL_VALUE_CONSTANT_OPERATOR_DESC. Fills a tensor with the given constant <i>Value</i> .
DML_FILL_VALUE_SEQUENCE_OPERATOR_DESC. Fills a tensor with a sequence.
DML_FOLD_OPERATOR_DESC structure. Combines an array of patches formed from a sliding window into a large containing tensor.
DML_GATHER_ELEMENTS_OPERATOR_DESC. Gathers elements from the input tensor along the given axis using the indices tensor to remap into the input.
DML_GATHER_ND_OPERATOR_DESC. Gathers elements from the input tensor, using the indices tensor to remap indices to entire subblocks of the input.

Topic and description
DML_GATHER_ND1_OPERATOR_DESC. Gathers elements from the input tensor, using the indices tensor to remap indices to entire subblocks of the input.
DML_GATHER_OPERATOR_DESC. Describes a DirectML data reorganization operator which, when given a data tensor of rank $r \geq 1$, and an indices tensor of rank q , gathers the entries in the axis dimension of the data (by default, the outermost one is $\text{axis} == 0$) indexed by indices, and concatenates them in an output tensor of rank $q + (r - 1)$.
DML_GEMM_OPERATOR_DESC. Describes a DirectML operator that performs a general matrix multiplication function on the input, $y = \alpha * \text{transposeA}(A) * \text{transposeB}(B) + \beta * C$.
DML_GRAPH_DESC. Describes a graph of DirectML operators used to compile a combined, optimized operator.
DML_GRAPH_EDGE_DESC. A generic container for a connection within a graph of DirectML operators defined by DML_GRAPH_DESC and passed to IDMLDevice1::CompileGraph .
DML_GRAPH_NODE_DESC. A generic container for a node within a graph of DirectML operators defined by DML_GRAPH_DESC and passed to IDMLDevice1::CompileGraph .
DML_GRU_OPERATOR_DESC. Describes a DirectML deep learning operator that performs a (standard layers) one-layer gated recurrent unit (GRU) function on the input.
DML_INPUT_GRAPH_EDGE_DESC. Describes a connection within a graph of DirectML operators defined by DML_GRAPH_DESC and passed to IDMLDevice1::CompileGraph . This structure is used to define a connection from a graph input to an input of an internal node.
DML_INTERMEDIATE_GRAPH_EDGE_DESC. Describes a connection within a graph of DirectML operators defined by DML_GRAPH_DESC and passed to IDMLDevice1::CompileGraph . This structure is used to define a connection between internal nodes.
DML_JOIN_OPERATOR_DESC. Describes a DirectML operator that performs a join function on an array of input tensors.
DML_LOCAL_RESPONSE_NORMALIZATION_GRAD_OPERATOR_DESC. Computes backpropagation gradients for local response normalization.
DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC. Describes a DirectML operator that performs a local response normalization (LRN) function on the input.
DML_LP_NORMALIZATION_OPERATOR_DESC. Describes a DirectML operator that performs an Lp-normalization function along the specified axis of the input tensor.
DML_LP_POOLING_OPERATOR_DESC. Describes a DirectML operator that performs an Lp pooling function across the input tensor.
DML_LP_POOLING1_OPERATOR_DESC. Computes the LP normalized value across the elements within the sliding window over the input tensor.

Topic and description
DML_LSTM_OPERATOR_DESC . Describes a DirectML deep learning operator that performs a one-layer long short term memory (LSTM) function on the input.
DML_MATRIX_MULTIPLY_INTEGER_OPERATOR_DESC . Performs a matrix multiplication function on integer data.
DML_MATRIX_MULTIPLY_INTEGER_TO_FLOAT_OPERATOR_DESC . Performs a matrix multiplication function on integer input tensor data, and produces floating point output.
DML_MAX_POOLING_GRAD_OPERATOR_DESC . Computes backpropagation gradients for max pooling (see DML_MAX_POOLING2_OPERATOR_DESC).
DML_MAX_POOLING_OPERATOR_DESC . Describes a DirectML operator that performs a max pooling function across the input tensor.
DML_MAX_POOLING1_OPERATOR_DESC . Describes a DirectML operator that performs a max pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = \max(x_1 + x_2 + \dots + x_{\text{pool_size}})$.
DML_MAX_POOLING2_OPERATOR_DESC . Computes the maximum value across the elements within the sliding window over the input tensor, and optionally returns the indices of the maximum values selected.
DML_MAX_UNPOOLING_OPERATOR_DESC . Describes a DirectML operator that fills the output tensor of the given shape (either explicit, or the input shape plus padding) with zeros, then writes each value from the input tensor into the output tensor at the element offset from the corresponding indices array.
DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC . Describes a DirectML operator that performs a mean variance normalization function on the input tensor.
DML_MEAN_VARIANCE_NORMALIZATION1_OPERATOR_DESC . Performs a mean variance normalization function on the input tensor. This operator will calculate the mean and variance of the input tensor to perform normalization.
DML_MEAN_VARIANCE_NORMALIZATION2_OPERATOR_DESC . TBD.
DML_MULTIHEAD_ATTENTION_OPERATOR_DESC . Performs a multi-head attention operation.
DML_MULTIHEAD_ATTENTION1_OPERATOR_DESC . TBD.
DML_NONZERO_COORDINATES_OPERATOR_DESC . Computes the N-dimensional coordinates of all non-zero elements of the input tensor.
DML_ONE_HOT_OPERATOR_DESC . Describes a DirectML operator that generates a tensor with each element filled with two values—either an 'on' or an 'off' value.
DML_OPERATOR_DESC . A generic container for an operator description. You construct DirectML operators using the parameters specified in this struct.

Topic and description
DML_OPERATOR_GRAPH_NODE_DESC. Describes a node within a graph of DirectML operators defined by DML_GRAPH_DESC and passed to IDMLDevice1::CompileGraph .
DML_OUTPUT_GRAPH_EDGE_DESC. Describes a connection within a graph of DirectML operators defined by DML_GRAPH_DESC and passed to IDMLDevice1::CompileGraph . This structure is used to define a connection from an output of an internal node to a graph output.
DML_PADDING_OPERATOR_DESC. Describes a DirectML data reorganization operator that inflates the input tensor with zeroes (or some other value) on the edges.
DML_PADDING1_OPERATOR_DESC. Inflates the input tensor with constant or mirrored values on the edges, and writes the result to the output.
DML_QUANTIZE_OPERATOR_DESC. TBD.
DML_QUANTIZED_LINEAR_AVERAGE_POOLING_OPERATOR_DESC. Averages quantized values across the elements within the sliding window over the input tensor. This operator is mathematically equivalent to dequantizing the inputs, then performing average pooling, and then quantizing the output.
DML_QUANTIZED_LINEAR_CONVOLUTION_OPERATOR_DESC. Performs a convolution of the <i>FilterTensor</i> with the <i>InputTensor</i> . This operator performs forward convolution on quantized data. This operator is mathematically equivalent to dequantizing the inputs, convolving, and then quantizing the output.
DML_QUANTIZED_LINEAR_MATRIX_MULTIPLY_OPERATOR_DESC. Performs a matrix multiplication function on quantized data. This operator is mathematically equivalent to dequantizing the inputs, then performing matrix multiply, and then quantizing the output.
DML_RANDOM_GENERATOR_OPERATOR_DESC. Fills an output tensor with deterministically-generated, pseudo-random, uniformly-distributed bits. This operator optionally may also output an updated internal generator state, which can be used during subsequent executions of the operator.
DML_REDUCE_OPERATOR_DESC. Describes a DirectML operator that performs the specified reduction function on the input.
DML_RESAMPLE_GRAD_OPERATOR_DESC. Computes backpropagation gradients for Resample (see DML_RESAMPLE1_OPERATOR_DESC).
DML_RESAMPLE_GRAD1_OPERATOR_DESC. Computes backpropagation gradients for DML_RESAMPLE2_OPERATOR_DESC .
DML_RESAMPLE_OPERATOR_DESC. Describes a DirectML operator that resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size.
DML_RESAMPLE1_OPERATOR_DESC. Resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size. You can use a linear or

Topic and description
nearest-neighbor interpolation mode.
DML_RESAMPLE2_OPERATOR_DESC . Resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size.
DML_RESAMPLE3_OPERATOR_DESC structure . Resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size.
DML_REVERSE_SUBSEQUENCES_OPERATOR_DESC . Reverses the elements of one or more <i>subsequences</i> of a tensor. The set of subsequences to be reversed is chosen based on the provided axis and sequence lengths.
DML_RNN_OPERATOR_DESC . Describes a DirectML deep learning operator that performs a one-layer simple recurrent neural network (RNN) function on the input.
DML_ROI_ALIGN_GRAD_OPERATOR_DESC . Computes backpropagation gradients for ROI_ALIGN and ROI_ALIGN1 .
DML_ROI_ALIGN_OPERATOR_DESC . Performs an ROI align operation, as described in the Mask R-CNN paper. In summary, the operation extracts crops from the input image tensor and resizes them to a common output size specified by the last 2 dimensions of <i>OutputTensor</i> using the specified <i>InterpolationMode</i> .
DML_ROI_ALIGN1_OPERATOR_DESC . Performs an ROI align operation, as described in the Mask R-CNN paper. In summary, the operation extracts cropped windows from the input image tensor, and resizes them to a common output size specified by the last 2 dimensions of <i>OutputTensor</i> using the specified <i>InterpolationMode</i> .
DML_ROI_POOLING_OPERATOR_DESC . Describes a DirectML operator that performs a pooling function across the input tensor (according to regions of interest, or ROIs).
DML_SCALAR_UNION . A union of scalar types.
DML_SCALE_BIAS . Contains the values of scale and bias terms supplied to a DirectML operator.
DML_SCATTER_ND_OPERATOR_DESC . Copies the whole input tensor to the output, then overwrites selected indices with corresponding values from the updates tensor.
DML_SCATTER_OPERATOR_DESC . Describes a DirectML operator that copies the whole input tensor to the output, then overwrites selected indices with corresponding values from the updates tensor.
DML_SIZE_2D . Contains values that can represent the size (as supplied to a DirectML operator) of a 2-D plane of elements within a tensor, or a 2-D scale, or any 2-D width/height value.
DML_SLICE_GRAD_OPERATOR_DESC . Computes backpropagation gradients for Slice (see DML_SLICE1_OPERATOR_DESC).
DML_SLICE_OPERATOR_DESC . Describes a DirectML data reorganization operator that produces a slice of the input tensor along multiple axes.

Topic and description
DML_SLICE1_OPERATOR_DESC . Extracts a single subregion (a "slice") of an input tensor.
DML_SPACE_TO_DEPTH_OPERATOR_DESC . Describes a DirectML data reorganization operator that rearranges blocks of spatial data into depth.
DML_SPACE_TO_DEPTH1_OPERATOR_DESC . Rearranges blocks of spatial data into depth. The operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension.
DML_SPLIT_OPERATOR_DESC . Describes a DirectML data reorganization operator that splits the input tensor into multiple output tensors, along the specified axis.
DML_TENSOR_DESC . A generic container for a DirectML tensor description.
DML_TILE_OPERATOR_DESC . Describes a DirectML data reorganization operator that constructs an output tensor by tiling the input tensor.
DML_TOP_K_OPERATOR_DESC . Describes a DirectML reduction operator that retrieves the top K elements along a specified axis.
DML_TOP_K1_OPERATOR_DESC . Selects the largest or smallest K elements from each sequence along an axis of the <i>InputTensor</i> , and returns the values and indices of those elements in the <i>OutputValueTensor</i> and <i>OutputIndexTensor</i> , respectively.
DML_UNFOLD_OPERATOR_DESC structure . Extracts sliding local blocks from a batched input tensor.
DML_UPSAMPLE_2D_OPERATOR_DESC . Describes a DirectML imaging operator that upsamples the image contained in the input tensor.
DML_VALUE_SCALE_2D_OPERATOR_DESC . Describes a DirectML operator that performs an element-wise scale-and-bias function on the values in the input tensor.

Related topics

- [DirectML reference](#)
- [Windows AI](#)
- [Core reference](#)
- [Direct3D 12 Reference](#)

Feedback

Was this page helpful?

 Yes

 No

DML_ACTIVATION_CELU_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Performs the continuously differentiable exponential linear unit (CELU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = max(0, x) + min(0, Alpha * (exp(x / Alpha) - 1));
```

Where:

- $\exp(x)$ is the natural exponentiation function
- $\max(a,b)$ returns the larger of the two values a,b
- $\min(a,b)$ returns the smaller of the two values a,b

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

```
C++  
  
struct DML_ACTIVATION_CELU_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Alpha;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The alpha coefficient. A typical default for this value is 1.0.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[+] Expand table

Tensor	Kind	Supported Dimension Counts	Supported Data Types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_ELU_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs an exponential linear unit (ELU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = x,           if x >= 0  
        Alpha * (exp(x) - 1), otherwise
```

Where $\exp(x)$ is the natural exponentiation function.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

```
C++  
  
struct DML_ACTIVATION_ELU_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Alpha;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The alpha coefficient. A typical default for this value is 1.0.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML FEATURE LEVEL 3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML FEATURE LEVEL 2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML FEATURE LEVEL 1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_GELU_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Performs the gaussian error linear unit (GELU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = 0.5 * x * (1.0 + erf(x / sqrt(2)))
```

Where $\text{erf}(x)$ is [DML_ELEMENT_WISE_ERF_OPERATOR_DESC](#).

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) ↗ version 1.9 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_ACTIVATION_GELU_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* OutputTensor;
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

The input tensor to read from.

`OutputTensor`

Type: [const DML_TENSOR_DESC*](#)

The output tensor to write the results to.

Availability

This operator was introduced in **DML_FEATURE_LEVEL_5_1**.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

Requirements

[] Expand table

Header	directml.h
--------	------------

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_HARDMAX_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs a hardmax function on each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

The operator computes the hardmax (1 for the first occurrence of the largest value in the layer, and 0 for all other values) of each row in the given input.

Syntax

C++

```
struct DML_ACTIVATION_HARDMAX_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: **const DML_TENSOR_DESC***

The tensor to read from for the input. This tensor must have an *effective rank* no greater than 2. The effective rank of a tensor is the *DimensionCount* of the tensor, excluding leftmost dimensions of size 1. For example a tensor size of `{ 1, 1, BatchCount, Width }` is valid, and is equivalent to a tensor of sizes `{ BatchCount, Width }`.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Remarks

The operator computes the hardmax (1 for the first maximum value, and 0 for all others) values for each layer in the batch of the given input. The input is a 2-D tensor (Tensor) of

size (batch_size x input_feature_dimensions). The output tensor has the same shape and contains the hardmax values of the corresponding input.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16

[Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_ARGMAX_OPERATOR_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_HARDMAX1_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Performs a hardmax function on each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

The operator computes the hardmax (1 for the first maximum value along the specified axes, and 0 for all other values) of each element in the given input.

```
reducedTensor = ReduceArgMax(InputTensor, axes = Axes, axisDirection =
DML_AXIS_DIRECTION_INCREASING)
broadcastedTensor = Broadcast the `reducedTensor` to `InputTensor`
for each coordinate in OutputTensor
    if broadcastedTensor[coordinate] == reducedIndex0f(coordinate)    //
reducedIndex0f(coordinate) is the index of the coordinate within reduced
axes `axes`.
        OutputTensor[coordinate] = 1
    else
        OutputTensor[coordinate] = 0
endfor
```

Where `ReduceArgMax(input = InputTensor, axis = Axes)` is [DML_REDUCE_OPERATOR](#) with [DML_REDUCE_FUNCTION_ARGMAX](#) as a reduce function.

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) ↗ version 1.9 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_ACTIVATION_HARDMAX1_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* OutputTensor;
    UINT AxisCount;
```

```
    _Field_size_(AxisCount) const UINT* Axes;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

AxisCount

Type: **UINT**

The number of axes to calculate reduce hardmax. This field determines the size of the *Axes* array.

Axes

Type: *_Field_size_(AxisCount)* **const UINT***

The axes along which to reduce hardmax. Values must be in the range [0, *InputTensor.DimensionCount - 1*].

Examples

The following examples all use this same three-dimensional input tensor:

```
InputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)  
[  
 [ [ 12, 0],  
   [-101, 11],  
 ],  
 [ [ 3, 234],  
   [ 0, -101],
```

```
    ]  
]
```

Example 1

```
AxisCount: 1  
Axes: {1}  
OutputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)  
[  
  [           // max element in {12, -101} is 12 and in {0, 11} is 11  
    [1, 0],  
    [0, 1],  
  ],  
  [           // max element in {3, 0} is 3 and in {234, -101} is 234  
    [1, 1],  
    [0, 0],  
  ]  
]
```

Example 2

```
AxisCount: 1  
Axes: {0}  
OutputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)  
[  
  [           // max element in {12, 3} is 12, in {0, 234} is 234, in  
   {-101, 0} is 0 and in {11, -101} is 11  
    [1, 0],  
    [0, 1],  
  ],  
  [  
    [0, 1],  
    [1, 0],  
  ]  
]
```

Example 3

```
AxisCount: 2  
Axes: {0, 2}  
OutputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)
```

```

[
    [
        // max element in {12, 0, 3, 234} is 234 and in {-101,
        11, 0, -101} is 11
        [0, 0],
        [0, 1],
    ],
    [
        [0, 1],
        [0, 0],
    ]
]

```

Remarks

This operator is equivalent to [DML_ACTIVATION_HARDMAX_OPERATOR_DESC](#) when *AxisCount* == 1, and *Axes* == `{DimensionCount - 1}`.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_5_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

Requirements

[] [Expand table](#)

Header	directml.h
--------	------------

See also

- [DML_ARGMAX_OPERATOR_DESC](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a hard sigmoid function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \max(0, \min(\text{Alpha} * x + \text{Beta}, 1))$$

Where `max(a,b)` returns the larger of the two values, and `min(a,b)` returns the smaller of the two values `a,b`.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT                 Alpha;
    FLOAT                 Beta;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The alpha coefficient. A typical default for this value is 0.2.

Beta

Type: **FLOAT**

The beta coefficient. A typical default for this value is 0.5.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_SIGMOID_OPERATOR_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_IDENTITY_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs the identity activation, effectively copying every element of *InputTensor* to the corresponding element of *OutputTensor*.

```
f(x) = x
```

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_IDENTITY_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a leaky rectified linear unit (ReLU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = x,           if x >= 0  
        Alpha * x, otherwise
```

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT                 Alpha;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The alpha coefficient. A typical default for this value is 0.01.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_RELU_OPERATOR_DESC](#)

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_LINEAR_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs the linear activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \text{Alpha} * x + \text{Beta}.$$

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_LINEAR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Alpha;
    FLOAT Beta;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The alpha coefficient.

Beta

Type: **FLOAT**

The beta coefficient.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML FEATURE LEVEL 3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML FEATURE LEVEL 2_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML FEATURE LEVEL 1_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a natural log-of-softmax activation function on each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
// Let x_i be the current value in the axis, and j be the total number of
elements along that axis.
f(x_i) = ln(exp(x_i) / sum(exp(x_0), ..., exp(x_j)))
```

Where $\exp(x)$ is the natural exponentiation function, and $\ln(x)$ is the natural logarithm.

Syntax

C++

```
struct DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from. This tensor must have an *effective rank* no greater than 2. The effective rank of a tensor is the *DimensionCount* of the tensor, excluding leftmost dimensions of size 1. For example a tensor size of `{ 1, 1, BatchCount, Width }` is valid, and is equivalent to a tensor of sizes `{ BatchCount, Width }`.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_SOFTMAX_OPERATOR_DESC structure](#)

[DML_ARGMAX_OPERATOR_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_LOG_SOFTMAX1_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Performs a natural log-of-softmax activation function on each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
For 1-D InputTensor:  
// Let x[i] to be the current element in the InputTensor, and j be the total  
// number of elements in the InputTensor  
f(x[i]) = ln(exp(x[i]) / sum(exp(x[0]), ..., exp(x[j-1])))
```

Where $\exp(x)$ is the natural exponentiation function, and $\ln(x)$ is the natural logarithm.

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) ↗ version 1.9 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_ACTIVATION_LOG_SOFTMAX1_OPERATOR_DESC  
{  
    const DML_TENSOR_DESC* InputTensor;  
    const DML_TENSOR_DESC* OutputTensor;  
    UINT AxisCount;  
    _Field_size_(AxisCount) const UINT* Axes;  
};
```

Members

InputTensor

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

`AxisCount`

Type: **UINT**

The number of axes to calculate reduce sum. This field determines the size of the `Axes` array.

`Axes`

Type: `_Field_size_(AxisCount) const UINT*`

The axes along which to reduce the sum. Values must be in the range `[0, InputTensor.DimensionCount - 1]`.

Examples

The following examples all use this same three-dimensional input tensor:

```
InputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)
[
    [
        [
            [ 12, 0],
            [-101, 11],
        ],
        [
            [ 3, 234],
            [ 0, -101],
        ]
]
```

Example 1

```
AxisCount: 1
Axes: {1}
OutputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)
[
    [ // max element in {12, -101} is 12 and in {0, 11} is 11
```

```
[1, 0],  
[0, 1],  
,  
[ // max element in {3, 0} is 3 and in {234, -101} is 234  
[1, 1],  
[0, 0],  
,  
]  
]
```

Example 2

```
AxisCount: 1  
Axes: {0}  
OutputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)  
[  
[ // max element in {12, 3} is 12, in {0, 234} is 234, in  
{-101, 0} is 0 and in {11, -101} is 11  
[1, 0],  
[0, 1],  
,  
[  
[0, 1],  
[1, 0],  
,  
]  
]
```

Example 3

```
AxisCount: 2  
Axes: {0, 2}  
OutputTensor: (Sizes:{2, 2, 2}, DataType:FLOAT32)  
[  
[ // max element in {12, 0, 3, 234} is 234 and in {-101,  
11, 0, -101} is 11  
[0, 0],  
[0, 1],  
,  
[  
[0, 1],  
[0, 0],  
,  
]  
]
```

Remarks

This operator is equivalent to [DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC](#) when *AxisCount* == 1, and *Axes* == {*DimensionCount* - 1}.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_5_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

Requirements

[+] Expand table

Header	directml.h
--------	------------

See also

- [DML_ACTIVATION_SOFTMAX1_OPERATOR_DESC](#)
- [DML_ARGMAX_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC structure (directml.h)

Article 07/20/2022

Performs a parameterized rectified linear unit (ReLU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x, slope) = x,           if x >= 0
                slope * x, otherwise
```

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *SlopeTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

SlopeTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the slope for each corresponding value of the input.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Requirements

[] Expand table

Requirement	Value
Header	<code>directml.h</code>

See also

[DML_ACTIVATION_RELU_OPERATOR_DESC](#)

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor, *OutputTensor*, and *SlopeTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

`DML_FEATURE_LEVEL_5_1` and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
<code>InputTensor</code>	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8
<code>SlopeTensor</code>	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8
<code>OutputTensor</code>	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
SlopeTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
SlopeTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
SlopeTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a parametric softplus activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \text{Alpha} * \ln(1 + \exp(\text{Beta} * x))$$

Where $\exp(x)$ is the natural exponentiation function, and $\ln(x)$ is the natural logarithm.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT                 Alpha;
    FLOAT                 Beta;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The alpha coefficient.

Beta

Type: **FLOAT**

The beta coefficient.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_RELU_GRAD_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes backpropagation gradients for a rectified linear unit (ReLU). This operator performs the following element-wise computation.

```
X = InputTensor  
dY = InputGradientTensor  
  
OutputGradientTensor = (X > 0 ? dY : 0)
```

The corresponding forward-pass operator is [DML_ACTIVATION_RELU_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_ACTIVATION_RELU_GRAD_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *InputGradientTensor;  
    const DML_TENSOR_DESC *OutputGradientTensor;  
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

The input (feature) tensor. This is typically the same input as was provided during the forward pass (see [DML_ACTIVATION_RELU_OPERATOR_DESC](#)).

`InputGradientTensor`

Type: [const DML_TENSOR_DESC*](#)

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. The *Sizes* and *DataType* of this tensor must exactly match those of the *InputTensor*.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

[DML_ACTIVATION_RELU_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_RELU_OPERATOR_DESC structure (directml.h)

Article 07/20/2022

Performs a rectified linear unit (ReLU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = max(0, x)
```

Where $\max(a,b)$ returns the larger of the two values a,b .

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_RELU_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 [Yes](#) [No](#)[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a scaled exponential linear unit (ELU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = Gamma * x,           if x > 0  
      Gamma * (Alpha * exp(x) - Alpha), otherwise
```

Where $\exp(x)$ is the natural exponentiation function.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Alpha;  
    FLOAT Gamma;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The value of alpha. A typical default for this value is 1.6732.

Gamma

Type: **FLOAT**

The value of gamma. A typical default for this value is 1.0507.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_ELU_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs a scaled hyperbolic tangent activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \text{Alpha} * \tanh(\text{Beta} * x)$$

Where $\tanh(x)$ is the hyperbolic tangent function.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Alpha;
    FLOAT Beta;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`Alpha`

Type: **FLOAT**

The value of alpha. A typical default for this value is 1.0.

Beta

Type: **FLOAT**

The value of beta. A typical default for this value is 0.5.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[] Expand table

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_TANH_OPERATOR_DESC](#)

Feedback

Was this page helpful?

👍 Yes

👎 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SHRINK_OPERATOR_DESC structure (directml.h)

Article 12/02/2022

Performs the shrink activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = x - Bias, if x > Threshold  
      x + Bias, if x < -Threshold  
      0,          otherwise
```

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_SHRINK_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Bias;  
    FLOAT Threshold;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`Bias`

Type: **FLOAT**

The value of the bias. A typical default for this value is 0.0.

Threshold

Type: **FLOAT**

The value of the threshold. A typical default for this value is 0.5.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SIGMOID_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs the sigmoid function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = 1 / (1 + \exp(-x))$$

Where $\exp(x)$ is the natural exponentiation function.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_SIGMOID_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SOFTMAX_OPERATOR_DESC structure (directml.h)

Article 07/20/2022

Performs a softmax activation function on *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
For 1-D InputTensor:  
// Let x[i] be the current element in the InputTensor, and j be the total  
// number of elements in the InputTensor  
f(x[i]) = exp(x[i]) / sum(exp(x[0]), ..., exp(x[j-1]))
```

Where $\exp(x)$ is the natural exponentiation function.

Syntax

C++

```
struct DML_ACTIVATION_SOFTMAX_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from. This tensor must have an *effective rank* no greater than 2. The effective rank of a tensor is the *DimensionCount* of the tensor, excluding leftmost dimensions of size 1. For example a tensor size of `{ 1, 1, BatchCount, Width }` is valid, and is equivalent to a tensor of sizes `{ BatchCount, Width }`.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC structure](#)

[DML_ARGMAX_OPERATOR_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SOFTMAX1_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Performs a softmax activation function on *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
For 1-D InputTensor:  
// Let x[i] to be the current element in the InputTensor, and j be the total  
// number of elements in the InputTensor  
f(x[i]) = exp(x[i]) / sum(exp(x[0]), ..., exp(x[j-1]))
```

Where $\exp(x)$ is the natural exponentiation function.

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) ↗ version 1.9 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_ACTIVATION_SOFTMAX1_OPERATOR_DESC  
{  
    const DML_TENSOR_DESC* InputTensor;  
    const DML_TENSOR_DESC* OutputTensor;  
    UINT AxisCount;  
    _Field_size_(AxisCount) const UINT* Axes;  
};
```

Members

InputTensor

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`AxisCount`

Type: `UINT`

The number of axes to calculate reduce sum. This field determines the size of the `Axes` array.

`Axes`

Type: `_Field_size_(AxisCount) const UINT*`

The axes along which to reduce the sum. Values must be in the range `[0, InputTensor.DimensionCount - 1]`.

Remarks

This operator is equivalent to `DML_ACTIVATION_SOFTMAX_OPERATOR_DESC` when `AxisCount == 1`, and `Axes == {DimensionCount - 1}`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_5_1`.

Tensor constraints

`InputTensor` and `OutputTensor` must have the same `DataType`, `DimensionCount`, and `Sizes`.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
<code>InputTensor</code>	Input	1 to 8	FLOAT32, FLOAT16
<code>OutputTensor</code>	Output	1 to 8	FLOAT32, FLOAT16

Requirements

 Expand table

Header	directml.h
--------	------------

See also

- [DML_ACTIVATION_LOG_SOFTMAX1_OPERATOR_DESC](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a parametric softplus activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \ln(1 + \exp(\text{Steepness} * x)) / \text{Steepness}$$

Where $\exp(x)$ is the natural exponentiation function and $\ln(x)$ is the natural logarithm.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Steepness;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The `Steepness` coefficient. A typical default for this value is 1.0. This value cannot be less than 1.

`Steepness`

Type: **FLOAT**

The steepness value.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML FEATURE LEVEL 3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML FEATURE LEVEL 2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML FEATURE LEVEL 1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16

[Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs the softsign function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = x / (1 + \text{abs}(x))$$

Where *abs(x)* returns the absolute value of *x*.

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_TANH_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a hyperbolic tangent activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = (1 - \exp(-2 * x)) / (1 + \exp(-2 * x)).$$

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_TANH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC structure](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a thresholded rectified linear unit (ReLU) activation function on every element in *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = x, if x > Alpha  
      0, otherwise
```

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Alpha;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Alpha

Type: **FLOAT**

The threshold for the function. A typical default for this value is 1.0.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

 Expand table

Requirement	Value
Header	directml.h

See also

[DML_ACTIVATION_SOFTMAX_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ADAM_OPTIMIZER_OPERATOR_DESC structure (directml.h)

Article 08/23/2022

Computes updated weights (parameters) using the supplied gradients, based on the Adam (ADAptive Moment estimation) algorithm. This operator is an optimizer, and is typically used in the weight update step of a training loop to perform gradient descent.

This operator performs the following computation:

```
X = InputParametersTensor
M = InputFirstMomentTensor
V = InputSecondMomentTensor
G = GradientTensor
T = TrainingStepTensor

// Compute updated first and second moment estimates.
M = Beta1 * M + (1.0 - Beta1) * G
V = Beta2 * V + (1.0 - Beta2) * G * G

// Compute bias correction factor for first and second moment estimates.
Alpha = sqrt(1.0 - pow(Beta2, T)) / (1.0 - pow(Beta1, T))

X -= LearningRate * Alpha * M / (sqrt(V) + Epsilon)

OutputParametersTensor = X
OutputFirstMomentTensor = M
OutputSecondMomentTensor = V
```

In addition to computing the updated weight parameters (returned in *OutputParametersTensor*), this operator also returns the updated first and second moment estimates in *OutputFirstMomentTensor* and *OutputSecondMomentTensor*, respectively. Typically, you should store these first and second moment estimates, and supply them as inputs during the subsequent training step.

Syntax

C++

```
struct DML_ADAM_OPTIMIZER_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputParametersTensor;
    const DML_TENSOR_DESC *InputFirstMomentTensor;
    const DML_TENSOR_DESC *InputSecondMomentTensor;
```

```
const DML_TENSOR_DESC *GradientTensor;
const DML_TENSOR_DESC *TrainingStepTensor;
const DML_TENSOR_DESC *OutputParametersTensor;
const DML_TENSOR_DESC *OutputFirstMomentTensor;
const DML_TENSOR_DESC *OutputSecondMomentTensor;
FLOAT LearningRate;
FLOAT Beta1;
FLOAT Beta2;
FLOAT Epsilon;
};
```

Members

`InputParametersTensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the parameters (weights) to apply this optimizer to. The gradient, first, and second moment estimates, current training step, as well as hyperparameters *LearningRate*, *Beta1*, and *Beta2*, are used by this operator to perform gradient descent on the weight values supplied in this tensor.

`InputFirstMomentTensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the first moment estimate of the gradient for each weight value. These values are typically obtained as the result of a previous execution of this operator, via the *OutputFirstMomentTensor*.

When applying this optimizer to a set of weights for the first time (for example, during the initial training step) the values of this tensor should typically be initialized to zero. Subsequent executions should use the values returned in *OutputFirstMomentTensor*.

The *Sizes* and *DataType* of this tensor must exactly match those of the *InputParametersTensor*.

`InputSecondMomentTensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the second moment estimate of the gradient for each weight value. These values are typically obtained as the result of a previous execution of this operator, via the *OutputSecondMomentTensor*.

When applying this optimizer to a set of weights for the first time (for example, during the initial training step) the values of this tensor should typically be initialized to zero. Subsequent executions should use the values returned in *OutputSecondMomentTensor*.

The *Sizes* and *DataType* of this tensor must exactly match those of the *InputParametersTensor*.

GradientTensor

Type: **const DML_TENSOR_DESC***

The gradients to apply to the input parameters (weights) for gradient descent. Gradients are typically obtained in a backpropagation pass during training.

The *Sizes* and *DataType* of this tensor must exactly match those of the *InputParametersTensor*.

TrainingStepTensor

Type: **const DML_TENSOR_DESC***

A scalar tensor containing a single integer value representing the current training step count. This value along with *Beta1* and *Beta2* is used to compute the exponential decay of the first and second moment estimate tensors.

Typically the training step value starts at 0 at the beginning of training, and is incremented by 1 on each successive training step. This operator doesn't update the training step value; you should do that manually, for example using [DML_OPERATOR_ELEMENT_WISE_ADD](#).

This tensor must be a scalar (that is, all *Sizes* equal to 1) and have data type [DML_TENSOR_DATA_TYPE_UINT32](#).

OutputParametersTensor

Type: **const DML_TENSOR_DESC***

An output tensor that holds the updated parameter (weight) values after gradient descent is applied.

During binding, this tensor is permitted to alias an eligible input tensor, which can be used to perform an in-place update of this tensor. See the [Remarks](#) section for more details.

The *Sizes* and *DataType* of this tensor must exactly match those of the *InputParametersTensor*.

`OutputFirstMomentTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor containing updated first moment estimates. You should store the values of this tensor, and supply them in *InputFirstMomentTensor* during the subsequent training step.

During binding, this tensor is permitted to alias an eligible input tensor, which can be used to perform an in-place update of this tensor. See the [Remarks](#) section for more details.

The *Sizes* and *DataType* of this tensor must exactly match those of the *InputParametersTensor*.

`OutputSecondMomentTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor containing updated second moment estimates. You should store the values of this tensor and supply them in *InputSecondMomentTensor* during the subsequent training step.

During binding, this tensor is permitted to alias an eligible input tensor, which can be used to perform an in-place update of this tensor. See the [Remarks](#) section for more details.

The *Sizes* and *DataType* of this tensor must exactly match those of the *InputParametersTensor*.

`LearningRate`

Type: `float`

The learning rate, also commonly referred to as the *step size*. The learning rate is a hyperparameter that determines the magnitude of the weight update along the gradient vector on each training step.

`Beta1`

Type: `float`

A hyperparameter representing the exponential decay rate of the gradient's first moment estimate. This value should be between 0.0 and 1.0. A value of 0.9f is typical.

`Beta2`

Type: `float`

A hyperparameter representing the exponential decay rate of the gradient's second moment estimate. This value should be between 0.0 and 1.0. A value of 0.999f is typical.

`Epsilon`

Type: `float`

A small value used to help numerical stability by preventing division-by-zero. For 32-bit floating-point inputs, typical values include 1e-8 or `FLT_EPSILON`.

Remarks

This operator supports in-place execution, meaning that each output tensor is permitted to alias an eligible input tensor during binding. For example, it's possible to bind the same resource for both the *InputParametersTensor* and *OutputParametersTensor* in order to effectively achieve an in-place update of the input parameters. All of this operator's input tensors, with the exception of the *TrainingStepTensor*, are eligible to be aliased in this way.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

GradientTensor, *InputFirstMomentTensor*, *InputParametersTensor*, *InputSecondMomentTensor*, *OutputFirstMomentTensor*, *OutputParametersTensor*, and *OutputSecondMomentTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
<code>InputParametersTensor</code>	Input	{ D0, D1, D2, D3 }	4	FLOAT32, FLOAT16

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputFirstMomentTensor	Input	{ D0, D1, D2, D3 }	4	FLOAT32, FLOAT16
InputSecondMomentTensor	Input	{ D0, D1, D2, D3 }	4	FLOAT32, FLOAT16
GradientTensor	Input	{ D0, D1, D2, D3 }	4	FLOAT32, FLOAT16
TrainingStepTensor	Input	{ 1, 1, 1, 1 }	4	FLOAT32, FLOAT16
OutputParametersTensor	Output	{ D0, D1, D2, D3 }	4	FLOAT32, FLOAT16
OutputFirstMomentTensor	Output	{ D0, D1, D2, D3 }	4	FLOAT32, FLOAT16
OutputSecondMomentTensor	Output	{ D0, D1, D2, D3 }	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_AVERAGE_POOLING_OPERATOR_DESCRIPTOR structure (directml.h)

Article 10/05/2021

Averages values across the elements within the sliding window over the input tensor.

Syntax

C++

```
struct DML_AVERAGE_POOLING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
    const UINT *EndPadding;
    BOOL IncludePadding;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

An input tensor of Sizes { BatchCount, ChannelCount, Height, Width } for 4D, and { BatchCount, ChannelCount, Depth, Height, Weight } for 5D.

OutputTensor

Type: **const DML_TENSOR_DESC***

A description of the output tensor. The sizes of the output tensor can be computed as follows.

C++

```
OutputTensor->Sizes[0] = InputTensor->Sizes[0];
OutputTensor->Sizes[1] = InputTensor->Sizes[1];

for (UINT i = 0; i < DimensionCount; ++i) {
```

```
    UINT PaddedSize = InputTensor->Sizes[i + 2] + StartPadding[i] +
EndPadding[i];
    OutputTensor->Sizes[i + 2] = (PaddedSize - WindowSizes[i]) / Strides[i]
+ 1;
}
```

DimensionCount

Type: **UINT**

The number of spatial dimensions of the input tensor *InputTensor*, which also corresponds to the number of dimensions of the sliding window *WindowSize*. This value also determines the size of the *Strides*, *StartPadding*, and *EndPadding* arrays. It should be set to 2 when *InputTensor* is 4D, and 3 when it's a 5D tensor.

Strides

Type: **_Field_size_(DimensionCount) const UINT***

The strides for the sliding window dimensions of sizes { Height, Width } when the *DimensionCount* is set to 2, or { Depth, Height, Width } when set to 3.

WindowSize

Type: **_Field_size_(DimensionCount) const UINT***

The dimensions of the sliding window in { Height, Width } when *DimensionCount* is set to 2, or { Depth, Height, Width } when set to 3.

StartPadding

Type: **_Field_size_(DimensionCount) const UINT***

The number of padding elements to be applied to the beginning of each spatial dimension of the input tensor *InputTensor*. The values are in { Height, Width } when *DimensionCount* is set to 2, or { Depth, Height, Width } when set to 3.

EndPadding

Type: **_Field_size_(DimensionCount) const UINT***

The number of padding elements to be applied to the end of each spatial dimension of the input tensor *InputTensor*. The values are in { Height, Width } when *DimensionCount* is set to 2, or { Depth, Height, Width } when set to 3.

IncludePadding

Type: **BOOL**

Indicates whether to include the padding elements around the spatial edges when calculating the average value across all elements within the sliding window. When the value is set to **FALSE**, the padding elements are not counted as part of the divisor value of the averaging calculation.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ARGMAX_OPERATOR_DESC structure (directml.h)

Article06/07/2022

Outputs the indices of the maximum-valued elements within one or more dimensions of the input tensor.

Each output element is the result of applying an *argmax* reduction on a subset of the input tensor. The *argmax* function outputs the index of the maximum-valued element within a set of input elements. The input elements involved in each reduction are determined by the provided input axes. Similarly, each output index is with respect to the provided input axes. If all input axes are specified, then the operator applies a single *argmax* reduction, and produces a single output element.

Syntax

C++

```
struct DML_ARGMAX_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT AxisCount;
    const UINT *Axes;
    DML_AXIS_DIRECTION AxisDirection;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. Each output element is the result of an *argmax* reduction on a subset of elements from the *InputTensor*.

- *DimensionCount* must match *InputTensor.DimensionCount* (the rank of the input tensor is preserved).
- *Sizes* must match *InputTensor.Sizes*, except for dimensions included in the reduced *Axes*, which must be size 1.

`AxisCount`

Type: **UINT**

The number of axes to reduce. This field determines the size of the *Axes* array.

`Axes`

Type: `_Field_size_(AxisCount) const UINT*`

The axes along which to reduce. Values must be in the range `[0, InputTensor.DimensionCount - 1]`.

`AxisDirection`

Type: **DML_AXIS_DIRECTION**

Determines which index to select when multiple input elements have the same value.

- **DML_AXIS_DIRECTION_INCREASING** returns the index of the first maximum-valued element (for example, `argmax({3,2,1,2,3}) = 0`)
- **DML_AXIS_DIRECTION_DECREASING** returns the index of the last maximum-valued element (for example, `argmax({3,2,1,2,3}) = 4`)

Examples

The examples in this section all use this same two-dimensional input tensor.

```
InputTensor: (Sizes:{3, 3}, DataType:FLOAT32)
[[1, 2, 3],
 [3, 0, 4],
 [2, 5, 2]]
```

Example 1. Applying *argmax* to columns

```
AxisCount: 1
Axes: {0}
AxisDirection: DML_AXIS_DIRECTION_INCREASING
OutputTensor: (Sizes:{1, 3}, DataType:UINT32)
[[1, // argmax({1, 3, 2})
 2, // argmax({2, 0, 5})
 1]] // argmax({3, 4, 2})
```

Example 2. Applying *argmax* to rows

```
AxisCount: 1
Axes: {1}
AxisDirection: DML_AXIS_DIRECTION_INCREASING
OutputTensor: (Sizes:{3, 1}, DataType:UINT32)
[[2], // argmax({1, 2, 3})
 [2], // argmax({3, 0, 4})
 [1]] // argmax({2, 5, 2})
```

Example 3. Applying *argmax* to all axes (the entire tensor)

```
AxisCount: 2
Axes: {0, 1}
AxisDirection: DML_AXIS_DIRECTION_INCREASING
OutputTensor: (Sizes:{1, 1}, DataType:UINT32)
[[7]] // argmax({1, 2, 3, 3, 0, 4, 2, 5, 2})
```

Remarks

The output tensor sizes must be the same as the input tensor sizes, except for the reduced axes, which must be 1.

When *AxisDirection* is [DML_AXIS_DIRECTION_INCREASING](#), this API is equivalent to [DML_REDUCE_OPERATOR_DESC](#) with [DML_REDUCE_FUNCTION_ARGMAX](#).

A subset of this functionality is exposed through the [DML_REDUCE_OPERATOR_DESC](#) operator, and is supported on earlier DirectML feature levels.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	INT64, INT32, UINT64, UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ARGMIN_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Outputs the indices of the minimum-valued elements within one or more dimensions of the input tensor.

Each output element is the result of applying an *argmin* reduction on a subset of the input tensor. The *argmin* function outputs the index of the minimum-valued element within a set of input elements. The input elements involved in each reduction are determined by the provided input axes. Similarly, each output index is with respect to the provided input axes. If all input axes are specified, then the operator applies a single *argmin* reduction, and produces a single output element.

Syntax

C++

```
struct DML_ARGMIN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT AxisCount;
    const UINT *Axes;
    DML_AXIS_DIRECTION AxisDirection;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. Each output element is the result of an *argmin* reduction on a subset of elements from the *InputTensor*.

- *DimensionCount* must match *InputTensor.DimensionCount* (the rank of the input tensor is preserved).
- *Sizes* must match *InputTensor.Sizes*, except for dimensions included in the reduced *Axes*, which must be size 1.

`AxisCount`

Type: **UINT**

The number of axes to reduce. This field determines the size of the *Axes* array.

`Axes`

Type: `_Field_size_(AxisCount) const UINT*`

The axes along which to reduce. Values must be in the range `[0, InputTensor.DimensionCount - 1]`.

`AxisDirection`

`DML_AXIS_DIRECTION AxisDirection;`

Type: **DML_AXIS_DIRECTION**

Determines which index to select when multiple input elements have the same value.

- **DML_AXIS_DIRECTION_INCREASING** returns the index of the first minimum-valued element (for example, `argmin({1,2,3,2,1}) = 0`)
- **DML_AXIS_DIRECTION_DECREASING** returns the index of the last minimum-valued element (for example, `argmin({1,2,3,2,1}) = 4`)

Examples

The examples in this section all use this same two-dimensional input tensor.

```
InputTensor: (Sizes:{3, 3}, DataType:FLOAT32)
[[1, 2, 3],
 [3, 0, 4],
 [2, 5, 2]]
```

Example 1. Applying *argmin* to columns

```
AxisCount: 1
Axes: {0}
AxisDirection: DML_AXIS_DIRECTION_INCREASING
OutputTensor: (Sizes:{1, 3}, DataType:UINT32)
[[0, // argmin({1, 3, 2})
 1, // argmin({2, 0, 5})
 2]] // argmin({3, 4, 2})
```

Example 2. Applying *argmin* to rows

```
AxisCount: 1
Axes: {1}
AxisDirection: DML_AXIS_DIRECTION_INCREASING
OutputTensor: (Sizes:{3, 1}, DataType:UINT32)
[[0], // argmin({1, 2, 3})
 [1], // argmin({3, 0, 4})
 [0]] // argmin({2, 5, 2})
```

Example 3. Applying *argmin* to all axes (the entire tensor)

```
AxisCount: 2
Axes: {0, 1}
AxisDirection: DML_AXIS_DIRECTION_INCREASING
OutputTensor: (Sizes:{1, 1}, DataType:UINT32)
[[4]] // argmin({1, 2, 3, 3, 0, 4, 2, 5, 2})
```

Remarks

The output tensor sizes must be the same as the input tensor sizes, except for the reduced axes, which must be 1.

When *AxisDirection* is [DML_AXIS_DIRECTION_INCREASING](#), this API is equivalent to [DML_REDUCE_OPERATOR_DESC](#) with [DML_REDUCE_FUNCTION_ARGMIN](#).

A subset of this functionality is exposed through the [DML_REDUCE_OPERATOR_DESC](#) operator, and is supported on earlier DirectML feature levels.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	INT64, INT32, UINT64, UINT32

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_AVERAGE_POOLING_GRAD_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Computes backpropagation gradients for average pooling (see [DML_AVERAGE_POOLING_OPERATOR_DESC](#)).

Consider a 2x2 **DML_AVERAGE_POOLING_OPERATOR_DESC**, without padding and a stride of 1, that performs the following.

```
InputTensor          OutputTensor
[[[1, 2, 3],     AvgPool  [[[3, 4],
    [4, 5, 6],      -->      [6, 7]]]
    [7, 8, 9]]]]
```

Each 2x2 window in the input tensor is averaged to produce one element of the output (reading zeros for elements beyond the edge). Here's an example of the output of **DML_AVERAGE_POOLING_GRAD_OPERATOR_DESC** given similar parameters.

```
InputGradientTensor      OutputGradientTensor
[[[1, 2],     AvgPoolGrad  [[[0.25, 0.75, 0.5],
    [3, 4]]]]      -->      [ 1, 2.5, 1.5],
                           [0.75, 1.75, 1]]]]
```

Notice that the values in the *OutputGradientTensor* represent the weighted contributions of that element to the *OutputTensor* during the original [DML_AVERAGE_POOLING_OPERATOR_DESC](#) operator.

Syntax

C++

```
struct DML_AVERAGE_POOLING_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
```

```
    const UINT          *EndPadding;
    BOOL                IncludePadding;
};
```

Members

InputGradientTensor

Type: **const DML_TENSOR_DESC***

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. Typically this tensor would have the same sizes as the *output* of the corresponding [DML_AVERAGE_POOLING_OPERATOR_DESC](#) in the forward pass.

OutputGradientTensor

Type: **const DML_TENSOR_DESC***

An output tensor containing the backpropagated gradients. Typically this tensor would have the same sizes as the *input* of the corresponding [DML_AVERAGE_POOLING_OPERATOR_DESC](#) in the forward pass.

DimensionCount

Type: **UINT**

The number of elements in the *Strides*, *WindowSize*, *StartPadding*, and *EndPadding* arrays. This value must equal the spatial dimension count. The spatial dimension count is 2 if 4D tensors are provided, or 3 if 5D tensors are provided.

Strides

Type: `_Field_size_(DimensionCount) const UINT*`

See *Strides* in [DML_AVERAGE_POOLING_OPERATOR_DESC](#).

WindowSize

Type: `_Field_size_(DimensionCount) const UINT*`

See *WindowSize* in [DML_AVERAGE_POOLING_OPERATOR_DESC](#).

StartPadding

Type: `_Field_size_(DimensionCount) const UINT*`

See *StartPadding* in [DML_AVERAGE_POOLING_OPERATOR_DESC](#).

EndPadding

Type: `_Field_size_(DimensionCount) const UINT*`

See *EndPadding* in [DML_AVERAGE_POOLING_OPERATOR_DESC](#).

IncludePadding

Type: `BOOL`

See *IncludePadding* in [DML_AVERAGE_POOLING_OPERATOR_DESC](#).

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputGradientTensor and *OutputGradientTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputGradientTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputGradientTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BATCH_NORMALIZATION_GRAD_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes backpropagation gradients for [batch normalization](#).

`DML_BATCH_NORMALIZATION_GRAD_OPERATOR_DESC` performs multiple computations, which are detailed in the separate output descriptions.

OutputScaleGradientTensor and *OutputBiasGradientTensor* are computed using sums across the set of dimensions for which *MeanTensor*, *ScaleTensor* and *VarianceTensor* sizes equal one.

Syntax

C++

```
struct DML_BATCH_NORMALIZATION_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *MeanTensor;
    const DML_TENSOR_DESC *VarianceTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    const DML_TENSOR_DESC *OutputScaleGradientTensor;
    const DML_TENSOR_DESC *OutputBiasGradientTensor;
    FLOAT Epsilon;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the input data. This is typically the same tensor that was provided as the *InputTensor* to [DML_BATCH_NORMALIZATION_OPERATOR_DESC](#) in the forward pass.

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer.

MeanTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the mean data. This is typically the same tensor that was provided as the *MeanTensor* to **DML_BATCH_NORMALIZATION_OPERATOR_DESC** in the forward pass.

VarianceTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the variance data. This is typically the same tensor that was provided as the *VarianceTensor* to **DML_OPERATOR_BATCH_NORMALIZATION** in the forward pass.

ScaleTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the scale data. This is typically the same tensor that was provided as the *ScaleTensor* to **DML_BATCH_NORMALIZATION_OPERATOR_DESC** in the forward pass.

OutputGradientTensor

Type: **const DML_TENSOR_DESC***

For every corresponding value in the inputs, `OutputGradient = InputGradient * (Scale / sqrt(Variance + Epsilon))`.

OutputScaleGradientTensor

Type: **const DML_TENSOR_DESC***

The following computation is done for every corresponding value in the inputs.

```
OutputScaleGradient = sum(InputGradient * (Input - Mean) / sqrt(Variance + Epsilon))
```

OutputBiasGradientTensor

Type: **const DML_TENSOR_DESC***

The following computation is done on every corresponding value in the inputs.

```
OutputBiasGradient = sum(InputGradient)
```

```
Epsilon
```

Type: **FLOAT**

A small value added to the variance to avoid zero.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_1`.

Tensor constraints

- *InputGradientTensor*, *InputTensor*, *MeanTensor*, *OutputBiasGradientTensor*, *OutputGradientTensor*, *OutputScaleGradientTensor*, *ScaleTensor*, and *VarianceTensor* must have the same *DataType* and *DimensionCount*.
- *MeanTensor*, *OutputBiasGradientTensor*, *OutputScaleGradientTensor*, *ScaleTensor*, and *VarianceTensor* must have the same *Sizes*.
- *InputGradientTensor*, *InputTensor*, and *OutputGradientTensor* must have the same *Sizes*.

Tensor support

[+] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
InputGradientTensor	Input	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
MeanTensor	Input	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
		}		
VarianceTensor	Input	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
ScaleTensor	Input	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputGradientTensor	Output	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputScaleGradientTensor	Output	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputBiasGradientTensor	Output	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BATCH_NORMALIZATION_OPERATOR_DESC structure (directml.h)

Article 12/02/2022

Performs a batch normalization on the input. This operator performs the following computation: `Output = FusedActivation(Scale * ((Input - Mean) / sqrt(Variance + Epsilon)) + Bias).`

Any dimension in *MeanTensor*, *VarianceTensor*, *ScaleTensor*, and *BiasTensor* can be set to 1, and be automatically broadcast to match *InputTensor*, but otherwise must equal the corresponding dimension's size from *InputTensor*.

Syntax

C++

```
struct DML_BATCH_NORMALIZATION_OPERATOR_DESC {
    const DML_TENSOR_DESC    *InputTensor;
    const DML_TENSOR_DESC    *MeanTensor;
    const DML_TENSOR_DESC    *VarianceTensor;
    const DML_TENSOR_DESC    *ScaleTensor;
    const DML_TENSOR_DESC    *BiasTensor;
    const DML_TENSOR_DESC    *OutputTensor;
    BOOL                     Spatial;
    FLOAT                    Epsilon;
    const DML_OPERATOR_DESC *FusedActivation;
};
```

Members

InputTensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the Input data.

MeanTensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the Mean data.

VarianceTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the Variance data.

ScaleTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the Scale data.

BiasTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the Bias data.

OutputTensor

Type: **const DML_TENSOR_DESC***

A tensor to write the results to.

Spatial

Type: **BOOL**

TRUE to specify that locations are spatial, otherwise **FALSE**. Setting this to **FALSE** will require the Width and Height dimensions of *MeanTensor* and *VarianceTensor* to not be broadcast. This parameter was deprecated in **DML_FEATURE_LEVEL_4_0**, and has no effect.

Epsilon

Type: **FLOAT**

The epsilon value to use to avoid division by zero.

FusedActivation

Type: **_Maybenull_ const DML_OPERATOR_DESC***

An optional fused activation layer to apply after the normalization. For more info, see [Using fused operators for improved performance](#).

Availability

This operator was introduced in **DML_FEATURE_LEVEL_1_0**.

Tensor constraints

- *BiasTensor*, *InputTensor*, *MeanTensor*, *OutputTensor*, *ScaleTensor*, and *VarianceTensor* must have the same *DataType* and *DimensionCount*.
- *InputTensor* and *OutputTensor* must have the same *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
MeanTensor	Input	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
VarianceTensor	Input	{ VarianceDimensions[] }	1 to 8	FLOAT32, FLOAT16
ScaleTensor	Input	{ ScaleDimensions[] }	1 to 8	FLOAT32, FLOAT16
BiasTensor	Input	{ BiasDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ InputDimensions[] }	4	FLOAT32, FLOAT16
MeanTensor	Input	{ MeanDimensions[] }	4	FLOAT32, FLOAT16
VarianceTensor	Input	{ VarianceDimensions[] }	4	FLOAT32, FLOAT16
ScaleTensor	Input	{ ScaleDimensions[] }	4	FLOAT32, FLOAT16
BiasTensor	Input	{ BiasDimensions[] }	4	FLOAT32, FLOAT16

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
OutputTensor	Output	{ InputDimensions[] }	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

See also

- [Using fused operators for improved performance](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_BATCH_NORMALIZATION_TRAINING_GRAD_OPERATOR_DESC structure (directml.h)

Article03/15/2023

Computes backpropagation gradients for [batch normalization training](#).

This operator performs multiple computations, which are detailed in the separate output descriptions.

Any dimension in *MeanTensor*, *VarianceTensor*, and *ScaleTensor* can be set to 1, and be automatically broadcast to match *InputTensor*, but otherwise must equal the corresponding dimension's size from *InputTensor*.

OutputScaleGradientTensor and *OutputBiasGradientTensor* are computed using sums across the set of dimensions for which *MeanTensor*, *ScaleTensor* and *VarianceTensor* sizes equal one.

Syntax

C++

```
struct DML_BATCH_NORMALIZATION_TRAINING_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *MeanTensor;
    const DML_TENSOR_DESC *VarianceTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    const DML_TENSOR_DESC *OutputScaleGradientTensor;
    const DML_TENSOR_DESC *OutputBiasGradientTensor;
    FLOAT Epsilon;
};
```

Members

InputTensor

Type: [const DML_TENSOR_DESC*](#)

A tensor containing the input data. This is typically the same tensor that was provided as the *InputTensor* to [DML_BATCH_NORMALIZATION_TRAINING_OPERATOR_DESC](#) in the forward pass.

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer.

`MeanTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the mean data. This is typically the same tensor that was returned by *MeanTensor* from [DML_BATCH_NORMALIZATION_TRAINING_OPERATOR_DESC](#) in the forward pass.

`VarianceTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the variance data. This is typically the same tensor that was returned as the *OutputVarianceTensor* from [DML_BATCH_NORMALIZATION_TRAINING_OPERATOR_DESC](#) in the forward pass.

`ScaleTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the scale data.

`OutputGradientTensor`

Type: `const DML_TENSOR_DESC*`

For every corresponding value in the inputs:

```
Coef0 = 1.0f / sqrt(Variance + Epsilon)
Coef1 = InputGradient * (Input - mean(Input))
InputGradientCentered = InputGradient - mean(InputGradient)
InputCentered = InputCentered - mean(InputCentered)
OutputGradient = Scale * Coef0 * (InputGradientCentered - InputCentered *
mean(Coef1) / (Variance + Epsilon))
```

`OutputScaleGradientTensor`

Type: `const DML_TENSOR_DESC*`

The following computation is done or every corresponding value in the inputs:

```
OutputScaleGradient = sum(InputGradient * (Input - Mean) / sqrt(Variance + Epsilon))
```

`OutputBiasGradientTensor`

Type: `const DML_TENSOR_DESC*`

The following computation is done or every corresponding value in the inputs:

```
OutputBiasGradient = sum(InputGradient)
```

`Epsilon`

Type: `FLOAT`

A small float value added to the variance to avoid zero.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_4_1`.

Tensor constraints

- *InputGradientTensor, InputTensor, MeanTensor, OutputBiasGradientTensor, OutputGradientTensor, OutputScaleGradientTensor, ScaleTensor, and VarianceTensor* must have the same *DataType* and *DimensionCount*.
- *MeanTensor, OutputBiasGradientTensor, OutputScaleGradientTensor, ScaleTensor, and VarianceTensor* must have the same *Sizes*.
- *InputGradientTensor, InputTensor, and OutputGradientTensor* must have the same *Sizes*.

Tensor support

`DML_FEATURE_LEVEL_4_1` and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
InputGradientTensor	Input	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
MeanTensor	Input	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
VarianceTensor	Input	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
ScaleTensor	Input	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputGradientTensor	Output	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputScaleGradientTensor	Output	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputBiasGradientTensor	Output	{ MeanDimensions[] }	1 to 8	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_BATCH_NORMALIZATION_TRAINING_OPERATOR_DESC structure (directml.h)

Article 12/02/2022

Performs a batch normalization on the input. This operator performs the following computation: `Output = FusedActivation(Scale * ((Input - Mean) / sqrt(Variance + Epsilon)) + Bias + FusedAdd).`

Any dimension in *ScaleTensor* and *BiasTensor* can be set to 1, and be automatically broadcast to match *InputTensor*, but otherwise must equal the corresponding dimension's size from *InputTensor*. *MeanTensor* and *VarianceTensor* are computed on the input across the set of dimensions for which *ScaleTensor* and *BiasTensor* sizes equal one.

Syntax

C++

```
struct DML_BATCH_NORMALIZATION_TRAINING_OPERATOR_DESC {
    const DML_TENSOR_DESC    *InputTensor;
    const DML_TENSOR_DESC    *ScaleTensor;
    const DML_TENSOR_DESC    *BiasTensor;
    const DML_TENSOR_DESC    *FusedAddTensor;
    const DML_TENSOR_DESC    *OutputTensor;
    const DML_TENSOR_DESC    *OutputMeanTensor;
    const DML_TENSOR_DESC    *OutputVarianceTensor;
    FLOAT                   Epsilon;
    const DML_OPERATOR_DESC *FusedActivation;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the Input data.

`ScaleTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the Scale data.

`BiasTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the Bias data.

`FusedAddTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing data that is added to the result prior to FusedActivation, if any.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor to write the results to.

`OutputMeanTensor`

Type: `const DML_TENSOR_DESC*`

A tensor to write the mean of the input to.

`OutputVarianceTensor`

Type: `const DML_TENSOR_DESC*`

A tensor to write the variance of the input to.

`Epsilon`

Type: `FLOAT`

The epsilon value to use to avoid division by zero.

`FusedActivation`

Type: `_Maybenull_ const DML_OPERATOR_DESC*`

An optional fused activation layer to apply after the normalization. For more info, see [Using fused operators for improved performance](#).

Remarks

Availability

This operator was introduced in [DML_FEATURE_LEVEL_4_1](#).

Tensor constraints

- *BiasTensor*, *FusedAddTensor*, *InputTensor*, *OutputMeanTensor*, *OutputTensor*, *OutputVarianceTensor*, and *ScaleTensor* must have the same *DataType* and *DimensionCount*.
- *BiasTensor*, *OutputMeanTensor*, *OutputVarianceTensor*, and *ScaleTensor* must have the same *Sizes*.
- *FusedAddTensor*, *InputTensor*, and *OutputTensor* must have the same *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[\]](#) Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
ScaleTensor	Input	{ ScaleDimensions[] }	1 to 8	FLOAT32, FLOAT16
BiasTensor	Input	{ ScaleDimensions[] }	1 to 8	FLOAT32, FLOAT16
FusedAddTensor	Optional input	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	{ InputDimensions[] }	1 to 8	FLOAT32, FLOAT16
OutputMeanTensor	Output	{ ScaleDimensions[] }	1 to 8	FLOAT32, FLOAT16

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
		}		
OutputVarianceTensor	Output	{ ScaleDimensions[] }	1 to 8	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

See also

- Using fused operators for improved performance

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BINDING_DESC structure (directml.h)

Article 02/22/2024

Contains the description of a binding so that you can add it to the binding table via a call to one of the [IDMLBindingTable](#) methods.

A binding can refer to an input or an output tensor resource, or to a persistent or a temporary resource, and there are methods on [IDMLBindingTable](#) to bind each kind. The type of the structure pointed to by *Desc* depends on the value of *Type*.

Syntax

C++

```
struct DML_BINDING_DESC {
    DML_BINDING_TYPE Type;
    const void       *Desc;
};
```

Members

Type

Type: [DML_BINDING_TYPE](#)

A [DML_BINDING_TYPE](#) specifying the type of the binding; whether it refers to a single buffer, or to an array of buffers.

Desc

Type: [const void*](#)

A pointer to a constant structure whose type depends on the value *Type*. If *Type* is [DML_BINDING_TYPE_BUFFER](#), then *Desc* should point to a [DML_BUFFER_BINDING](#). If *Type* is [DML_BINDING_TYPE_BUFFER_ARRAY](#), then *Desc* should point to a [DML_BUFFER_ARRAY_BINDING](#).

Requirements

Requirement	Value
Header	directml.h

See also

[Binding in DirectML](#)

[IDMLBindingTable::BindInputs](#)

[IDMLBindingTable::BindOutputs](#)

[IDMLBindingTable::BindPersistentResource](#)

[IDMLBindingTable::BindTemporaryResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BINDING_PROPERTIES structure (directml.h)

Article 02/22/2024

Contains information about the binding requirements of a particular compiled operator, or operator initializer. This struct is retrieved from [IDMLDispatchable::GetBindingProperties](#).

Syntax

C++

```
struct DML_BINDING_PROPERTIES {
    UINT RequiredDescriptorCount;
    UINT64 TemporaryResourceSize;
    UINT64 PersistentResourceSize;
};
```

Members

`RequiredDescriptorCount`

Type: [UINT](#)

The minimum size, in descriptors, of the binding table required for a particular dispatchable object (an operator initializer, or a compiled operator).

`TemporaryResourceSize`

Type: [UINT64](#)

The minimum size in bytes of the temporary resource that must be bound to the binding table for a particular dispatchable object. A value of zero means that a temporary resource is not required.

`PersistentResourceSize`

Type: [UINT64](#)

The minimum size in bytes of the persistent resource that must be bound to the binding table for a particular dispatchable object. Persistent resources must be supplied during initialization of a compiled operator (where it is bound as an output of the operator).

initializer) as well as during execution. A value of zero means that a persistent resource is not required. Only compiled operators have persistent resources—operator initializers always return a value of 0 for this member.

Requirements

 Expand table

Requirement	Value
Header	directml.h

See also

[Binding in DirectML](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_BINDING_TABLE_DESC structure (directml.h)

Article 02/22/2024

Specifies parameters to [IDMLDevice::CreateBindingTable](#) and [IDMLBindingTable::Reset](#).

Syntax

C++

```
struct DML_BINDING_TABLE_DESC {
    IDMLDispatchable           *Dispatchable;
    D3D12_CPU_DESCRIPTOR_HANDLE CPUDescriptorHandle;
    D3D12_GPU_DESCRIPTOR_HANDLE GPUDescriptorHandle;
    UINT                        SizeInDescriptors;
};
```

Members

`Dispatchable`

Type: [IDMLDispatchable*](#)

A pointer to an [IDMLDispatchable](#) interface representing the dispatchable object (an operator initializer, or a compiled operator) for which this binding table will represent the bindings—either an [IDMLCompiledOperator](#) or an [IDMLOperatorInitializer](#). The binding table maintains a strong reference to this interface pointer. This value may not be null.

`CPUDescriptorHandle`

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A valid CPU descriptor handle representing the start of a range into a constant buffer view (CBV)/shader resource view (SRV)/unordered access view (UAV) descriptor heap into which DirectML may write descriptors.

`GPUDescriptorHandle`

Type: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

A valid GPU descriptor handle representing the start of a range into a constant buffer view (CBV)/shader resource view (SRV)/unordered access view (UAV) descriptor heap that DirectML may use to bind resources to the pipeline.

`SizeInDescriptors`

Type: [UINT](#)

The size of the binding table, in descriptors. This is the maximum number of descriptors that DirectML is permitted to write, from the start of both the supplied CPU and GPU descriptor handles. Call [IDMLDispatchable::GetBindingProperties](#) to determine the number of descriptors required to execute a dispatchable object.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

See also

[Binding in DirectML](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BUFFER_ARRAY_BINDING structure (directml.h)

Article 02/22/2024

Specifies a resource binding that is an array of individual buffer bindings.

Syntax

C++

```
struct DML_BUFFER_ARRAY_BINDING {
    UINT BindingCount;
    const DML_BUFFER_BINDING *Bindings;
};
```

Members

BindingCount

Type: **UINT**

The number of individual buffer ranges to bind to this slot. This field determines the size of the *Bindings* array.

Bindings

Type: **const DML_BUFFER_BINDING***

The individual buffer ranges to bind.

Requirements

 Expand table

Requirement	Value
Header	directml.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BUFFER_BINDING structure (directml.h)

Article 02/22/2024

Specifies a resource binding described by a range of bytes in a Direct3D 12 buffer, represented by an offset and size into an [ID3D12Resource](#).

Syntax

C++

```
struct DML_BUFFER_BINDING {
    ID3D12Resource *Buffer;
    UINT64          Offset;
    UINT64          SizeInBytes;
};
```

Members

Buffer

Type: [ID3D12Resource](#)*

An optional pointer to an [ID3D12Resource](#) interface representing a buffer. The resource must have dimension [D3D12_RESOURCE_DIMENSION_BUFFER](#), and the range described by this struct must lie within the bounds of the buffer. You may supply `nullptr` for this member to indicate 'no binding'.

Offset

Type: [UINT64](#)

The offset, in bytes, from the start of the buffer where the range begins. This offset must be aligned to a multiple of [DML_MINIMUM_BUFFER_TENSOR_ALIGNMENT](#) or the [GuaranteedBaseOffsetAlignment](#) supplied as part of the [DML_BUFFER_TENSOR_DESC](#).

SizeInBytes

Type: [UINT64](#)

The size of the range, in bytes.

Requirements

 Expand table

Requirement	Value
Header	directml.h

See also

[Binding in DirectML](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BUFFER_TENSOR_DESC structure (directml.h)

Article 10/09/2023

Describes a tensor that will be stored in a Direct3D 12 buffer resource. The corresponding tensor type is [DML_TENSOR_TYPE_BUFFER](#), and the corresponding binding type is [DML_BINDING_TYPE_BUFFER](#).

Syntax

C++

```
struct DML_BUFFER_TENSOR_DESC {
    DML_TENSOR_DATA_TYPE DataType;
    DML_TENSOR_FLAGS      Flags;
    UINT                  DimensionCount;
    const UINT            *Sizes;
    const UINT            *Strides;
    UINT64                TotalTensorSizeInBytes;
    UINT                  GuaranteedBaseOffsetAlignment;
};
```

Members

DataType

Type: [DML_TENSOR_DATA_TYPE](#)

The type of the values in the tensor.

Flags

Type: [DML_TENSOR_FLAGS](#)

Specifies additional options for the tensor.

DimensionCount

Type: [UINT](#)

The number of dimensions of the tensor. This member determines the size of the *Sizes* and *Strides* arrays (if provided). In DirectML, the dimension count may range from 1 up to 8, depending on the operator. Most operators support at least 4 dimensions.

Sizes

Type: **const UINT***

The size, in elements, of each dimension in the tensor. Specifying a size of zero in any dimension is invalid, and will result in an error. For operators where the axes have semantic meaning (for example, batch, channel, depth, height, width), the *Sizes* member is always specified in the order {N, C, H, W} if *DimensionCount* is 4, and {N, C, D, H, W} if *DimensionCount* is 5. Otherwise, the dimensions generally have no particular meaning.

Strides

Type: **const UINT***

Optional. Determines the number of elements (not bytes) to linearly traverse in order to reach the next element in that dimension. For example, a stride of 5 in dimension 1 means that the distance between elements (n) and (n+1) in that dimension is 5 elements when traversing the buffer linearly. For operators where the axes have semantic meaning (for example, batch, channel, depth, height, width), the *Strides* member is always specified in the order {N, C, H, W} if *DimensionCount* is 4, and {N, C, D, H, W} if *DimensionCount* is 5.

Strides can be used to express broadcasting (by specifying a stride of 0) as well as padding (for example, by using a stride larger than the physical size of a row, to pad the end of a row).

If *Strides* is not specified, each dimension in the tensor is considered to be contiguously packed, with no additional padding.

TotalTensorSizeInBytes

Type: **UINT64**

Defines a minimum size in bytes for the buffer that will contain this tensor.

TotalTensorSizeInBytes must be at least as large as the minimum implied size given the sizes, strides, and data type of the tensor. You can calculate the minimum implied size by calling the [DMLCalcBufferTensorSize](#) utility free function.

Providing a *TotalTensorSizeInBytes* that is larger than the minimum implied size may enable additional optimizations by allowing DirectML to elide bounds checking in some cases if the *TotalTensorSizeInBytes* defines sufficient padding beyond the end of the tensor data.

When binding this tensor, the size of the buffer range must be at least as large as the *TotalTensorSizeInBytes*. For output tensors, this has the additional effect of permitting

DirectML to write to any memory within the *TotalTensorSizeInBytes*. That is, your application mustn't assume that DirectML will preserve any padding bytes inside output tensors that are inside the *TotalTensorSizeInBytes*.

The total size of a buffer tensor may not exceed $(2^{32} - 1)$ elements—for example, 16GB for a **FLOAT32** tensor.

GuaranteedBaseOffsetAlignment

Type: **UINT**

Optional. Defines a minimum guaranteed alignment in bytes for the base offset of the buffer range that will contain this tensor, or 0 to provide no minimum guaranteed alignment. If specified, this value must be a power of two that is at least as large as the element size.

When binding this tensor, the offset in bytes of the buffer range from the start of the buffer must be a multiple of the *GuaranteedBaseOffsetAlignment*, if provided.

Buffer tensors always have a minimum alignment of 16 bytes. However, providing a larger value for the *GuaranteedBaseOffsetAlignment* may allow DirectML to achieve better performance, because a larger alignment enables the use of vectorized load/store instructions.

Although this member is optional, for best performance we recommend that you align tensors to boundaries of 32 bytes or more, where possible.

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

See also

[Binding in DirectML](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_CAST_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Casts each element in the input to the data type of the output tensor, and stores the result in the corresponding element of the output.

Syntax

C++

```
struct DML_CAST_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. This tensor's `Sizes` should match `InputTensor`.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

Remarks

Some data types might not be supported on certain hardware. To determine whether a data type is supported, use [IDMLDevice::CheckFeatureSupport](#) with `DML_FEATURE_TENSOR_DATA_TYPE_SUPPORT`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount* and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8,

Tensor	Kind	Supported dimension counts	Supported data types
			UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT16, INT8, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT16, INT8, UINT16, UINT8

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_CONVOLUTION_INTEGER_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Performs a convolution of the *FilterTensor* with the *InputTensor*. This operator performs forward convolution on integer data. Optional zero point tensors can also be used to subtract zero point values from the input and filter tensor.

Syntax

C++

```
struct DML_CONVOLUTION_INTEGER_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *InputZeroPointTensor;
    const DML_TENSOR_DESC *FilterTensor;
    const DML_TENSOR_DESC *FilterZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT DimensionCount;
    const UINT Strides;
    const UINT Dilations;
    const UINT StartPadding;
    const UINT EndPadding;
    UINT GroupCount;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the input data. The expected dimensions of the *InputTensor* are `{ BatchCount, InputChannelCount, InputHeight, InputWidth }`.

`InputZeroPointTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the input zero point data. The expected dimensions of the *InputZeroPointTensor* are `{ 1, 1, 1, 1 }`.

`FilterTensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the filter data. The expected dimensions of the *FilterTensor* are {
FilterBatchCount, FilterChannelCount, FilterHeight, FilterWidth }.

FilterZeroPointTensor

Type: **_Maybenull_ const DML_TENSOR_DESC***

An optional tensor containing the filter zero point data. The expected dimensions of the *FilterZeroPointTensor* are { 1, 1, 1, 1 } if per tensor quantization is required, or { 1, OutputChannelCount, 1, 1 } if per-channel quantization is required.

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the results to. The expected dimensions of the *OutputTensor* are {
BatchCount, OutputChannelCount, OutputHeight, OutputWidth }.

DimensionCount

Type: **UINT**

The number of spatial dimensions for the convolution operation. Spatial dimensions are the lower dimensions of the convolution *FilterTensor*. This value also determines the size of the *Strides*, *Dilations*, *StartPadding*, and *EndPadding* arrays. Only a value of 2 is supported.

Strides

Type: **_Field_size_(DimensionCount) const UINT***

An array containing the strides of the convolution operation. These strides are applied to the convolution filter. They are separate from the tensor strides included in [DML_TENSOR_DESC](#).

Dilations

Type: **_Field_size_(DimensionCount) const UINT***

An array containing the dilations of the convolution operation. Dilations are strides applied to the elements of the filter kernel. This has the effect of simulating a larger filter kernel by padding the internal filter kernel elements with zeros.

StartPadding

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the padding values to be applied to the beginning of each spatial dimension of the filter and input tensor of the convolution operation.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the padding values to be applied to the end of each spatial dimension of the filter and input tensor of the convolution operation.

`GroupCount`

Type: `UINT`

The number of groups which to divide the convolution operation up into. *GroupCount* can be used to achieve depth-wise convolution by setting the *GroupCount* equal to the input channel count. This divides the convolution up into a separate convolution per input channel.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- *FilterZeroPointTensor* and *InputZeroPointTensor* must have the same *DimensionCount*.
- *FilterTensor*, *InputTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *InputTensor* and *InputZeroPointTensor* must have the same *DataType*.
- *FilterTensor* and *FilterZeroPointTensor* must have the same *DataType*.

Tensor support

`DML_FEATURE_LEVEL_4_0` and above

[+] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, [InputHeight], InputWidth }	3 to 4	INT8, UINT8
InputZeroPointTensor	Optional input	{ [1], [1], [1], 1 }	1 to 4	INT8, UINT8
FilterTensor	Input	{ FilterBatchCount, FilterChannelCount, [FilterHeight], FilterWidth }	3 to 4	INT8, UINT8
FilterZeroPointTensor	Optional input	{ [1], FilterZeroPointChannelCount, [1], [1] }	1 to 4	INT8, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, [OutputHeight], OutputWidth }	3 to 4	INT32

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	INT8, UINT8
InputZeroPointTensor	Optional input	{ 1, 1, 1, 1 }	4	INT8, UINT8
FilterTensor	Input	{ FilterBatchCount, FilterChannelCount, FilterHeight, FilterWidth }	4	INT8, UINT8
FilterZeroPointTensor	Optional input	{ 1, FilterZeroPointChannelCount, 1, 1 }	4	INT8, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	INT32

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_CONVOLUTION_OPERATOR_DESC structure (directml.h)

Article 12/02/2022

Performs a convolution of the *FilterTensor* with the *InputTensor*. This operator supports a number of standard convolution configurations. These standard configurations include forward and backward (transposed) convolution by setting the *Direction* and *Mode* fields, as well as depth-wise convolution by setting the *GroupCount* field.

A summary of the steps involved: perform the convolution into the output tensor; reshape the bias to the same dimension sizes as the output tensor; add the reshaped bias tensor to the output tensor.

Syntax

C++

```
struct DML_CONVOLUTION_OPERATOR_DESC {
    const DML_TENSOR_DESC      *InputTensor;
    const DML_TENSOR_DESC      *FilterTensor;
    const DML_TENSOR_DESC      *BiasTensor;
    const DML_TENSOR_DESC      *OutputTensor;
    DML_CONVOLUTION_MODE       Mode;
    DML_CONVOLUTION_DIRECTION  Direction;
    UINT                      DimensionCount;
    const UINT                 *Strides;
    const UINT                 *Dilations;
    const UINT                 *StartPadding;
    const UINT                 *EndPadding;
    const UINT                 *OutputPadding;
    UINT                      GroupCount;
    const DML_OPERATOR_DESC    *FusedActivation;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input data. The expected dimensions of the *InputTensor* are:

- { `BatchCount, InputChannelCount, InputWidth` } for 3D,
- { `BatchCount, InputChannelCount, InputHeight, InputWidth` } for 4D, and
- { `BatchCount, InputChannelCount, InputDepth, InputHeight, InputWidth` } for 5D.

FilterTensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the filter data. The expected dimensions of the *FilterTensor* are:

- `{ FilterBatchCount, FilterChannelCount, FilterWidth }` for 3D,
- `{ FilterBatchCount, FilterChannelCount, FilterHeight, FilterWidth }` for 4D, and
- `{ FilterBatchCount, FilterChannelCount, FilterDepth, FilterHeight, FilterWidth }` for 5D.

BiasTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the bias data. The bias tensor is a tensor containing data which is broadcasted across the output tensor at the end of the convolution which is added to the result. The expected dimensions of the *BiasTensor* are:

- `{ 1, OutputChannelCount, 1 }` for 3D,
- `{ 1, OutputChannelCount, 1, 1 }` for 4D, and
- `{ 1, OutputChannelCount, 1, 1, 1 }` for 5D.

For each output channel, the single bias value for that channel is added to every element in that channel of the *OutputTensor*. That is, the *BiasTensor* is broadcasted to the size of the *OutputTensor*, and what the operator returns is the summation of this broadcasted *BiasTensor* with the result from convolution.

OutputTensor

Type: `const DML_TENSOR_DESC*`

A tensor to write the results to. The expected dimensions of the *OutputTensor* are:

- `{ BatchCount, OutputChannelCount, OutputWidth }` for 3D,
- `{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }` for 4D, and
- `{ BatchCount, OutputChannelCount, OutputDepth, OutputHeight, OutputWidth }` for 5D.

Mode

Type: `DML_CONVOLUTION_MODE`

The mode to use for the convolution operation.

`DML_CONVOLUTION_MODE_CROSS_CORRELATION` is the behavior required for typical inference scenarios. In contrast, `DML_CONVOLUTION_MODE_CONVOLUTION` flips the order of elements in each filter kernel along each spatial dimension.

Direction

Type: [DML_CONVOLUTION_DIRECTION](#)

The direction of the convolution operation. [DML_CONVOLUTION_DIRECTION_FORWARD](#) is the primary form of convolution used for inference where a combination of [DML_CONVOLUTION_DIRECTION_FORWARD](#) and [DML_CONVOLUTION_DIRECTION_BACKWARD](#) are used during training.

`DimensionCount`

Type: [UINT](#)

The number of spatial dimensions for the convolution operation. Spatial dimensions are the lower dimensions of the convolution *FilterTensor*. For example, the width and height dimension are spatial dimensions of a 4D convolution filter tensor. This value also determines the size of the *Strides*, *Dilations*, *StartPadding*, *EndPadding*, and *OutputPadding* arrays. It should be set to 2 when *InputTensor.DimensionCount* is 4, and 3 when *InputTensor.DimensionCount* is 5.

`Strides`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the strides of the convolution operation. These strides are applied to the convolution filter. They are separate from the tensor strides included in [DML_TENSOR_DESC](#).

`Dilations`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the dilations of the convolution operation. Dilations are strides applied to the elements of the filter kernel. This has the effect of simulating a larger filter kernel by padding the internal filter kernel elements with zeros.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the padding values to be applied to the beginning of each spatial dimension of the filter and input tensor of the convolution operation. The start padding values are interpreted according to the *Direction* field.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the padding values to be applied to the end of each spatial dimension of the filter and input tensor of the convolution operation. The end padding values are interpreted according to the *Direction* field.

`OutputPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the output padding of the convolution operation. *OutputPadding* applies a zero padding to the result of the convolution. This padding is applied to the end of each spatial dimension of the output tensor.

`GroupCount`

Type: `UINT`

The number of groups which to divide the convolution operation up into. This can be used to achieve depth-wise convolution by setting *GroupCount* equal to the input channel count, and *Direction* equal to `DML_CONVOLUTION_DIRECTION_FORWARD`. This divides the convolution up into a separate convolution per input channel.

`FusedActivation`

Type: `_Maybenull_ const DML_OPERATOR_DESC*`

An optional fused activation layer to apply after the convolution. For more info, see [Using fused operators for improved performance](#).

Mode interactions

[+] Expand table

Convolution mode	Convolution direction	Filter orientation
<code>DML_CONVOLUTION_MODE_CROSS_CORRELATION</code>	<code>DML_CONVOLUTION_DIRECTION_FORWARD</code>	filter has identity orientation
<code>DML_CONVOLUTION_MODE_CROSS_CORRELATION</code>	<code>DML_CONVOLUTION_DIRECTION_BACKWARD</code>	filter is transposed along x,y axes
<code>DML_CONVOLUTION_MODE_CONVOLUTION</code>	<code>DML_CONVOLUTION_DIRECTION_FORWARD</code>	filter is transposed along x,y axes
<code>DML_CONVOLUTION_MODE_CONVOLUTION</code>	<code>DML_CONVOLUTION_DIRECTION_BACKWARD</code>	filter has identity orientation

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

BiasTensor, *FilterTensor*, *InputTensor*, and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_0 and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, [InputDepth], [InputHeight], InputWidth }	3 to 5	FLOAT32, FLOAT16
FilterTensor	Input	{ FilterBatchCount, FilterChannelCount, [FilterDepth], [FilterHeight], FilterWidth }	3 to 5	FLOAT32, FLOAT16
BiasTensor	Optional input	{ 1, OutputChannelCount, [1], [1], 1 }	3 to 5	FLOAT32, FLOAT16
OutputTensor	Output	{ BatchCount, OutputChannelCount, [OutputDepth], [OutputHeight], OutputWidth }	3 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, [InputDepth], InputHeight, InputWidth }	4 to 5	FLOAT32, FLOAT16
FilterTensor	Input	{ FilterBatchCount, FilterChannelCount, [FilterDepth], FilterHeight, FilterWidth }	4 to 5	FLOAT32, FLOAT16
BiasTensor	Optional input	{ 1, OutputChannelCount, [1], 1, 1 }	4 to 5	FLOAT32, FLOAT16

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
OutputTensor	Output	{ BatchCount, OutputChannelCount, [OutputDepth], OutputHeight, OutputWidth }	4 to 5	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

See also

- Using fused operators for improved performance

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_CUMULATIVE_PRODUCT_OPERATOR_DESC structure (directml.h)

Article 01/21/2022

Multiplies the elements of a tensor along an axis, writing the running tally of the product into the output tensor.

Syntax

C++

```
struct DML_CUMULATIVE_PRODUCT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT Axis;
    DML_AXIS_DIRECTION AxisDirection;
    BOOL HasExclusiveProduct;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the input data. This is typically the same tensor that was provided as the `InputTensor` to `DML_BATCH_NORMALIZATION_OPERATOR_DESC` in the forward pass.

The input tensor containing elements to be multiplied.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the resulting cumulative products to. This tensor must have the same sizes and data type as `InputTensor`.

`Axis`

Type: `UINT`

The index of the dimension to multiply elements over. This value must be less than the *DimensionCount* of the *InputTensor*.

AxisDirection

Type: [DML_AXIS_DIRECTION](#)

One of the values of the [DML_AXIS_DIRECTION](#) enumeration. If set to [DML_AXIS_DIRECTION_INCREASING](#), then the product occurs by traversing the tensor along the specified axis by ascending element index. If set to [DML_AXIS_DIRECTION_DECREASING](#), the reverse is true and the product occurs by traversing elements by descending index.

HasExclusiveProduct

Type: [BOOL](#)

If **TRUE**, then the value of the current element is excluded when writing the running tally to the output tensor. If **FALSE**, then the value of the current element is included in the running tally.

Examples

The examples in this section all use this same input tensor.

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[2, 1, 3, 5],
 [3, 8, 7, 3],
 [9, 6, 2, 4]]]
```

Example 1. Cumulative product across horizontal slivers

```
Axis: 3
AxisDirection: DML_AXIS_DIRECTION_INCREASING
HasExclusiveProduct: FALSE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[2, 2, 6, 30],      // i.e. [2, 2*1, 2*1*3, 2*1*3*5]
 [3, 24, 168, 504],   //      [...]
 [9, 54, 108, 432]]]] //      [...]
```

Example 2. Exclusive products

Setting *HasExclusiveProduct* to **TRUE** has the effect of excluding the current element's value from the running tally when writing to the output tensor.

```
Axis: 3
AxisDirection: DML_AXIS_DIRECTION_INCREASING
HasExclusiveProduct: TRUE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[1, 2, 2, 6], // Notice the product is written before multiplying
   [1, 3, 24, 168], // and the final total is not written to any
   [1, 9, 54, 108]]]]
```

Example 3. Axis direction

Setting the *AxisDirection* to **DML_AXIS_DIRECTION_DECREASING** has the effect of reversing the traversal order of elements when computing the running tally.

```
Axis: 3
AxisDirection: DML_AXIS_DIRECTION_DECREASING
HasExclusiveProduct: FALSE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[ 30, 15, 15, 5], // i.e. [2*1*3*5, 1*3*5, 3*5, 5]
   [504, 168, 21, 3], //      [...           ]
   [432, 48, 8, 4]]]] //      [...]         ]
```

Example 4. Multiplying along a different axis

In this example, the product occurs vertically, along the height axis (second dimension).

```
Axis: 2
AxisDirection: DML_AXIS_DIRECTION_INCREASING
HasExclusiveProduct: FALSE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[ 2, 1, 3, 5], // i.e. [2, ...]
```

```
[ 6, 8, 21, 15], // [2*3, ...]
[54, 48, 42, 60]]] // [2*3*9 ...]
```

Remarks

This operator supports in-place execution, meaning that the output tensor is permitted to alias *InputTensor* during binding.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_3_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_4_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_CUMULATIVE_SUMMATION_OPERATOR_DESC structure (directml.h)

Article 01/21/2022

Sums the elements of a tensor along an axis, writing the running tally of the summation into the output tensor.

Syntax

C++

```
struct DML_CUMULATIVE_SUMMATION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT Axis;
    DML_AXIS_DIRECTION AxisDirection;
    BOOL HasExclusiveSum;
};
```

Members

`InputTensor`

Type: **const DML_TENSOR_DESC***

The input tensor containing elements to be summed.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the resulting cumulative summations to. This tensor must have the same sizes and data type as the *InputTensor*.

`Axis`

Type: **UINT**

The index of the dimension to sum elements over. This value must be less than the *DimensionCount* of the *InputTensor*.

`AxisDirection`

Type: [DML_AXIS_DIRECTION](#)

One of the values of the [DML_AXIS_DIRECTION](#) enumeration. If set to [DML_AXIS_DIRECTION_INCREASING](#), then the summation occurs by traversing the tensor along the specified axis by ascending element index. If set to [DML_AXIS_DIRECTION_DECREASING](#), the reverse is true, and the summation occurs by traversing elements by descending index.

`HasExclusiveSum`

Type: [BOOL](#)

If [TRUE](#), then the value of the current element is excluded when writing the running tally to the output tensor. If [FALSE](#), then the value of the current element is included in the running tally.

Examples

The examples in this section all use an input tensor with the following properties.

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[2, 1, 3, 5],
   [3, 8, 7, 3],
   [9, 6, 2, 4]]]]
```

Example 1. Cumulative summation across horizontal slivers

```
Axis: 3
AxisDirection: DML_AXIS_DIRECTION_INCREASING
HasExclusiveSum: FALSE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[2, 3, 6, 11],    // i.e. [2, 2+1, 2+1+3, 2+1+3+5]
   [3, 11, 18, 21],  //      [...]
   [9, 15, 17, 21]]]] //      [...]
```

Example 2. Exclusive sums

Setting *HasExclusiveSum* to **TRUE** has the effect of excluding the current element's value from the running tally when writing to the output tensor.

```
Axis: 3
AxisDirection: DML_AXIS_DIRECTION_INCREASING
HasExclusiveSum: TRUE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[0, 2, 3, 6], // Notice the sum is written before adding the
   [0, 3, 11, 18], // and the final total is not written to any output.
   [0, 9, 15, 17]]]]
```

Example 3. Axis direction

Setting the *AxisDirection* to [DML_AXIS_DIRECTION_DECREASING](#) has the effect of reversing the traversal order of elements when computing the running tally.

```
Axis: 3
AxisDirection: DML_AXIS_DIRECTION_DECREASING
HasExclusiveSum: FALSE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[11, 9, 8, 5], // i.e. [2+1+3+5, 1+3+5, 3+5, 5]
   [21, 18, 10, 3], //      [...]
   [21, 12, 6, 4]]]] //      [...]
```

Example 4. Summing along a different axis

In this example, the summation occurs vertically, along the height axis (dimension 2).

```
Axis: 2
AxisDirection: DML_AXIS_DIRECTION_INCREASING
HasExclusiveSum: FALSE

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[ 2, 1, 3, 5], // i.e. [2, ...]
   [ 5, 9, 10, 8], //      [2+3, ...]
   [14, 15, 12, 12]]]] //      [2+3+9 ...]
```

Remarks

This operator supports in-place execution, meaning that the *OutputTensor* is permitted to alias the *InputTensor* during binding.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_4_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, UINT32

Requirements

[\[\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 [Yes](#) [No](#)[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_DEPTH_TO_SPACE_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Rearranges (permutes) data from depth into blocks of spatial data. The operator outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the height and width dimensions.

This is the inverse transformation of [DML_SPACE_TO_DEPTH_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_DEPTH_TO_SPACE_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT BlockSize;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to read from. The input tensor's dimensions are `{ BatchCount, InputChannelCount, InputHeight, InputWidth }`.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. The output tensor's dimensions are `{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }`, where:

- `OutputChannelCount` is computed as `InputChannelCount / (BlockSize * BlockSize)`.
- `OutputHeight` is computed as `InputHeight * BlockSize`.
- `OutputWidth` is computed as `InputWidth * BlockSize`.

`BlockSize`

Type: [UINT](#)

The width and height of the blocks that are moved.

Example

```
BlockSize: 2
InputTensor: (Sizes:{1, 8, 2, 3}, DataType:UINT32)
[[[0, 1, 2],
 [3, 4, 5]],
 [[9, 10, 11],
 [12, 13, 14]],
 [[18, 19, 20],
 [21, 22, 23]],
 [[27, 28, 29],
 [30, 31, 32]],
 [[36, 37, 38],
 [39, 40, 41]],
 [[45, 46, 47],
 [48, 49, 50]],
 [[54, 55, 56],
 [57, 58, 59]],
 [[63, 64, 65],
 [66, 67, 68]]]

OutputTensor: (Sizes:{1, 2, 4, 6}, DataType:UINT32)
[[[0, 18, 1, 19, 2, 20],
 [36, 54, 37, 55, 38, 56],
 [3, 21, 4, 22, 5, 23],
 [39, 57, 40, 58, 41, 59]],
 [[9, 27, 10, 28, 11, 29],
 [45, 63, 46, 64, 47, 65],
 [12, 30, 13, 31, 14, 32],
 [48, 66, 49, 67, 50, 68]]]
```

Remarks

A newer version of this operator, [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#), was introduced in [DML_FEATURE_LEVEL_2_1](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

See also

- [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#)
- [DML_SPACE_TO_DEPTH_OPERATOR_DESC](#)

Feedback

Was this page helpful?

[Yes](#)[No](#)[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_DEPTH_TO_SPACE1_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Rearranges (permutes) data from depth into blocks of spatial data. The operator outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the height and width dimensions.

This is the inverse transformation of [DML_SPACE_TO_DEPTH1_OPERATOR_DESC](#).

Syntax

```
C++  
  
struct DML_DEPTH_TO_SPACE1_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT BlockSize;  
    DML_DEPTH_SPACE_ORDER Order;  
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

The tensor to read from. The input tensor's dimensions are `{ BatchCount, InputChannelCount, InputHeight, InputWidth }`.

`OutputTensor`

Type: [const DML_TENSOR_DESC*](#)

The tensor to write the results to. The output tensor's dimensions are `{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }`, where:

- `OutputChannelCount` is computed as `InputChannelCount / (BlockSize * BlockSize)`
- `OutputHeight` is computed as `InputHeight * BlockSize`
- `OutputWidth` is computed as `InputWidth * BlockSize`

`BlockSize`

Type: [UINT](#)

The width and height of the blocks that are moved.

`Order`

Type: [DML_DEPTH_SPACE_ORDER](#)

See [DML_DEPTH_SPACE_ORDER](#).

Examples

The examples in this section all use the input below.

```
InputTensor: (Sizes:{1, 8, 2, 3}, DataType:UINT32)
[[[ [0, 1, 2],
      [3, 4, 5]],
    [[9, 10, 11],
     [12, 13, 14]],
   [[18, 19, 20],
    [21, 22, 23]],
  [[27, 28, 29],
   [30, 31, 32]],
 [[36, 37, 38],
  [39, 40, 41]],
 [[45, 46, 47],
  [48, 49, 50]],
 [[54, 55, 56],
  [57, 58, 59]],
 [[63, 64, 65],
  [66, 67, 68]]]]
```

Example 1. Depth-column-row order

```
BlockSize: 2
Order: DML_DEPTH_SPACE_ORDER_DEPTH_COLUMN_ROW
OutputTensor: (Sizes:{1, 2, 4, 6}, DataType:UINT32)
[[[ [ 0, 18, 1, 19, 2, 20],
      [36, 54, 37, 55, 38, 56],
      [ 3, 21, 4, 22, 5, 23],
      [39, 57, 40, 58, 41, 59]],
    [[ 9, 27, 10, 28, 11, 29],
```

```
[45, 63, 46, 64, 47, 65],  
[12, 30, 13, 31, 14, 32],  
[48, 66, 49, 67, 50, 68]]]
```

Example 2. Column-row-depth order

```
BlockSize: 2  
Order: DML_DEPTH_SPACE_ORDER_COLUMN_ROW_DEPTH  
OutputTensor: (Sizes:{1, 2, 4, 6}, DataType:UINT32)  
[[[[ 0, 9, 1, 10, 2, 11],  
   [18, 27, 19, 28, 20, 29],  
   [ 3, 12, 4, 13, 5, 14],  
   [21, 30, 22, 31, 23, 32]],  
  [[36, 45, 37, 46, 38, 47],  
   [54, 63, 55, 64, 56, 65],  
   [39, 48, 40, 49, 41, 50],  
   [57, 66, 58, 67, 59, 68]]]]
```

Remarks

When *Order* is set to [DML_DEPTH_SPACE_ORDER_DEPTH_COLUMN_ROW](#), [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#) is equivalent to [DML_DEPTH_TO_SPACE_OPERATOR_DESC](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*.

Tensor support

[DML_FEATURE_LEVEL_5_0 and above](#)

[] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- DML_DEPTH_TO_SPACE_OPERATOR_DESC
 - DML_SPACE_TO_DEPTH1_OPERATOR_DESC
-

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_DIAGONAL_MATRIX_OPERATOR_D ESC structure (directml.h)

Article 07/20/2022

Generates an identity-like matrix with ones (or other explicit value) on the major diagonal, and zeros everywhere else. The diagonal ones may be shifted (via *Offset*) where `OutputTensor[i, i + Offset]` = *Value*, meaning that an argument of *Offset* greater than zero shifts all values to the right, and less than zero shifts them to the left. This generator operator is useful for models to avoid storing a large constant tensor. Any leading dimensions before the last two are treated as a batch count, meaning that the tensor is treated as stack of 2D matrices.

This operator performs the following pseudocode.

```
for each coordinate in OutputTensor
    OutputTensor[coordinate] = if (coordinate.y + Offset == coordinate.x)
        then Value else 0
    endfor
```

Syntax

C++

```
struct DML_DIAGONAL_MATRIX_OPERATOR_DESC {
    const DML_TENSOR_DESC *OutputTensor;
    INT                 Offset;
    FLOAT              Value;
};
```

Members

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. The dimensions are `{ Batch1, Batch2, OutputHeight, OutputWidth }`. The height and width don't need to be square.

`Offset`

Type: **INT**

An offset to shift the diagonal lines of *Value*, with positive offsets shifting the written value to the right/up (viewing the output as a matrix with the top left as 0,0), and negative offsets to the left/down.

Value

Type: **FLOAT**

A value to fill along the 2D diagonal. The standard value is 1.0. Note that if the *DataType* of the tensors is not [DML_TENSOR_DATA_TYPE_FLOAT16](#) or [DML_TENSOR_DATA_TYPE_FLOAT32](#), then the value might be truncated (for example, 10.6 will become 10).

Examples

Default identity matrix:

```
Offset: 0
Value: 1.0
OutputTensor: (Sizes:{1,1,3,3}, DataType:FLOAT32)
[[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1]]]
```

Shift ones right/up:

```
Offset: 1
Value: 1.0
OutputTensor: (Sizes:{1,1,3,3}, DataType:FLOAT32)
[[[0, 1, 0],
 [0, 0, 1],
 [0, 0, 0]]]
```

Shift ones left/down:

```
Offset: -1
Value: 1.0
OutputTensor: (Sizes:{1,1,3,2}, DataType:FLOAT32)
```

```
[[[[0, 0],  
[1, 0],  
[0, 1]]]]
```

Shift the diagonal line of ones so far that all become zeroes:

```
Offset: -3  
Value: 1.0  
OutputTensor: (Sizes:{1,1,3,2}, DataType:FLOAT32)  
[[[[0, 0],  
[0, 0],  
[0, 0]]]]
```

Remarks

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_0](#).

Tensor support

DML_FEATURE_LEVEL_5_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	2 to 4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_4_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	2 to 4	FLOAT32, FLOAT16, INT32, INT16, INT8,

Tensor	Kind	Supported dimension counts	Supported data types
			UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_DIAGONAL_MATRIX1_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Generates an identity-like matrix with ones (or other explicit value) along the given diagonal span, with other elements being filled with either the input values or zeros (if no *InputTensor* is passed). The diagonal values may be shifted anywhere between *DiagonalFillBegin* and *DiagonalFillEnd*, where a value greater than zero shifts all values to the right, and less than zero shifts them to the left. This generator operator is useful for models to avoid storing a large constant tensor. Any leading dimensions before the last two are treated as a batch count, meaning that the tensor is treated as stack of 2D matrices. This operator performs the following pseudocode:

```
// Given each current coordinate's x, figure out the corresponding x on the
// top (0th) row.
// Then fill it with Value if that x coordinate is either within the fill
// bounds, or outside
// the fill bounds when inverted. Otherwise, use the original input value or
// zero.

for each coordinate in OutputTensor
    topX = coordinate.x - coordinate.y
    useValue = (DiagonalFillEnd >= DiagonalFillBegin) ^ (topX >=
DiagonalFillBegin) ^ (topX < DiagonalFillEnd)
    OutputTensor[coordinate] = if useValue then Value
                                else if InputTensor not null then
InputTensor[coordinate]
                                else 0
endfor
```

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) ↗ version 1.9 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_DIAGONAL_MATRIX1_OPERATOR_DESC
{
    _Maybenull_ const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* OutputTensor;
    DML_TENSOR_DATA_TYPE ValueDataType;
    DML_SCALAR_UNION Value;
    INT DiagonalFillBegin;
    INT DiagonalFillEnd;
};
```

Members

`InputTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional input tensor.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. The dimensions are `{ Batch1, Batch2, OutputHeight, OutputWidth }`. The height and width don't need to be square.

`ValueDataType`

Type: `DML_TENSOR_DATA_TYPE`

The data type of the `Value` member, which must match `OutputTensor::DataType`.

`Value`

Type: `DML_SCALAR_UNION`

A constant value to fill the output, with `ValueDataType` determining how to interpret the member.

`DiagonalFillBegin`

Type: `INT`

The lower limit (beginning inclusive) of the range to fill with `Value`. If the beginning and ending are inverted (`begin > end`), then the fill will be reversed.

`DiagonalFillEnd`

Type: [INT](#)

The upper limit (ending exclusive) of the range to fill with *Value*. If the beginning and ending are inverted (begin > end), then the fill will be reversed.

Examples

Default identity matrix:

```
InputTensor: none
ValueType: FLOAT32
Value: 7
DiagonalFillBegin: 0
DiagonalFillEnd: 1
OutputTensor: (Sizes:{4,5} DataType:FLOAT32)
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0]]
```

Fill a 3-element wide diagonal strip:

```
InputTensor: none
ValueType: FLOAT32
Value: 7
DiagonalFillBegin: 0
DiagonalFillEnd: 3
OutputTensor: (Sizes:{4,5} DataType:FLOAT32)
[[7, 7, 7, 0, 0],
 [0, 7, 7, 7, 0],
 [0, 0, 7, 7, 7],
 [0, 0, 0, 7, 7]]
```

Keep top-right diagonal half (the strictly upper triangle), zeroing the bottom left.

```
InputTensor: (Sizes:{4,5} DataType:FLOAT32)
[[4, 7, 3, 7, 9],
 [1, 2, 8, 6, 9],
 [9, 4, 1, 8, 7],
 [4, 3, 4, 2, 4]]
ValueType: FLOAT32
Value: 0
DiagonalFillBegin: INT32_MIN
```

```
DiagonalFillEnd: 1
OutputTensor: (Sizes:{4,5} DataType:FLOAT32)
    [[0, 7, 3, 7, 9],
     [0, 0, 8, 6, 9],
     [0, 0, 0, 8, 7],
     [0, 0, 0, 0, 4]]
```

Keep only the values along the diagonal, zeroing all others.

```
InputTensor: (Sizes:{4,5} DataType:FLOAT32)
    [[4, 7, 3, 7, 9],
     [1, 2, 8, 6, 9],
     [9, 4, 1, 8, 7],
     [4, 3, 4, 2, 4]]
ValueDataType: FLOAT32
Value: 0
DiagonalFillBegin: 1
DiagonalFillEnd: 0
OutputTensor: (Sizes:{4,5} DataType:FLOAT32)
    [[4, 0, 0, 0, 0],
     [0, 2, 0, 0, 0],
     [0, 0, 1, 0, 0],
     [0, 0, 0, 2, 0]]
```

Remarks

This operator is equivalent to [DML_DIAGONAL_MATRIX_OPERATOR_DESC](#) when *DiagonalFillBegin* is set to *DML_OPERATOR_DIAGONAL_MATRIX::Offset*, and *DiagonalFillEnd* is set to *DML_OPERATOR_DIAGONAL_MATRIX::Offset + 1*.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_5_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Optional input	2 to 4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	2 to 4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Header	directml.h
--------	------------

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_DYNAMIC_QUANTIZE_LINEAR_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Calculates the quantization scale and zero point values necessary to quantize the *InputTensor*, then applies that quantization, writing the result to *OutputTensor*.

This operator uses the following equation to quantize.

```
InputMax = Max(InputTensor)
InputMin = Min(InputTensor)

AValue = (A - AZeroPoint) * AScale
BValue = (B - BZeroPoint) * BScale

// For uint8 output, Min = 0, Max = 255
// For int8 output, Min = -128, Max = 127
OutputScale = (InputMax - InputMin) / (Max - Min)

OutputZeroPoint = Min - InputMin / OutputScale

OutputTensor = clamp(round(InputValue / OutputScale) + OutputZeroPoint, Min,
Max)
```

Syntax

C++

```
struct DML_DYNAMIC_QUANTIZE_LINEAR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_TENSOR_DESC *OutputScaleTensor;
    const DML_TENSOR_DESC *OutputZeroPointTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor containing the inputs.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`OutputScaleTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the output scale factor for *OutputTensor*. The expected number of elements in *OutputScaleTensor* is 1.

`OutputZeroPointTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the output zero point for *OutputTensor*. The expected number of elements in *OutputZeroPointTensor* is 1.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_4_0`.

Tensor constraints

- *InputTensor*, *OutputScaleTensor*, *OutputTensor*, and *OutputZeroPointTensor* must have the same *DimensionCount*.
- *OutputTensor* and *OutputZeroPointTensor* must have the same *DataType*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	INT8, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
OutputScaleTensor	Output	1 to 8	FLOAT32
OutputZeroPointTensor	Output	1 to 8	INT8, UINT8

Expand table

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	directml.h

See also

- [DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ABS_OPERATOR_DESCRIPTOR structure (directml.h)

Article 02/22/2024

Computes the absolute value for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = abs(x)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ABS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[\[+\] Yes](#)

[\[+\] No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ACOS_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the arccosine for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \arccos(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ACOS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Computes the hyperbolic arccosine for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = acosh(x) // ln(x + sqrt(x * x - 1))
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ADD_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Adds every element in *ATensor* to its corresponding element in *BTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = a + b
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one or more of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ADD_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Remarks

A newer version of this operator, [DML_ELEMENT_WISE_ADD1_OPERATOR_DESC](#), was introduced in [DML_FEATURE_LEVEL_2_0](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_4_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4 to 5	FLOAT32, FLOAT16
BTensor	Input	4 to 5	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

 Expand table

Requirement	Value
Header	directml.h

See also

[DML_ELEMENT_WISE_ADD1_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ADD1_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Adds every element in *ATensor* to its corresponding element in *BTensor* and places the result into the corresponding element of *OutputTensor*, with the option for fused activation.

```
f(a, b) = FusedActivation(a + b)
```

The fused activation operator description, if provided, then executes the given activation operator on the output.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one or more of the input tensors during binding.

Syntax

```
C++  
  
struct DML_ELEMENT_WISE_ADD1_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_OPERATOR_DESC *FusedActivation;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`FusedActivation`

Type: `_Maybenull_ const DML_OPERATOR_DESC*`

An optional fused activation layer to apply after the addition. For more info, see [Using fused operators for improved performance](#).

Fused activation may be used only when the output datatype is `FLOAT16` or `FLOAT32`.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

`DML_FEATURE_LEVEL_3_0` and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
<code>ATensor</code>	Input	1 to 8	<code>FLOAT32</code> , <code>FLOAT16</code>
<code>BTensor</code>	Input	1 to 8	<code>FLOAT32</code> , <code>FLOAT16</code>
<code>OutputTensor</code>	Output	1 to 8	<code>FLOAT32</code> , <code>FLOAT16</code>

`DML_FEATURE_LEVEL_2_0` and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4 to 5	FLOAT32, FLOAT16
BTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

See also

- [Using fused operators for improved performance](#)

Feedback

Was this page helpful?

[!\[\]\(03521717e54239a4a58ea5d917930916_img.jpg\) Yes](#)[!\[\]\(f7792b40947a46857d484f2f1ad17204_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ASIN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the arcsine for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = asin(x)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ASIN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ASINH_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the hyperbolic arcsine for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = asinh(x) // ln(x + sqrt(x * x + 1))
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ASINH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ATAN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the arctangent for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = atan(x)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ATAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ATANH_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the hyperbolic arctangent for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = atanh(x) // ln((1 + x) / (1 - x)) / 2
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ATANH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ATAN_YX_OPERATOR_DESC structure (directml.h)

Article 08/23/2022

Computes the 2-argument arctangent for each element of *ATensor* and *BTensor*, where *ATensor* is the Y-axis and *BTensor* is the X-axis, placing the result into the corresponding element of *OutputTensor*. This operator is undefined for the origin (that is, when *ATensor* and *BTensor* are both 0 for corresponding elements).

$$\text{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y > 0, \\ -\frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y < 0, \\ \arctan\left(\frac{y}{x}\right) \pm \pi & \text{if } x < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

```
f(y, x) = atan2(y, x)
```

This operator supports in-place execution, meaning that the output tensor is permitted to alias *ATensor* or *BTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ATAN_YX_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: `const DML_TENSOR_DESC*`

The input tensor to read the Y-axis values from.

BTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read the X-axis values from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_1`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16
BTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_BIT_AND_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Computes the bitwise AND between each corresponding element of the input tensors, and writes the result into the output tensor.

The bitwise operation is applied to tensor data in its native encoding. Therefore, the tensor data type is ignored except for determining the width of each element.

This operator supports in-place execution, meaning the output tensor is permitted to alias one or more of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_BIT_AND_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT16, UINT8
BTensor	Input	1 to 8	UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT16, UINT8

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_BIT_COUNT_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the bitwise population count (the number of bits set to 1) for each element of the input tensor, and writes the result into the output tensor.

The bitwise operation is applied to tensor data in its native encoding. Therefore, the tensor data type is ignored except for determining the width of each element.

Syntax

C++

```
struct DML_ELEMENT_WISE_BIT_COUNT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Example

```
InputTensor: (Sizes:{2,2}, DataType:UINT32)
[[0, 123], // 0b000000000000, 0b0001111011
 [456, 789]] // 0b0111001000, 0b1100010101

OutputTensor: (Sizes:{2,2}, DataType:UINT32)
```

```
[[0, 6],  
 [4, 5]]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount* and *Sizes*.

Tensor support

DML FEATURE LEVEL 4_1 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML FEATURE LEVEL 3_0 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_BIT_NOT_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the bitwise NOT for each element of the input tensor, and writes the result into the output tensor.

The bitwise operation is applied to tensor data in its native encoding. Therefore, the tensor data type is ignored except for determining the width of each element.

This operator supports in-place execution, meaning that the output tensor is permitted to alias the input tensor during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_BIT_NOT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Example

```

InputTensor: (Sizes:{2,2}, DataType:UINT8)
[[0, 128], // 0b00000000, 0b10000000
 [42, 255]] // 0b00101010, 0b11111111

OutputTensor: (Sizes:{2,2}, DataType:UINT8)
[[255, 127], // 0b11111111, 0b01111111
 [213, 0 ]] // 0b11010101, 0b00000000

```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT16, UINT8

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_BIT_OR_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Computes the bitwise OR between each corresponding element of the input tensors, and writes the result into the output tensor.

The bitwise operation is applied to tensor data in its native encoding. Therefore, the tensor data type is ignored except for determining the width of each element.

This operator supports in-place execution, meaning that the output tensor is permitted to alias one or more of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_BIT_OR_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT16, UINT8
BTensor	Input	1 to 8	UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_BIT_SHIFT_LEFT_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a logical left shift of each element of *ATensor* by a number of bits given by the corresponding element of *BTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = (a << b)
```

The bitwise operation is applied to tensor data in its native encoding. Therefore, the tensor data type is ignored except for determining the width of each element.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_BIT_SHIFT_LEFT_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT16, UINT8
BTensor	Input	1 to 8	UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32, UINT16, UINT8
BTensor	Input	4	UINT32, UINT16, UINT8
OutputTensor	Output	4	UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[!\[\]\(3d44da46260913e4f03ae4326c065ea0_img.jpg\) Yes](#)[!\[\]\(20866737474b1071a02f6425e7538cab_img.jpg\) No](#)[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_BIT_SHIFT_RIGHT_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Performs a logical right shift of each element of *ATensor* by a number of bits given by the corresponding element of *BTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = (a >> b)
```

The bitwise operation is applied to tensor data in its native encoding. Therefore, the tensor data type is ignored except for determining the width of each element.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_BIT_SHIFT_RIGHT_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT16, UINT8
BTensor	Input	1 to 8	UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32, UINT16, UINT8
BTensor	Input	4	UINT32, UINT16, UINT8
OutputTensor	Output	4	UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[!\[\]\(24f5a142ba94a7431f11ea51182cd094_img.jpg\) Yes](#)[!\[\]\(ac6a1d4548e189f0fe4afc2342b0e741_img.jpg\) No](#)[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_BIT_XOR_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the bitwise XOR (eXclusive OR) between each corresponding element of the input tensors, and writes the result into the output tensor.

The bitwise operation is applied to tensor data in its native encoding. Therefore, the tensor data type is ignored except for determining the width of each element.

This operator supports in-place execution, meaning that the output tensor is permitted to alias one or more of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_BIT_XOR_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Example

```
InputTensor: (Sizes:{2,2}, DataType:UINT8)
[[0, 128], // 0b00000000, 0b10000000
 [42, 255]] // 0b00101010, 0b11111111

OutputTensor: (Sizes:{2,2}, DataType:UINT8)
[[255, 127], // 0b11111111, 0b01111111
 [213, 0]] // 0b11010101, 0b00000000
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT16, UINT8
BTensor	Input	1 to 8	UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)[No](#)[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_CEIL_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the ceiling for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*. The ceiling of x is the smallest integer that is greater than or equal to x .

```
f(x) = ceil(x)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_CEIL_OPERATOR_DESCRIPTOR {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_CLIP_GRAD_OPERATOR_DESC structure (directml.h)

Article 08/23/2022

Computes backpropagation gradients for [element-wise clip](#).

```
f(x, gradient) = if x <= Min then 0  
                  if x >= Max then 0  
                  else      then gradient
```

This operator supports in-place execution, meaning `OutputTensor` is permitted to alias `InputTensor` during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_CLIP_GRAD_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *InputGradientTensor;  
    const DML_TENSOR_DESC *OutputGradientTensor;  
    FLOAT             Min;  
    FLOAT             Max;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input feature tensor. This is typically the same tensor that was provided as the `InputTensor` to [DML_ELEMENT_WISE_CLIP_OPERATOR_DESC](#) in the forward pass.

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. Typically this tensor would have the same sizes as

the *output* of the corresponding `DML_OPERATOR_ELEMENT_WISE_CLIP` in the forward pass.

`OutputGradientTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor containing the backpropagated gradients. Typically this tensor would have the same sizes as the *input* of the corresponding `DML_OPERATOR_ELEMENT_WISE_CLIP` in the forward pass.

`Min`

Type: `FLOAT`

The minimum value. If x is at or below this value, then the gradient result is 0.

`Max`

Type: `FLOAT`

The maximum value. If x is at or above this value, then the gradient result is 0.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_1`.

Tensor constraints

InputGradientTensor, *InputTensor*, and *OutputGradientTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

`DML_FEATURE_LEVEL_5_0` and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
InputGradientTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputGradientTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
InputGradientTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputGradientTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

DML_ELEMENT_WISE_CLIP_GRAD1_OPERATOR_DESC structure (directml.h)

Article03/15/2023

Computes backpropagation gradients for [DML_OPERATOR_ELEMENT_WISE_CLIP1](#).

```
f(x, gradient) = if x <= Min then 0  
                  if x >= Max then 0  
                  else      then gradient
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_CLIP_GRAD1_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *InputGradientTensor;  
    const DML_TENSOR_DESC *OutputGradientTensor;  
    DML_TENSOR_DATA_TYPE  MinMaxDataType;  
    DML_SCALAR_UNION      Min;  
    DML_SCALAR_UNION      Max;  
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

The input feature tensor. This is typically the same tensor that was provided as the *InputTensor* to [DML_OPERATOR_ELEMENT_WISE_CLIP1](#) in the forward pass.

`InputGradientTensor`

Type: [const DML_TENSOR_DESC*](#)

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. Typically this tensor would have the same sizes as the *output* of the corresponding [DML_OPERATOR_ELEMENT_WISE_CLIP1](#) in the forward pass.

`OutputGradientTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor containing the backpropagated gradients. Typically this tensor would have the same sizes as the *input* of the corresponding [DML_OPERATOR_ELEMENT_WISE_CLIP1](#) in the forward pass.

`MinMaxDataType`

Type: `DML_TENSOR_DATA_TYPE`

The data type of the *Min* and *Max* members, which must match *OutputTensor.DataType*.

`Min`

Type: `DML_SCALAR_UNION`

The minimum value. If x is at or below this value, then the gradient result is 0. *MinMaxDataType* determines how to interpret the field.

`Max`

Type: `DML_SCALAR_UNION`

The maximum value. If x is at or above this value, then the gradient result is 0. *MinMaxDataType* determines how to interpret the field.

Remarks

Availability

This operator was introduced in [DML_FEATURE_LEVEL_5_0](#).

Tensor constraints

InputGradientTensor, *InputTensor*, and *OutputGradientTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
InputGradientTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputGradientTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_CLIP_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs the following operation for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*. This operator clamps (or limits) every element in the input within the closed interval [Min, Max].

$$f(x) = \max(\text{Min}, \min(x, \text{Max}))$$

Where `max(a,b)` returns the larger of the two values, and `min(a,b)` returns the smaller of the two values `a,b`.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

```
C++  
  
struct DML_ELEMENT_WISE_CLIP_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
    FLOAT Min;  
    FLOAT Max;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

ScaleBias

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each *input* element prior to computing this operator.

Min

Type: `FLOAT`

The minimum value, below which the operator replaces the value with *Min*.

Max

Type: `FLOAT`

The maximum value, above which the operator replaces the value with *Max*.

Remarks

If the tensor data type is not `float`, then *Min* and *Max* are cast to the tensor data type before applying the clipping operation (which for integers means truncating toward zero; and for floating point values rounding to the nearest even).

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

`DML_FEATURE_LEVEL_5_0` and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

 Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_CLIP1_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs the following operation for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*. This operator clamps (or limits) every element in the input within the closed interval $[Min, Max]$.

$$f(x) = \max(\text{Min}, \min(x, \text{Max}))$$

Where `max(a,b)` returns the larger of the two values, and `min(a,b)` returns the smaller of the two values a,b.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_CLIP1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
    DML_TENSOR_DATA_TYPE MinMaxDataType;
    DML_SCALAR_UNION     Min;
    DML_SCALAR_UNION     Max;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

ScaleBias

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

MinMaxDataType

Type: `DML_TENSOR_DATA_TYPE`

The data type of the *Min* and *Max* members, which must match *OutputTensor.DataType*.

Min

Type: `DML_SCALAR_UNION`

The minimum value, below which the operator replaces the value with *Min*. *MinMaxDataType* determines how to interpret the field.

Max

Type: `DML_SCALAR_UNION`

The maximum value, above which the operator replaces the value with *Max*. *MinMaxDataType* determines how to interpret the field.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_5_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[!\[\]\(5d565beb3ca65c6dafa206d98259e47a_img.jpg\) Yes](#)[!\[\]\(5001ff934faea44756130a23a2560b6e_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Raises each element of *InputTensor* to the power of *Exponent*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = pow(x, Exponent)
```

Negative bases are supported for integral exponents, otherwise this operator returns NaN.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
    FLOAT                 Exponent;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each *input* element prior to computing this operator.

Exponent

Type: `FLOAT`

The exponent that all inputs will be raised to.

Remarks

Also see the POW operator `DML_ELEMENT_WISE_POW_OPERATOR_DESC`, which accepts a second tensor as exponents.

Requirements

[] [Expand table](#)

Requirement	Value
Header	<code>directml.h</code>

See also

[DML_ELEMENT_WISE_POW_OPERATOR_DESC](#)

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_COS_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the trigonometric cosine of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \cos(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_COS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_COSH_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the hyperbolic cosine of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \cosh(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_COSH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC structure (directml.h)

Article 11/18/2024

Performs the following linear dequantization function on every element in *InputTensor* with respect to its corresponding element in *ScaleTensor* and *ZeroPointTensor*, placing the results in the corresponding element of *OutputTensor*.

```
f(input, scale, zero_point) = (input - zero_point) * scale
```

Quantization is a common way to increase performance at the cost of precision. A group of 8-bit int values can be computed faster than a group of 32-bit float values can. Dequantizing converts the encoded data back to its domain.

Syntax

C++

```
struct DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *ZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the inputs.

`ScaleTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the scales. A scale value of 0 will result in undefined behavior.

ⓘ Note

A scale value of 0 results in undefined behavior.

`ZeroPointTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the zero point that was used for quantization.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *InputTensor*, *OutputTensor*, *ScaleTensor*, and *ZeroPointTensor* must have the same *DimensionCount* and *Sizes*.
- *InputTensor* and *ZeroPointTensor* must have the same *DataType*.
- *OutputTensor* and *ScaleTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_6_2 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	INT32, INT16, INT8, UINT32, UINT16, UINT8
ScaleTensor	Input	1 to 8	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
ZeroPointTensor	Optional input	1 to 8	INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_6_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	INT32, INT16, INT8, UINT32, UINT16, UINT8
ScaleTensor	Input	1 to 8	FLOAT32, FLOAT16
ZeroPointTensor	Input	1 to 8	INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	INT32, INT16, INT8, UINT32, UINT16, UINT8
ScaleTensor	Input	1 to 8	FLOAT32
ZeroPointTensor	Input	1 to 8	INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	INT32, INT16, INT8, UINT32, UINT16, UINT8
ScaleTensor	Input	4	FLOAT32
ZeroPointTensor	Input	4	INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	UINT8
ScaleTensor	Input	4	FLOAT32
ZeroPointTensor	Input	4	UINT8
OutputTensor	Output	4	FLOAT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

DML_ELEMENT_WISE_DIFFERENCE_SQUARE_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Subtracts each element of *BTensor* from the corresponding element of *ATensor*, multiplies the result by itself, and places the result into the corresponding element of *OutputTensor*.

$$f(a, b) = (a - b) * (a - b)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *ATensor* or *BTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_DIFFERENCE_SQUARE_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_1`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC structure (directml.h)

Article 05/09/2023

Computes the quotient of each element of *ATensor* over the corresponding element of *BTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = a / b
```

For integer divisions, the result is truncated.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_6_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_5_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ERF_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs the Gaussian error function (erf) on each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = 2 / sqrt(pi) * integrate( i = 0 to x, exp(-(i^2)) )
```

Where $\exp(x)$ is the natural exponentiation function.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ERF_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_EXP_OPERATOR_D ESC structure (directml.h)

Article02/22/2024

Applies the natural exponentiation function to each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \exp(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_EXP_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the floor for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

The floor of x is the largest integer that is less than or equal to x .

```
f(x) = floor(x)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML FEATURE LEVEL 3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML FEATURE LEVEL 1_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Computes the identity for each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = x
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

The identity operation is often used to copy a tensor.

It can also be used to transform the layout of tensors by manipulating strides (see [Using strides to express padding and memory layout](#)).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_ELEMENT_WISE_IF_OPERATOR_DESC structure (directml.h)

Article 01/21/2022

Selects elements either from *ATensor* or *BTensor*, depending on the value of the corresponding element in *ConditionTensor*. Non-zero elements of *ConditionTensor* select from *ATensor*, while zero-valued elements select from *BTensor*.

```
f(cond, a, b) = a, if cond != 0  
                b, otherwise
```

Example:

```
[[1, 0], [1, 1]] // ConditionTensor  
[[1, 2], [3, 4]] // ATensor  
[[9, 8], [7, 6]] // BTensor  
  
[[1, 8], [3, 4]] // Output
```

Syntax

C++

```
struct DML_ELEMENT_WISE_IF_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ConditionTensor;  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ConditionTensor

Type: **const DML_TENSOR_DESC***

The condition tensor to read from.

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

`ATensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the right-hand side inputs.

`BTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Remarks

Can be used to functionally build up other aggregate operators, such as LeakyRelu.

Here's an illustration in pseudo-code (not the most efficient way, but possible):

```
LeakyRelu(x) = If(Less(x, 0), Mul(x, alpha), x).
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

- *ATensor*, *BTensor*, *ConditionTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.
- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ConditionTensor	Input	1 to 8	UINT8

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ConditionTensor	Input	1 to 8	UINT8
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ConditionTensor	Input	4	UINT8
ATensor	Input	4	FLOAT16
BTensor	Input	4	FLOAT16
OutputTensor	Output	4	FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_IS_INFINITY_OPERATOR_DESC structure (directml.h)

Checks each element of *InputTensor* for IEEE-754 -inf, inf, or both, depending on the given *InfinityMode*, and places the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

```
f(x) = isinf(x) && (
    (x > 0 && InfinityMode == DML_IS_INFINITY_MODE_POSITIVE) ||
    (x < 0 && InfinityMode == DML_IS_INFINITY_MODE_NEGATIVE) ||
    InfinityMode == DML_IS_INFINITY_MODE_EITHER)
```

Syntax

C++

```
struct DML_ELEMENT_WISE_IS_INFINITY_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_IS_INFINITY_MODE InfinityMode;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

InfinityMode

Type: **DML_IS_INFINITY_MODE**

A **DML_IS_INFINITY_MODE** determining the sign of the infinity to check for.

- If **DML_IS_INFINITY_MODE_EITHER**, then 1 will be returned if the element is -inf or inf, otherwise 0.
- If **DML_IS_INFINITY_MODE_POSITIVE**, then 1 will be returned if the element is inf, otherwise 0.
- If **DML_IS_INFINITY_MODE_NEGATIVE**, then 1 will be returned if the element is -inf, otherwise 0.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount* and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	UINT8

DML_FEATURE_LEVEL_2_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT8

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Last updated on 02/03/2026

DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

For each element of the input tensor, returns 1 if the input is NaN (as defined by IEEE-754), and 0 otherwise. The result is placed into the corresponding element of the output tensor.

Syntax

C++

```
struct DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount* and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32, UINT8

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

[+] Yes

[-] No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs a logical AND on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

```
f(a, b) = (a && b)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT8
BTensor	Input	1 to 8	UINT32, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32, UINT8
BTensor	Input	4	UINT32, UINT8
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32
BTensor	Input	4	UINT32
OutputTensor	Output	4	UINT32

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs a logical *equals* on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

```
f(a, b) = (a == b)
```

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.
- *ATensor* and *BTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs a logical *greater than* on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

```
f(a, b) = (a > b)
```

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.
- *ATensor* and *BTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a logical *greater than or equal to* on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

```
f(a, b) = (a >= b)
```

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.
- *ATensor* and *BTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a logical *less than* on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

$$f(a, b) = (a < b)$$

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.
- *ATensor* and *BTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_3_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Performs a logical *less than or equal to* on each pair of corresponding elements of the input tensors, placing the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

```
f(a, b) = (a <= b)
```

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.
- *ATensor* and *BTensor* must have the same *DataType*.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

- *ATensor* and *BTensor* must have the same *DataType*.
- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)[No](#)[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs a logical NOT on each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = !x
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	UINT32, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	UINT32, UINT8
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	UINT32
OutputTensor	Output	4	UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a logical OR on each pair of corresponding elements of the input tensors, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = (a || b)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT8
BTensor	Input	1 to 8	UINT32, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32, UINT8
BTensor	Input	4	UINT32, UINT8
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32
BTensor	Input	4	UINT32
OutputTensor	Output	4	UINT32

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a logical XOR (exclusive or) on each pair of corresponding elements of the input tensors, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = (!!a) != (!!b)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	UINT32, UINT8
BTensor	Input	1 to 8	UINT32, UINT8
OutputTensor	Output	1 to 8	UINT32, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32, UINT8
BTensor	Input	4	UINT32, UINT8
OutputTensor	Output	4	UINT32, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	UINT32
BTensor	Input	4	UINT32
OutputTensor	Output	4	UINT32

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_LOG_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the base-e (natural) logarithm of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

If *x* is negative, then this function returns `indefinite`. If *x* is 0, then this function returns -INF.

$$f(x) = \ln(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_LOG_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS  *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML FEATURE LEVEL 3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML FEATURE LEVEL 1_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_MAX_OPERATOR_DESC structure (directml.h)

Article01/21/2022

Takes the greater of two corresponding elements from the input tensors, and places the result into the corresponding element of the output tensor.

```
f(a, b) = max(a, b)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_MAX_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_MEAN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Averages each pair of corresponding elements of the input tensors, placing the result into the corresponding element of *OutputTensor*.

$$f(a, b) = (a + b) / 2$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_MEAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_MIN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Takes the lesser of two corresponding elements from the input tensors, and places the result into the corresponding element of *OutputTensor*.

```
f(a, b) = min(a, b)
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_MIN_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_IS_INFINITY_OPERATOR_DESC structure (directml.h)

Checks each element of *InputTensor* for IEEE-754 -inf, inf, or both, depending on the given *InfinityMode*, and places the result (1 for true, 0 for false) into the corresponding element of *OutputTensor*.

```
f(x) = isinf(x) && (
    (x > 0 && InfinityMode == DML_IS_INFINITY_MODE_POSITIVE) ||
    (x < 0 && InfinityMode == DML_IS_INFINITY_MODE_NEGATIVE) ||
    InfinityMode == DML_IS_INFINITY_MODE_EITHER)
```

Syntax

C++

```
struct DML_ELEMENT_WISE_IS_INFINITY_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_IS_INFINITY_MODE InfinityMode;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

InfinityMode

Type: **DML_IS_INFINITY_MODE**

A **DML_IS_INFINITY_MODE** determining the sign of the infinity to check for.

- If **DML_IS_INFINITY_MODE_EITHER**, then 1 will be returned if the element is -inf or inf, otherwise 0.
- If **DML_IS_INFINITY_MODE_POSITIVE**, then 1 will be returned if the element is inf, otherwise 0.
- If **DML_IS_INFINITY_MODE_NEGATIVE**, then 1 will be returned if the element is -inf, otherwise 0.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount* and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	UINT8

DML_FEATURE_LEVEL_2_1 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT8

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Last updated on 02/03/2026

DML_ELEMENT_WISE_MODULUS_TRUNCATE_OPERATOR_DESC structure (directml.h)

Article05/09/2023

Computes the C modulus operator for each pair of corresponding elements of the input tensors, placing the result into the corresponding element of *OutputTensor*.

Because the quotient is rounded towards 0, the result will have the same sign as the dividend.

```
f(a, b) = a - (b * trunc(a / b))
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_MODULUS_TRUNCATE_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_6_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
BTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC structure (directml.h)

Article 07/20/2022

Computes the product of each pair of corresponding elements of the input tensors, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = a * b
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_1 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_4_1 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4 to 5	FLOAT32, FLOAT16
BTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_NEGATE_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Negates each element of *InputTensor*, storing the result into the corresponding element of *OutputTensor*.

$$f(x) = -x$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_NEGATE_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_5_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_POW_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Computes each element of *InputTensor* raised to the power of the corresponding element of *ExponentTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(input, exponent) = pow(input, exponent)
```

Negative bases are supported for exponents with integral values (though datatype can still be float), otherwise this operator returns NaN.

When the input tensor and exponent tensor both have integral data type, this operator guarantees exact results.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

```
C++  
  
struct DML_ELEMENT_WISE_POW_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *ExponentTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the input values.

`ExponentTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the exponent values.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

Until `DML_FEATURE_LEVEL_3_0`, *ExponentTensor* must match the type of *InputTensor*.

See [DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC](#) for a POW operator that accepts a `FLOAT` constant for the exponent.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *ExponentTensor*, *InputTensor*, and *OutputTensor* must have the same *DimensionCount* and *Sizes*.
- *InputTensor* and *OutputTensor* must have the same *DataType*.

Tensor support

`DML_FEATURE_LEVEL_3_0` and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
ExponentTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
ExponentTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

See also

[DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC structure (directml.h)

Article 08/21/2024

Performs the following linear quantization function on every element in *InputTensor* with respect to its corresponding element in *ScaleTensor* and *ZeroPointTensor*, placing the results in the corresponding element of *OutputTensor*.

```
// For uint8 output, Min = 0, Max = 255
// For int8 output, Min = -128, Max = 127
f(input, scale, zero_point) = clamp(round(input / scale) + zero_point, Min,
Max)
```

Quantizing involves converting to a lower-precision data type in order to accelerate arithmetic. It's a common way to increase performance at the cost of precision. A group of 8-bit values can be computed faster than a group of 32-bit values can.

Syntax

C++

```
struct DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *ZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the inputs.

`ScaleTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the scales.

ⓘ Note

A scale value of 0 results in undefined behavior.

If *InputTensor* is **INT32**, then *ScaleTensor* must be **FLOAT32**. Otherwise, *ScaleTensor* must have the same *DataType* as *InputTensor*.

`ZeroPointTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the desired zero point for the quantization.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *InputTensor*, *OutputTensor*, *ScaleTensor*, and *ZeroPointTensor* must have the same *DimensionCount* and *Sizes*.
- *OutputTensor* and *ZeroPointTensor* must have the same *DataType*.

Tensor support

`DML_FEATURE_LEVEL_6_2` and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32
ScaleTensor	Input	1 to 8	FLOAT32, FLOAT16
ZeroPointTensor	Optional input	1 to 8	INT8, UINT8
OutputTensor	Output	1 to 8	INT8, UINT8

DML_FEATURE_LEVEL_6_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32
ScaleTensor	Input	1 to 8	FLOAT32, FLOAT16
ZeroPointTensor	Input	1 to 8	INT8, UINT8
OutputTensor	Output	1 to 8	INT8, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, INT32
ScaleTensor	Input	1 to 8	FLOAT32
ZeroPointTensor	Input	1 to 8	INT8, UINT8
OutputTensor	Output	1 to 8	INT8, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, INT32

Tensor	Kind	Supported dimension counts	Supported data types
ScaleTensor	Input	4	FLOAT32
ZeroPointTensor	Input	4	INT8, UINT8
OutputTensor	Output	4	INT8, UINT8

DML_FEATURE_LEVEL_1_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32
ScaleTensor	Input	4	FLOAT32
ZeroPointTensor	Input	4	UINT8
OutputTensor	Output	4	UINT8

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

See also

[DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_QUANTIZED_LINEAR_ADD_OPERATOR_DESC structure (directml.h)

Article 08/22/2024

Adds every element in *ATensor* to its corresponding element in *BTensor*, placing the result into the corresponding element of *OutputTensor*. Values contained in *ATensor* and *BTensor* are dequantized using the following equation, and then added and requantized.

```
AValue = (A - AZeroPoint) * AScale
BValue = (B - BZeroPoint) * BScale

OutputValue = AValue + BValue

// For uint8 output, Min = 0, Max = 255
// For int8 output, Min = -128, Max = 127
OutputTensor = clamp(round(OutputValue / OutputScale) + OutputZeroPoint,
Min, Max)
```

Syntax

C++

```
struct DML_ELEMENT_WISE_QUANTIZED_LINEAR_ADD_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *AScaleTensor;
    const DML_TENSOR_DESC *AZeroPointTensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *BScaleTensor;
    const DML_TENSOR_DESC *BZeroPointTensor;
    const DML_TENSOR_DESC *OutputScaleTensor;
    const DML_TENSOR_DESC *OutputZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand-side inputs.

`AScaleTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the desired scale factor for *ATensor*. The expected number of elements in *AScaleTensor* is 1.

 **Note**

A scale value of 0 results in undefined behavior.

`AZeroPointTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

The tensor containing the desired zero point for *ATensor*. The expected number of elements in *AZeroPointTensor* is 1. *AZeroPointTensor* is an optional tensor that defaults to 0 if not provided.

`BTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the right-hand-side inputs.

`BScaleTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the desired scale factor for *BTensor*. The expected number of elements in *BScaleTensor* is 1.

 **Note**

A scale value of 0 results in undefined behavior.

`BZeroPointTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

The tensor containing the desired zero point for *BTensor*. The expected number of elements in *BZeroPointTensor* is 1. *BZeroPointTensor* is an optional tensor that defaults to 0 if not provided.

OutputScaleTensor

Type: `const DML_TENSOR_DESC*`

The tensor containing the desired scale factor for *OutputTensor*. This is an input tensor defining the output quantization scale factor to use while quantizing the output values. The expected number of elements in *OutputScaleTensor* is 1.

ⓘ Note

A scale value of 0 results in undefined behavior.

OutputZeroPointTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

The tensor containing the desired zero point for *OutputTensor*. This is an input tensor defining the output quantization zero point to use while quantizing the output values. The expected number of elements in *OutputZeroPointTensor* is 1. *OutputZeroPointTensor* is an optional tensor that defaults to 0 if not provided.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_4_0`.

Tensor constraints

- *AScaleTensor*, *ATensor*, *AZeroPointTensor*, *BScaleTensor*, *BTensor*, *BZeroPointTensor*, *OutputScaleTensor*, *OutputTensor*, and *OutputZeroPointTensor* must have the same *DimensionCount*.
- *BTensor* and *BZeroPointTensor* must have the same *DataType*.
- *OutputTensor* and *OutputZeroPointTensor* must have the same *DataType*.
- *ATensor* and *AZeroPointTensor* must have the same *DataType*.

Tensor support

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	INT8, UINT8
AScaleTensor	Input	1 to 8	FLOAT32
AZeroPointTensor	Optional input	1 to 8	INT8, UINT8
BTensor	Input	1 to 8	INT8, UINT8
BScaleTensor	Input	1 to 8	FLOAT32
BZeroPointTensor	Optional input	1 to 8	INT8, UINT8
OutputScaleTensor	Input	1 to 8	FLOAT32
OutputZeroPointTensor	Optional input	1 to 8	INT8, UINT8
OutputTensor	Output	1 to 8	INT8, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	directml.h

See also

- [DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_RECIP_OPERATOR_DESC structure (directml.h)

Article02/22/2024

Computes the reciprocal for each element of the input tensor, placing the result into the corresponding element of the output tensor.

$$f(x) = 1 / x$$

This operator supports in-place execution, meaning that the output tensor is permitted to alias the input tensor during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_RECIP_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor to read from for the first input tensor, x.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

ScaleBias

Type: **_Maybenull_ const DML_SCALE_BIAS***

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_ROUND_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Rounds each element of *InputTensor* to an integer value, placing the result into the corresponding element of *OutputTensor*.

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_ROUND_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_ROUNDING_MODE     RoundingMode;
};
```

Members

`InputTensor`

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

`RoundingMode`

Type: **DML_ROUNDING_MODE**

A **DML_ROUNDING_MODE** determining the direction to round towards.

- If **DML_ROUNDING_MODE_HALVES_TO_NEAREST_EVEN**: values are rounded to the nearest integer, with halfway values (for example, 0.5) being rounded toward

the nearest even integer.

- If **DML_ROUNDING_MODE_TOWARD_ZERO**: values are rounded toward zero. This effectively truncates the fractional part.
- If **DML_ROUNDING_MODE_TOWARD_INFINITY**: values are rounded to the nearest integer, with halfway values (for example, 0.5) being rounded away from zero (toward positive or negative infinity, depending on the sign of the value).

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML FEATURE LEVEL 3_0 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML FEATURE LEVEL 2_1 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_SIGN_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Returns a value representing the sign of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(x) = -1, if x < 0  
      1, if x > 0  
      0, otherwise // This includes negative zero.
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_SIGN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Remarks

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

[\[+\] Yes](#)

[\[+\] No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_SIN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the trigonometric sine of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \sin(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_SIN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_SINH_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the hyperbolic sine of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \sinh(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_SINH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_SQRT_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the square root of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \sqrt{x}$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_SQRT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Subtracts each element of *BTensor* from the corresponding element of *ATensor*, placing the result into the corresponding element of *OutputTensor*.

```
f(a, b) = a - b
```

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias one of the input tensors during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the left-hand side inputs.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the right-hand side inputs.

OutputTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

ATensor, *BTensor*, and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_5_1 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_4_1 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
BTensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	FLOAT32, FLOAT16
BTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_TAN_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the trigonometric tangent of each element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \tan(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_TAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_TANH_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the hyperbolic tangent of element of *InputTensor*, placing the result into the corresponding element of *OutputTensor*.

$$f(x) = \tanh(x)$$

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_TANH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

Remarks

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

`DML_FEATURE_LEVEL_3_0` and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

`DML_FEATURE_LEVEL_2_0` and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Replaces all elements of *InputTensor* below the given threshold, *Min*, with *Min*. Results are placed into the corresponding element of *OutputTensor*.

$$f(x) = \max(x, \text{Min})$$

Where $\max(a,b)$ returns the larger of the two values a,b .

This operator supports in-place execution, meaning that *OutputTensor* is permitted to alias *InputTensor* during binding.

Syntax

C++

```
struct DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
    FLOAT Min;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to.

`ScaleBias`

Type: `_Maybenull_ const DML_SCALE_BIAS*`

An optional scale and bias to apply to the input. If present, this has the effect of applying the function `g(x) = x * scale + bias` to each *input* element prior to computing this operator.

`Min`

Type: `FLOAT`

The minimum value, below which the operator replaces the value with *Min*.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[\[+\] Yes](#)

[\[+\] No](#)

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT structure (directml.h)

Article 02/22/2024

Provides detail about whether a DirectML device supports a particular data type within tensors. See [IDMLDevice::CheckFeatureSupport](#). The query type is [DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT](#), and the support data type is [DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT](#).

Syntax

C++

```
struct DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT {
    BOOL IsSupported;
};
```

Members

`IsSupported`

Type: [BOOL](#)

TRUE if the tensor data type is supported within tensors by the DirectML device. Otherwise, FALSE.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

See also

[IDMLDevice::CheckFeatureSupport](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT structure (directml.h)

Article 02/22/2024

Used to query a DirectML device for its support for a particular data type within tensors. See [IDMLDevice::CheckFeatureSupport](#). The query type is [DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT](#), and the support data type is [DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT](#).

Syntax

C++

```
struct DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT {
    DML_TENSOR_DATA_TYPE DataType;
};
```

Members

`DataType`

Type: [DML_TENSOR_DATA_TYPE](#)

The data type about which you're querying for support.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

DML_FILL_VALUE_CONSTANT_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Fills a tensor with the given constant *Value*. This operator performs the following pseudocode.

```
for each coordinate in OutputTensor
    OutputTensor[coordinate] = Value
endfor
```

Syntax

C++

```
struct DML_FILL_VALUE_CONSTANT_OPERATOR_DESC {
    const DML_TENSOR_DESC *OutputTensor;
    DML_TENSOR_DATA_TYPE ValueDataType;
    DML_SCALAR_UNION    Value;
};
```

Members

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the results to. This tensor may have any size.

ValueDataType

Type: **DML_TENSOR_DATA_TYPE**

The data type of the *Value* field, which must match *OutputTensor.DataType*.

Value

Type: **DML_SCALAR_UNION**

A constant value to fill the output, with *ValueDataType* determining how to interpret the field.

Examples

```
Value = 13.0

OutputTensor: (Sizes:{1,1,2,4}, DataType:FLOAT32)
[[[[13.0f, 13.0f, 13.0f, 13.0f],
    [13.0f, 13.0f, 13.0f, 13.0f]]]]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_4_0 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)[No](#)[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_FILL_VALUE_SEQUENCE_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Fills a tensor with a sequence. This operator performs the following pseudocode.

```
for each elementIndex in OutputTensor
    OutputTensor[elementIndex] = Value + elementIndex * Delta
endfor
```

Syntax

C++

```
struct DML_FILL_VALUE_SEQUENCE_OPERATOR_DESC {
    const DML_TENSOR_DESC *OutputTensor;
    DML_TENSOR_DATA_TYPE ValueDataType;
    DML_SCALAR_UNION    ValueStart;
    DML_SCALAR_UNION    ValueDelta;
};
```

Members

`OutputTensor`

Type: [const DML_TENSOR_DESC*](#)

The tensor to write the results to. This tensor may have any size.

`ValueDataType`

Type: [DML_TENSOR_DATA_TYPE](#)

The data type of `Value` field, which must match `OutputTensor.DataType`.

`ValueStart`

Type: [DML_SCALAR_UNION](#)

The initial value to fill the first element in the output, with *ValueDataType* determining how to interpret the field.

`ValueDelta`

Type: [DML_SCALAR_UNION](#)

A step to add to the value for each element written, with *ValueDataType* determining how to interpret the field.

Examples

Example 1. 1D ascending step

```
ValueStart = 3
ValueDelta = 2
ValueDataType = DML_TENSOR_DATA_TYPE_FLOAT32

OutputTensor: (Sizes:{1,1,1,3}, DataType:FLOAT32)
[[[[3, 5, 7]]]]
```

Example 2. 2D ascending step

```
ValueStart = 10
ValueDelta = -2
ValueDataType = DML_TENSOR_DATA_TYPE_UINT8

OutputTensor: (Sizes:{1,1,2,2}, DataType:UINT8)
[[[[10, 8],
   [6, 4]]]]
```

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_1](#).

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_4_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_FOLD_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Combines an array of patches formed from a sliding window into a large containing tensor.

Consider a batched input tensor containing sliding local blocks, e.g., patches of images, of shape $(N, C \times \prod(\text{WindowSizes}), \text{BlockCount})$, where N is batch dimension, $C \times \prod(\text{WindowSizes})$ is the number of values within a window (a window has $\prod(\text{WindowSizes})$ spatial locations each containing a C -channeled vector), and BlockCount is the total number of blocks. This operation combines these local blocks into the large output tensor of shape $(N, C, \text{OutputSize}[0], \text{OutputSize}[1], \dots)$ by summing the overlapping values. The arguments must satisfy:

$$\text{BlocksPerDimension}[d] = ((\text{SpatialSize}[d] + \text{StartPadding}[d] + \text{EndPadding}[d] - \text{Dilations}[d] * (\text{WindowSizes}[d] - 1) - 1) / \text{stride}[d]) + 1$$

$$\text{BlockCount} = \prod_d \text{BlocksPerDimension}[d]$$

Where:

- $0 \leq d < \text{DimensionCount}$

The `OutputSize` (the `OutputTensor`'s size) describes the spatial shape of the large containing tensor of the sliding local blocks.

The `StartPadding`, `EndPadding`, `Strides`, and `Dilations` arguments specify how the sliding blocks are retrieved.

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) version 1.15.1 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_FOLD_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* OutputTensor;
    UINT DimensionCount;
    _Field_size_(DimensionCount) const UINT* WindowSizes;
    _Field_size_(DimensionCount) const UINT* Strides;
    _Field_size_(DimensionCount) const UINT* Dilations;
    _Field_size_(DimensionCount) const UINT* StartPadding;
    _Field_size_(DimensionCount) const UINT* EndPadding;
};
```

Members

`InputTensor`

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

`DimensionCount`

Type: **UINT**

The spatial dimensions of `InputTensor`. `DimensionCount` must be ≤ 6 .

`WindowSizes`

Type: `_Field_size_(DimensionCount) const UINT*`

The size of the sliding window. Size of the extracted patch.

`Strides`

Type: `_Field_size_(DimensionCount) const UINT*`

The stride of the sliding window (with dimensions `WindowSizes`) in the input spatial dimensions. They are separate from the tensor strides included in **DML_TENSOR_DESC**. Step size of the extracted patches.

`Dilations`

Type: `_Field_size_(DimensionCount) const UINT*`

The dilations of the sliding window (with dimensions *WindowSizes*) in the input spatial dimensions, by scaling the space between the kernel points. Dilations of the extracted patch.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the amount of implicit zero-padding to be applied to the beginning of each spatial dimension of *InputTensor*. Start padding of the source tensor.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the amount of implicit zero-padding to be applied to the end of each spatial dimension of *InputTensor*. End padding of the source tensor.

Examples

Example 1

A 1-channel fold.

```
InputTensor: (Sizes:{1, 9, 4}, DataType:FLOAT32)
[[[ 0.,  1.,  2.,  3.],
 [ 4.,  5.,  6.,  7.],
 [ 8.,  9., 10., 11.],
 [12., 13., 14., 15.],
 [16., 17., 18., 19.],
 [20., 21., 22., 23.],
 [24., 25., 26., 27.],
 [28., 29., 30., 31.],
 [32., 33., 34., 35.]]]
DimensionCount: 2
WindowSizes: {3, 3}
Strides: {1, 1}
Dilations: {1, 1}
StartPadding: {0, 0}
EndPadding: {0, 0}
OutputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[[ 0.,  5., 13.,  9.],
 [ 14., 38., 54., 32.],
```

```
[ 38.,  86., 102.,  56.],  
[ 26.,  57.,  65.,  35.]]]
```

Example 2

A 1-channel, padded fold.

```
InputTensor: (Sizes:{1, 9, 8}, DataType:FLOAT32)  
[[[ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.],  
  [ 8.,  9., 10., 11., 12., 13., 14., 15.],  
  [16., 17., 18., 19., 20., 21., 22., 23.],  
  [24., 25., 26., 27., 28., 29., 30., 31.],  
  [32., 33., 34., 35., 36., 37., 38., 39.],  
  [40., 41., 42., 43., 44., 45., 46., 47.],  
  [48., 49., 50., 51., 52., 53., 54., 55.],  
  [56., 57., 58., 59., 60., 61., 62., 63.],  
  [64., 65., 66., 67., 68., 69., 70., 71.]]]  
DimensionCount: 2  
WindowSizes: {3, 3}  
Strides: {1, 1}  
Dilations: {1, 1}  
StartPadding: {1, 0}  
EndPadding: {1, 0}  
OutputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)  
[[[[ 26., 70., 102., 60.],  
   [ 78., 183., 231., 129.],  
   [ 84., 195., 243., 135.],  
   [ 82., 182., 214., 116.]]]]
```

Example 3

A 2-channel, padded fold.

```
InputTensor: (Sizes:{1, 18, 8}, DataType:FLOAT32)  
[[[ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.],  
  [ 8.,  9., 10., 11., 12., 13., 14., 15.],  
  [16., 17., 18., 19., 20., 21., 22., 23.],  
  [24., 25., 26., 27., 28., 29., 30., 31.],  
  [32., 33., 34., 35., 36., 37., 38., 39.],  
  [40., 41., 42., 43., 44., 45., 46., 47.],  
  [48., 49., 50., 51., 52., 53., 54., 55.],  
  [56., 57., 58., 59., 60., 61., 62., 63.],  
  [64., 65., 66., 67., 68., 69., 70., 71.],  
  [72., 73., 74., 75., 76., 77., 78., 79.],  
  [80., 81., 82., 83., 84., 85., 86., 87.],
```

```

[ 88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.],
[ 96.,  97.,  98.,  99., 100., 101., 102., 103.],
[104., 105., 106., 107., 108., 109., 110., 111.],
[112., 113., 114., 115., 116., 117., 118., 119.],
[120., 121., 122., 123., 124., 125., 126., 127.],
[128., 129., 130., 131., 132., 133., 134., 135.],
[136., 137., 138., 139., 140., 141., 142., 143.]]]
DimensionCount: 2
WindowSizes: {3, 3}
Strides: {1, 1}
Dilations: {1, 1}
StartPadding: {1, 0}
EndPadding: {1, 0}
OutputTensor: (Sizes:{1, 2, 4, 4}, DataType:FLOAT32)
[[[[ 26.,  70., 102.,  60.],
   [ 78., 183., 231., 129.],
   [ 84., 195., 243., 135.],
   [ 82., 182., 214., 116.]],

  [[170., 358., 390., 204.],
   [294., 615., 663., 345.],
   [300., 627., 675., 351.],
   [226., 470., 502., 260.]]]]

```

Availability

This operator was introduced in **DML_FEATURE_LEVEL_6_4**.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount*.

Tensor support

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	3 to 8	
OutputTensor	Output	3 to 8	

Requirements

Expand table

Header	directml.h
---------------	------------

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GATHER_ELEMENTS_OPERATOR_D ESC structure (directml.h)

Article 10/20/2021

Gathers elements from the input tensor along the given axis using the indices tensor to remap into the input. This operator performs the following pseudocode, with the exact behavior dependent on the axis, input dimension count, and indices dimension count.

```
output[i, j, k, ...] = input[index[i, j, k, ...], j, k, ...] // if axis == 0
output[i, j, k, ...] = input[i, index[i, j, k, ...], k, ...] // if axis == 1
output[i, j, k, ...] = input[i, j, index[i, j, k, ...], ...] // if axis == 2
...
...
```

Syntax

C++

```
struct DML_GATHER_ELEMENTS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor to read from.

IndicesTensor

Type: **const DML_TENSOR_DESC***

The indices into the input tensor along the active axis. The *Sizes* must match *InputTensor.Sizes* for every dimension except *Axis*.

Starting with `DML_FEATURE_LEVEL_3_0`, this operator supports negative index values when using a signed integral type with this tensor. Negative indices are interpreted as being relative to the end of the axis dimension. For example, an index of -1 refers to the last element along that dimension.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. The `Sizes` must match `IndicesTensor.Sizes`, and `DataType` must match `InputTensor.DataType`.

Axis

Type: `UINT`

The axis dimension of `InputTensor` to gather along, ranging `[0, *InputTensor.DimensionCount*)`.

Examples

```
Axis = 0

InputTensor: (Sizes:{3,3}, DataType:FLOAT32)
    [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

IndicesTensor: (Sizes:{2,3}, DataType:UINT32)
    [[1, 2, 0],
     [2, 0, 0]]

// output[y, x] = input[indices[y, x], x]
OutputTensor: (Sizes:{2,3}, DataType:UINT32)
    [[4, 8, 3], // select elements vertically from data
     [7, 2, 3]]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- `IndicesTensor`, `InputTensor`, and `OutputTensor` must have the same `DimensionCount`.
- `InputTensor` and `OutputTensor` must have the same `DataType`.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GATHER_ND_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Gathers elements from the input tensor, using the indices tensor to remap indices to entire subblocks of the input. This operator performs the following pseudocode, where "... " represents a series of coordinates, with the exact behavior dependent on the input and indices dimension count.

```
output[...] = input[indices[...]]
```

Syntax

C++

```
struct DML_GATHER_ND_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT InputDimensionCount;
    UINT IndicesDimensionCount;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to read from.

`IndicesTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the indices. The *DimensionCount* of this tensor must match *InputTensor.DimensionCount*. The last dimension of the *IndicesTensor* is actually the number of coordinates per index tuple, and it cannot exceed *InputTensor.DimensionCount*. For example, an indices tensor of *Sizes* {1,4,5,2} with

IndicesDimensionCount = 3 means a 4x5 array of 2-coordinate tuples that index into *InputTensor*.

Starting with `DML_FEATURE_LEVEL_3_0`, this operator supports negative index values when using a signed integral type with this tensor. Negative indices are interpreted as being relative to the end of the respective dimension. For example, an index of -1 refers to the last element along that dimension.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. The *DimensionCount* and *DataType* of this tensor must match *InputTensor.DimensionCount*. The expected *OutputTensor.Sizes* are the concatenation of the *IndicesTensor.Sizes* leading segments and *InputTensor.Sizes* trailing segment to yield:

```
indexTupleSize = IndicesTensor.Sizes[IndicesTensor.DimensionCount - 1]
OutputTensor.Sizes = {
    1...,
    IndicesTensor.Sizes[(IndicesTensor.DimensionCount -
IndicesDimensionCount) .. (IndicesTensor.DimensionCount - 1)],
    InputTensor.Sizes[(InputTensor.DimensionCount - indexTupleSize) ..
InputTensor.DimensionCount]
}
```

The output dimensions are right-aligned, with leading 1 values prepended if needed to satisfy up to *OutputTensor.DimensionCount*.

Here's an example.

```
InputTensor.Sizes = {3,4,5,6,7}
InputDimensionCount = 5
IndicesTensor.Sizes = {1,1, 1,2,3}
IndicesDimensionCount = 3 // can be thought of as a {1,2} array of 3-
coordinate tuples

// The {1,2} comes from the indices tensor (ignoring last dimension which is
the tuple size),
// and the {6,7} comes from input tensor, ignoring the first 3 dimensions
// since the index tuples are 3 elements (from the indices tensor last
dimension).
OutputTensor.Sizes = {1, 1,2,6,7}
```

```
InputDimensionCount
```

Type: **UINT**

The number of actual input dimensions within the *InputTensor* after ignoring any irrelevant leading ones, ranging `[1, *InputTensor.DimensionCount*]`. For example, given *InputTensor.Sizes* = `{1,1,4,6}` and *InputDimensionCount* = 3, the actual meaningful indices are `{1,4,6}`.

```
IndicesDimensionCount
```

Type: **UINT**

The number of actual index dimensions within the *IndicesTensor* after ignoring any irrelevant leading ones, ranging `[1, IndicesTensor.DimensionCount]`. For example, given *IndicesTensor.Sizes* = `{1,1,4,6}`, and *IndicesDimensionCount* = 3, the actual meaningful indices are `{1,4,6}`.

Examples

Example 1. 1D remapping

```
InputDimensionCount: 2
IndicesDimensionCount: 2

InputTensor: (Sizes:{2,2}, DataType:FLOAT32)
    [[0,1],[2,3]]

IndicesTensor: (Sizes:{2,1}, DataType:UINT32)
    [[1],[0]]

// output[y, x] = input[indices[y], x]
OutputTensor: (Sizes:{2,2}, DataType:FLOAT32)
    [[2,3],[0,1]]
```

Example 2. 2D remapping

```
InputDimensionCount: 3
IndicesDimensionCount: 2

InputTensor: (Sizes:{1, 2,2,2}, DataType:FLOAT32)
```

```

[ [[[0,1],[2,3]],[[4,5],[6,7]]] ]

IndicesTensor: (Sizes:{1,1, 2,2}, DataType:UINT32)
[[ [[0,1],[1,0]] ]]

// output[y, x] = input[indices[y, 0], indices[y, 1], x]
OutputTensor: (Sizes:{1,1, 2,2}, DataType:FLOAT32)
[[ [[2,3],[4,5]] ]]

```

Remarks

A newer version of this operator, `DML_OPERATOR_GATHER_ND1`, was introduced in `DML_FEATURE_LEVEL_3_0`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- *IndicesTensor*, *InputTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *InputTensor* and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)[No](#)

DML_GATHER_ND1_OPERATOR_DESC structure (directml.h)

Article07/27/2022

Gathers elements from the input tensor, using the indices tensor to remap indices to entire subblocks of the input. This operator performs the following pseudocode, where "... " represents a series of coordinates, with the exact behavior dependent on the batch, input, and indices dimension count.

```
output[batch, ...] = input[batch, indices[batch, ...], ...]
```

Syntax

C++

```
struct DML_GATHER_ND1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT InputDimensionCount;
    UINT IndicesDimensionCount;
    UINT BatchDimensionCount;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to read from.

`IndicesTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the indices. The *DimensionCount* of this tensor must match *InputTensor.DimensionCount*. The last dimension of the *IndicesTensor* is actually the number of coordinates per index tuple, and it cannot exceed

InputTensor.DimensionCount. For example, an indices tensor of *Sizes* {1,4,5,2} with *IndicesDimensionCount* = 3 means a 4x5 array of 2-coordinate tuples that index into *InputTensor*.

Starting with `DML_FEATURE_LEVEL_3_0`, this operator supports negative index values when using a signed integral type with this tensor. Negative indices are interpreted as being relative to the end of the respective dimension. For example, an index of -1 refers to the last element along that dimension.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. The *DimensionCount* and *DataType* of this tensor must match *InputTensor.DimensionCount*. The expected *OutputTensor.Sizes* are the concatenation of the *IndicesTensor.Sizes* leading segments and *InputTensor.Sizes* trailing segment, which yields the following.

```
indexTupleSize = IndicesTensor.Sizes[IndicesTensor.DimensionCount - 1]
OutputTensor.Sizes = {
    1...,
    IndicesTensor.Sizes[(IndicesTensor.DimensionCount -
    IndicesDimensionCount) .. (IndicesTensor.DimensionCount - 1)],
    InputTensor.Sizes[(InputTensor.DimensionCount - indexTupleSize) ..
    InputTensor.DimensionCount]
}
```

The dimensions are right-aligned, with leading 1 values prepended if needed to satisfy *OutputTensor.DimensionCount*.

Here's an example.

```
InputTensor.Sizes = {3,4,5,6,7}
InputDimensionCount = 5
IndicesTensor.Sizes = {1,1, 1,2,3}
IndicesDimensionCount = 3 // can be thought of as a {1,2} array of 3-
coordinate tuples

// The {1,2} comes from the indices tensor (ignoring last dimension which is
// the tuple size),
// and the {6,7} comes from input tensor, ignoring the first 3 dimensions
// since the index tuples are 3 elements (from the indices tensor last
```

```
dimension).  
OutputTensor.Sizes = {1, 1, 2, 6, 7}
```

`InputDimensionCount`

Type: [UINT](#)

The number of actual input dimensions within the *InputTensor* after ignoring any irrelevant leading ones, ranging [1, `*InputTensor.DimensionCount`]. For example, given *InputTensor.Sizes* = {1,1,4,6} and *InputDimensionCount* = 3, the actual meaningful indices are {1,4,6}.

`IndicesDimensionCount`

Type: [UINT](#)

The number of actual index dimensions within the *IndicesTensor* after ignoring any irrelevant leading ones, ranging [1, *IndicesTensor.DimensionCount*]. For example, given *IndicesTensor.Sizes* = {1,1,4,6}, and *IndicesDimensionCount* = 3, the actual meaningful indices are {1,4,6}.

`BatchDimensionCount`

Type: [UINT](#)

The number of dimensions within each tensor (*InputTensor*, *IndicesTensor*, *OutputTensor*) that are considered independent batches, ranging within both [0, *InputTensor.DimensionCount*] and [0, *IndicesTensor.DimensionCount*). The batch count can be 0, implying a single batch. For example, given *IndicesTensor.Sizes* = {1,3,4,5,6,7}, and *IndicesDimensionCount* = 5 and *BatchDimensionCount* = 2, there are batches {3,4} and meaningful indices {5,6,7}.

Remarks

[DML_GATHER_ND1_OPERATOR_DESC](#) adds *BatchDimensionCount*, and is equivalent to [DML_GATHER_ND_OPERATOR_DESC](#) when *BatchDimensionCount* = 0.

Examples

Example 1. 1D remapping

```

InputDimensionCount: 2
IndicesDimensionCount: 2
BatchDimensionCount: 0

InputTensor: (Sizes:{2,2}, DataType:FLOAT32)
[[0,1],[2,3]]

IndicesTensor: (Sizes:{2,1}, DataType:UINT32)
[[1],[0]]

// output[y, x] = input[indices[y], x]
OutputTensor: (Sizes:{2,2}, DataType:FLOAT32)
[[2,3],[0,1]]

```

Example 2. 2D remapping with batch count

```

InputDimensionCount: 3
IndicesDimensionCount: 3
BatchDimensionCount: 1

// 3 batches.
InputTensor: (Sizes:{1, 3,2,2}, DataType:FLOAT32)
[
    [[[0,1],[2,3]], // batch 0
     [[4,5],[6,7]], // batch 1
     [[8,9],[10,11]] // batch 2
]

// A 3x2 array of 2D tuples indexing into InputTensor.
// e.g. a tuple of <1,0> in batch 1 corresponds to input value 6.
IndicesTensor: (Sizes:{1, 3,2,2}, DataType:UINT32)
[
    [[[0,0],[1,1]],
     [[1,1],[0,0]],
     [[0,1],[1,0]]]
]

// output[batch, x] = input[batch, indices[batch, x, 0], indices[batch, x, 1]]
OutputTensor: (Sizes:{1,1, 3,2}, DataType:FLOAT32)
[
    [[0,3],
     [7,4],
     [9,10]]
]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

- *IndicesTensor*, *InputTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *InputTensor* and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GATHER_OPERATOR_DESC structure (directml.h)

Article01/26/2024

Gathers elements from the input tensor along **Axis**, using **IndicesTensor** to remap indices. This operator performs the following pseudocode, where "..." represents a series of coordinates, with the exact behavior determined by the axis and indices dimension count:

```
output[...] = input[..., indices[...], ...]
```

Syntax

C++

```
struct DML_GATHER_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
    UINT                 IndexDimensions;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor to read from.

IndicesTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the indices. The *DimensionCount* of this tensor must match *InputTensor.DimensionCount*.

Starting with `DML_FEATURE_LEVEL_3_0`, this operator supports negative index values when using a signed integral type with this tensor. Negative indices are interpreted as being relative to the end of the axis dimension. For example, an index of -1 refers to the last element along that dimension.

Invalid indices will yield incorrect outputs, but no failure will occur, and all reads will be clamped safely within the input tensor's memory.

OutputTensor

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. The *DimensionCount* and *DataType* of this tensor must match *InputTensor.DimensionCount*. The expected *OutputTensor.Sizes* are the concatenation of the *InputTensor.Sizes* leading and trailing segments split at the current *Axis* with the *IndicesTensor.Sizes* inserted between.

```
OutputTensor.Sizes = {  
    InputTensor.Sizes[0..Axis],  
    IndicesTensor.Sizes[(IndicesTensor.DimensionCount - IndexDimensions) ..  
IndicesTensor.DimensionCount],  
    InputTensor.Sizes[(Axis+1) .. InputTensor.DimensionCount]  
}
```

The dimensions are right-aligned such that any leading 1 values from the input sizes are cropped which would otherwise overflow the output *DimensionCount*.

The number of relevant dimensions in this tensor depends on *IndexDimensions* and the *original rank* of *InputTensor*. The original rank is the number of dimensions prior to any padding with leading ones. The number of relevant dimensions in the output can be computed by the *original rank* of *InputTensor* + *IndexDimensions* - 1. This value must be less than or equal to the *DimensionCount* of *OutputTensor*.

Axis

Type: `UINT`

The axis dimension of *InputTensor* to gather on, ranging `[0, *InputTensor.DimensionCount*)`.

IndexDimensions

Type: `UINT`

The number of actual index dimensions within the `IndicesTensor` after ignoring any irrelevant leading ones, ranging [0, `IndicesTensor.DimensionCount`). For example, given `IndicesTensor.Sizes` = { 1, 1, 4, 6 } and `IndexDimensions` = 3, the actual meaningful indices are { 1, 4, 6 }.

Examples

Example 1. 1D remapping

```
Axis: 0
IndexDimensions: 1

InputTensor: (Sizes:{4}, DataType:FLOAT32)
[11,12,13,14]

IndicesTensor: (Sizes:{5}, DataType:UINT32)
[3,1,3,0,2]

// output[x] = input[indices[x]]
OutputTensor: (Sizes:{5}, DataType:FLOAT32)
[14,12,14,11,13]
```

Example 2. 2D output, 1D indices, Axis 0, concatenate rows

```
Axis: 0
IndexDimensions: 1

InputTensor: (Sizes:{3,2}, DataType:FLOAT32)
[[1,2], // row 0
 [3,4], // row 1
 [5,6]] // row 2

IndicesTensor: (Sizes:{1, 4}, DataType:UINT32)
[[0,
 1,
 1,
 2]] 

// output[y, x] = input[indices[y], x]
OutputTensor: (Sizes:{4,2}, DataType:FLOAT32)
[[1,2], // input row 0
```

```
[3,4], // input row 1  
[3,4], // input row 1  
[5,6]] // input row 2
```

Example 3. 2D, Axis 1, swap columns

```
Axis: 1  
IndexDimensions: 2  
  
InputTensor: (Sizes:{3,2}, DataType:FLOAT32)  
[[1,2],  
 [3,4],  
 [5,6]]  
  
IndicesTensor: (Sizes:{1, 2}, DataType:UINT32)  
[[1,0]]  
  
// output[y, x] = input[y, indices[x]]  
OutputTensor: (Sizes:{3,2}, DataType:FLOAT32)  
[[2,1],  
 [4,3],  
 [6,5]]
```

Example 4. 2D, Axis 1, nested indices

```
Axis: 2  
IndexDimensions: 2  
  
InputTensor: (Sizes:{1, 3,3}, DataType:FLOAT32)  
[ [[1,2,3],  
  [4,5,6],  
  [7,8,9]] ]  
  
IndicesTensor: (Sizes:{1, 1,2}, DataType:UINT32)  
[ [[0,2]] ]  
  
// output[z, y, x] = input[z, indices[y, x]]  
OutputTensor: (Sizes:{3,1,2}, DataType:FLOAT32)  
[[[1,3]],  
 [[4,6]],  
 [[7,9]]]
```

Example 5. 2D, Axis 0, nested indices

```

Axis: 1
IndexDimensions: 2

InputTensor: (Sizes:{1, 3,2}, DataType:FLOAT32)
[ [[1,2],
  [3,4],
  [5,6]] ]

IndicesTensor: (Sizes:{1, 2,2}, DataType:UINT32)
[ [[0,1],
  [1,2]] ]

// output[z, y, x] = input[indices[z, y], x]
OutputTensor: (Sizes:{2,2,2}, DataType:FLOAT32)
[[[1,2], [3,4]],
 [[3,4], [5,6]]]

```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- `IndicesTensor`, `InputTensor`, and `OutputTensor` must have the same *DimensionCount*.
- `InputTensor` and `OutputTensor` must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32,

Tensor	Kind	Supported dimension counts	Supported data types
			INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
IndicesTensor	Input	4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

 Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_GEMM_OPERATOR_DESC structure (directml.h)

Article 12/02/2022

Performs a general matrix multiplication function of the form `Output = FusedActivation(Alpha * TransA(A) x TransB(B) + Beta * C)`, where `x` denotes matrix multiplication, and `*` denotes multiplication with a scalar.

This operator requires 4D tensors with layout `{ BatchCount, ChannelCount, Height, Width }`, and it will perform `BatchCount * ChannelCount` number of independent matrix multiplications.

For example, if `ATensor` has `Sizes` of `{ BatchCount, ChannelCount, M, K }`, and `BTensor` has `Sizes` of `{ BatchCount, ChannelCount, K, N }`, and `OutputTensor` has `Sizes` of `{ BatchCount, ChannelCount, M, N }`, then this operator performs `BatchCount * ChannelCount` independent matrix multiplications of dimensions $\{M,K\} \times \{K,N\} = \{M,N\}$.

Syntax

C++

```
struct DML_GEMM_OPERATOR_DESC {
    const DML_TENSOR_DESC    *ATensor;
    const DML_TENSOR_DESC    *BTensor;
    const DML_TENSOR_DESC    *CTensor;
    const DML_TENSOR_DESC    *OutputTensor;
    DML_MATRIX_TRANSFORM     TransA;
    DML_MATRIX_TRANSFORM     TransB;
    FLOAT                   Alpha;
    FLOAT                   Beta;
    const DML_OPERATOR_DESC *FusedActivation;
};
```

Members

ATensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the A matrix. This tensor's `Sizes` should be `{ BatchCount, ChannelCount, M, K }` if `TransA` is `DML_MATRIX_TRANSFORM_NONE`, or `{ BatchCount,`

`ChannelCount, K, M }` if `TransA` is `DML_MATRIX_TRANSFORM_TRANSPOSE`.

`BTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the B matrix. This tensor's `Sizes` should be `{ BatchCount, ChannelCount, K, N }` if `TransB` is `DML_MATRIX_TRANSFORM_NONE`, or `{ BatchCount, ChannelCount, N, K }` if `TransB` is `DML_MATRIX_TRANSFORM_TRANSPOSE`.

`CTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

A tensor containing the C matrix, or `nullptr`. Values default to 0 when not provided. If provided, this tensor's `Sizes` should be `{ BatchCount, ChannelCount, M, N }`.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. This tensor's `Sizes` are `{ BatchCount, ChannelCount, M, N }`.

`TransA`

Type: `DML_MATRIX_TRANSFORM`

The transform to be applied to `ATensor`; either a transpose, or no transform.

`TransB`

Type: `DML_MATRIX_TRANSFORM`

The transform to be applied to `BTensor`; either a transpose, or no transform.

`Alpha`

Type: `FLOAT`

The value of the scalar multiplier for the product of inputs `ATensor` and `BTensor`.

`Beta`

Type: `FLOAT`

The value of the scalar multiplier for the optional input *CTensor*. If *CTensor* is not provided, then this value is ignored.

FusedActivation

Type: `_Maybenull_ const DML_OPERATOR_DESC*`

An optional fused activation layer to apply after the GEMM. For more info, see [Using fused operators for improved performance](#).

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *ATensor*, *BTensor*, *CTensor*, and *OutputTensor* must have the same *DataType* and *DimensionCount*.
- *CTensor* and *OutputTensor* must have the same *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_0 and above

[] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
ATensor	Input	{ [BatchCount], [ChannelCount], M, K }	2 to 4	FLOAT32, FLOAT16
BTensor	Input	{ [BatchCount], [ChannelCount], K, N }	2 to 4	FLOAT32, FLOAT16
CTensor	Optional input	{ [BatchCount], [ChannelCount], M, N }	2 to 4	FLOAT32, FLOAT16
OutputTensor	Output	{ [BatchCount], [ChannelCount], M, N }	2 to 4	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
ATensor	Input	{ BatchCount, ChannelCount, M, K }	4	FLOAT32, FLOAT16
BTensor	Input	{ BatchCount, ChannelCount, K, N }	4	FLOAT32, FLOAT16
CTensor	Optional input	{ BatchCount, ChannelCount, M, N }	4	FLOAT32, FLOAT16
OutputTensor	Output	{ BatchCount, ChannelCount, M, N }	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

- [Using fused operators for improved performance](#)

Feedback

Was this page helpful?

[Yes](#)[No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GRAPH_DESC structure (directml.h)

Article 10/20/2021

Describes a graph of DirectML operators used to compile a combined, optimized operator. See [IDMLDevice1::CompileGraph](#).

Syntax

C++

```
struct DML_GRAPH_DESC {
    UINT             InputCount;
    UINT             OutputCount;
    UINT             NodeCount;
    const DML_GRAPH_NODE_DESC *Nodes;
    UINT             InputEdgeCount;
    const DML_GRAPH_EDGE_DESC *InputEdges;
    UINT             OutputEdgeCount;
    const DML_GRAPH_EDGE_DESC *OutputEdges;
    UINT             IntermediateEdgeCount;
    const DML_GRAPH_EDGE_DESC *IntermediateEdges;
};
```

Members

`InputCount`

Type: [UINT](#)

The number of inputs of the overall graph. Each graph input may be connected to a variable number of internal nodes, therefore this may be different from `InputEdgeCount`.

`OutputCount`

Type: [UINT](#)

The number of outputs of the overall graph. Each graph output may be connected to a variable number of internal nodes, therefore this may be different from `OutputEdgeCount`.

`NodeCount`

Type: **UINT**

The number of internal nodes in the graph.

Nodes

Type: `_Field_size_(NodeCount) const DML_GRAPH_NODE_DESC*`

The internal nodes in the graph.

InputEdgeCount

Type: **UINT**

The number of connections between graph inputs and inputs of internal nodes in the graph.

InputEdges

Type: `_Field_size_(InputEdgeCount) const DML_GRAPH_EDGE_DESC*`

An array of connections between graph inputs and inputs of internal nodes in the graph. The *Type* field within each element should be set to [DML_GRAPH_EDGE_TYPE_INPUT](#).

OutputEdgeCount

Type: **UINT**

The number of connections between graph outputs and outputs of internal nodes in the graph.

OutputEdges

Type: `_Field_size_(OutputEdgeCount) const DML_GRAPH_EDGE_DESC*`

An array of connections between graph outputs and outputs of internal nodes in the graph. The *Type* field within each element should be set to [DML_GRAPH_EDGE_TYPE_OUTPUT](#).

IntermediateEdgeCount

Type: **UINT**

The number of internal connections between nodes in the graph.

IntermediateEdges

Type: `_Field_size_(IntermediateEdgeCount) const DML_GRAPH_EDGE_DESC*`

An array of connections between inputs and outputs of internal nodes in the graph. The Type field within each element should be set to [DML_GRAPH_EDGE_TYPE_INTERMEDIATE](#)

Remarks

The graph described by this structure must be a directed acyclic graph. You must define a connection for the input and output of each supplied node, except for inputs and outputs that are optional for the associated operator.

Nodes may use operators that were created using the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag for certain inputs. Any operator inputs using this flag must be connected to graph inputs. All operator inputs connected to the same graph input must use or omit this flag equivalently.

It is legal to connect operators whose connected inputs and outputs use different dimension counts, sizes, and data types. This implies that the tensor data blob is interpreted differently by each operator. The *TotalTensorSizeInBytes* field of connected tensor inputs and outputs must be identical, though. Operators should only read regions of tensors written by earlier operators. Any padding regions in the output of an operation (resulting from the use of strides) are not guaranteed to be read as zero by down-stream operators.

Availability

This API was introduced in DirectML version [1.1.0](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- IDMLDevice1::CompileGraph method
 - DML_GRAPH_NODE_DESC struct
 - DML_GRAPH_EDGE_DESC struct
-

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GRAPH_EDGE_DESC structure (directml.h)

Article 10/20/2021

A generic container for a connection within a graph of DirectML operators defined by [DML_GRAPH_DESC](#) and passed to [IDMLDevice1::CompileGraph](#).

Syntax

C++

```
struct DML_GRAPH_EDGE_DESC {
    DML_GRAPH_EDGE_TYPE Type;
    const void          *Desc;
};
```

Members

Type

Type: [DML_GRAPH_EDGE_TYPE](#)

The type of graph edge. See [DML_GRAPH_EDGE_TYPE](#) for available types, and [DML_GRAPH_DESC](#) for where to use each type.

Desc

Type: `_Field_size_(Inexpressible_("Dependent on edge type")) const void*`

A pointer to the graph edge description. The type of the pointed-to struct must match the value specified in *Type*.

Availability

This API was introduced in DirectML version [1.1.0](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method](#)
- [DML_GRAPH_DESC structure](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GRAPH_NODE_DESC structure (directml.h)

Article 02/22/2024

A generic container for a node within a graph of DirectML operators defined by [DML_GRAPH_DESC](#) and passed to [IDMLDevice1::CompileGraph](#).

Syntax

C++

```
struct DML_GRAPH_NODE_DESC {
    DML_GRAPH_NODE_TYPE Type;
    const void          *Desc;
};
```

Members

Type

Type: [DML_GRAPH_NODE_TYPE](#)

The type of graph node. See [DML_GRAPH_NODE_TYPE](#) for available types.

Desc

Type: `_Field_size_(Inexpressible_("Dependent on node type")) const void*`

A pointer to the graph node description. The type of the pointed-to struct must match the value specified in *Type*.

Availability

This API was introduced in DirectML version [1.1.0](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method](#)
- [DML_GRAPH_DESC structure](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GRU_OPERATOR_DESC structure (directml.h)

Article 05/09/2023

Performs a (standard layers) one-layer gated recurrent unit (GRU) function on the input. This operator uses multiple gates to perform this layer. These gates are performed multiple times in a loop dictated by the sequence length dimension and the *SequenceLengthsTensor*.

Equation for the forward direction

```
for (t = 0; t < seq_length; t++)  
{  
     $z_t = f(X_t * W_z^T + H_{t-1} * R_z^T + W_{bz} + R_{bz})$   
     $r_t = f(X_t * W_r^T + H_{t-1} * R_r^T + W_{br} + R_{br})$   
  
    if (LinearBeforeReset = 0)  
         $h_t = g(X_t * W_h^T + (r_t \odot H_{t-1}) * R_h^T + W_{bh} + R_{bh})$   
    else  
         $h_t = g(X_t * W_h^T + (r_t \odot (H_{t-1} * R_h^T + R_{bh}) + W_{bh})$   
  
     $H_t = ((1 - z_t) \odot h_t) + (z_t \odot H_{t-1})$   
}
```

Equation for the backward direction

```
for (t = seq_length - 1; t >= 0; t--)  
{  
     $z_t = f(X_t * W_{Bz}^T + H_{t-1} * R_{Bz}^T + W_{Bbz} + R_{Bbz})$   
     $r_t = f(X_t * W_{Br}^T + H_{t-1} * R_{Br}^T + W_{Bbr} + R_{Bbr})$   
  
    if (LinearBeforeReset = 0)  
         $h_t = g(X_t * W_{Bh}^T + (r_t \odot H_{t-1}) * R_{Bh}^T + W_{Bbh} + R_{Bbh})$   
    else  
         $h_t = g(X_t * W_{Bh}^T + (r_t \odot (H_{t-1} * R_{Bh}^T + R_{Bbh}) + W_{Bbh})$   
  
     $H_t = ((1 - z_t) \odot h_t) + (z_t \odot H_{t-1})$   
}
```

Equation legend

- t current GRU layer index.
- $t - 1$ previous GRU layer index. If backwards direction, then it means $t + 1$ index but still the previously calculated t .
- T signifies that a matrix will be transposed before use.
- $*$ signifies a matrix multiplication.
- $+$ signifies an element-wise addition of two matrices.
- \odot signifies an element-wise multiply of two matrices.
- H_t output of current GRU index t .
- $f(), g()$ activation functions dictated by ActivationDescs.
- X_t Sub-matrix of the InputTensor, which is defined by matrix size [1, batch_size, input_size], at index * t of the seq_length dimension.
- $W_{[zrh]}^T$ Forward sub-matrix defined in WeightTensor, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- $W_{B[zrh]}^T$ Backwards sub-matrix defined in WeightTensor, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- H_{t-1} an intermediate tensor which is defined to be the output of the previous layer. For the first index of t , $H_{\{t-1\}}$ is replaced with HiddenInitTensor. This is defined to be size [1, batch_size, hidden_size]. A different HiddenInitTensor sub-matrix is used for each direction.
- $R_{[zrh]}^T$ Forward sub-matrix of RecurrenceTensor, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- $R_{B[zrh]}^T$ Backwards sub-matrix of RecurrenceTensor, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- $W_{b[zrh]}, R_{b[zrh]}$ are the bias tensors of the weight tensor, and recurrence tensors, which are sub-matrices of BiasTensor. This matrix is size [1, hidden_size] and is broadcasted up to the required size which is [1, batch_size, hidden_size].
- $W_{Bb[zrh]}, R_{Bb[zrh]}$ are the bias tensors for the backwards direction
- z_t stands for update gate at index t .
- r_t stands for reset gate at index t .
- h_t stands for hidden gate at index t .

Syntax

C++

```
struct DML_GRU_OPERATOR_DESC {
    const DML_TENSOR_DESC           *InputTensor;
    const DML_TENSOR_DESC           *WeightTensor;
    const DML_TENSOR_DESC           *RecurrenceTensor;
    const DML_TENSOR_DESC           *BiasTensor;
    const DML_TENSOR_DESC           *HiddenInitTensor;
    const DML_TENSOR_DESC           *SequenceLengthsTensor;
    const DML_TENSOR_DESC           *OutputSequenceTensor;
    const DML_TENSOR_DESC           *OutputSingleTensor;
    UINT                           ActivationDescCount;
    const DML_OPERATOR_DESC         *ActivationDescs;
    DML_RECURRENT_NETWORK_DIRECTION Direction;
    BOOL                           LinearBeforeReset;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input data, X. Packed (and potentially padded) into one 4D tensor with the Sizes of { 1, seq_length, batch_size, input_size }. seq_length is the

dimension that is mapped to the index, t. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

WeightTensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the weight data, W. Concatenation of $W_{[zrh]}$ and $W_B[zrh]$ (if bidirectional). The tensor has `Sizes { 1, num_directions, 3 * hidden_size, input_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

RecurrenceTensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the recurrence data, R. Concatenation of $R_{[zrh]}$ and $R_B[zrh]$ (if bidirectional). The tensor has `Sizes { 1, num_directions, 3 * hidden_size, hidden_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

BiasTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the bias data, B. Concatenation of ($W_b[zrh]$, $R_b[zrh]$) and ($W_Bb[zrh]$, $R_Bb[zrh]$) (if bidirectional). The tensor has `Sizes { 1, 1, num_directions, 6 * hidden_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

HiddenInitTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the hidden node initializer tensor, H_{t-1} for the first loop index t. If not specified, then defaults to 0. This tensor has `Sizes { 1, num_directions, batch_size, hidden_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

SequenceLengthsTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing an independent `seq_length` for each element in the batch. If not specified, then all sequences in the batch have length `seq_length`. This tensor has `Sizes { 1, 1, 1, batch_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

`OutputSequenceTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor with which to write the concatenation of all the intermediate output values of the hidden nodes, H_t . This tensor has `Sizes { seq_length, num_directions, batch_size, hidden_size }`. `seq_length` is mapped to the loop index t .

`OutputSingleTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor with which to write the last output value of the hidden nodes, H_t . This tensor has `Sizes { 1, num_directions, batch_size, hidden_size }`.

`ActivationDescCount`

Type: `UINT`

This field determines the size of the `ActivationDescs` array.

`ActivationDescs`

Type: `_Field_size_(ActivationDescCount) const DML_OPERATOR_DESC*`

An array of `DML_OPERATOR_DESC` containing the descriptions of the activation operators, $f()$ and $g()$. Both $f()$ and $g()$ are defined independently of direction, meaning that if `DML_RECURRENT_NETWORK_DIRECTION_FORWARD` or `DML_RECURRENT_NETWORK_DIRECTION_BACKWARD` are supplied in `Direction`, then two activations must be provided. If `DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL` is supplied, then four activations must be provided. For bidirectional, activations must be provided $f()$ and $g()$ for forward followed by $f()$ and $g()$ for backwards.

`Direction`

Type: `const DML_RECURRENT_NETWORK_DIRECTION*`

The direction of the operator—forward, backwards, or bidirectional.

`LinearBeforeReset`

Type: `BOOL`

`TRUE` to specify that, when computing the output of the hidden gate, the linear transformation should be applied before multiplying by the output of the reset gate.

Otherwise, FALSE.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

BiasTensor, *HiddenInitTensor*, *InputTensor*, *OutputSequenceTensor*, *OutputSingleTensor*, *RecurrenceTensor*, and *WeightTensor* must have the same *DataType*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
WeightTensor	Input	4	FLOAT32, FLOAT16
RecurrenceTensor	Input	4	FLOAT32, FLOAT16
BiasTensor	Optional input	4	FLOAT32, FLOAT16
HiddenInitTensor	Optional input	4	FLOAT32, FLOAT16
SequenceLengthsTensor	Optional input	4	UINT32
OutputSequenceTensor	Optional output	4	FLOAT32, FLOAT16
OutputSingleTensor	Optional output	4	FLOAT32, FLOAT16

Requirements

[] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_INPUT_GRAPH_EDGE_DESC structure (directml.h)

Article02/22/2024

Describes a connection within a graph of DirectML operators defined by [DML_GRAPH_DESC](#) and passed to [IDMLDevice1::CompileGraph](#). This structure is used to define a connection from a graph input to an input of an internal node.

Syntax

C++

```
struct DML_INPUT_GRAPH_EDGE_DESC {
    UINT      GraphInputIndex;
    UINT      ToNodeIndex;
    UINT      ToNodeInputIndex;
    const char *Name;
};
```

Members

`GraphInputIndex`

Type: [UINT](#)

The index of the graph input from which a connection to an internal node input is specified.

`ToNodeIndex`

Type: [UINT](#)

The index of the internal node onto which the connection from the graph input is specified.

`ToNodeInputIndex`

Type: [UINT](#)

The index of the input on the internal node where the connection is specified.

`Name`

Type: _Field_z_ _Maybenull_ const char*

An optional name for the graph connection. If provided, this might be used within certain error messages emitted by the DirectML debug layer.

Availability

This API was introduced in DirectML version 1.1.0.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method](#)
- [DML_GRAPH_DESC structure](#)
- [DML_GRAPH_EDGE_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_INTERMEDIATE_GRAPH_EDGE_DESC

C structure (directml.h)

Article 02/22/2024

Describes a connection within a graph of DirectML operators defined by [DML_GRAPH_DESC](#) and passed to [IDMLDevice1::CompileGraph](#). This structure is used to define a connection between internal nodes.

Syntax

C++

```
struct DML_INTERMEDIATE_GRAPH_EDGE_DESC {
    UINT      FromNodeIndex;
    UINT      FromNodeOutputIndex;
    UINT      ToNodeIndex;
    UINT      ToNodeInputIndex;
    const char *Name;
};
```

Members

`FromNodeIndex`

Type: [UINT](#)

The index of the graph node from which a connection to another node is specified.

`FromNodeOutputIndex`

Type: [UINT](#)

The index of the output on the node at index *FromNodeIndex* where the connection is specified.

`ToNodeIndex`

Type: [UINT](#)

The index of the graph node into which a connection from another node is specified.

`ToNodeInputIndex`

Type: [UINT](#)

The index of the input on the node at index *ToNodeIndex* where the connection is specified.

Name

Type: [_Field_z_ _Maybenull_ const char*](#)

An optional name for the graph connection. If provided, this might be used within certain error messages emitted by the DirectML debug layer.

Availability

This API was introduced in DirectML version [1.1.0](#).

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method, DML_GRAPH_DESC struct](#)
- [DML_GRAPH_DESC structure](#)
- [DML_GRAPH_EDGE_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_JOIN_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Concatenates an array of input tensors along a specified axis.

Input tensors may only be joined if their sizes are identical in all dimensions except for the join axis, which may contain any non-zero size. The output sizes are equal to the input sizes except for the join axis, which is the sum of all inputs' join axis size. These constraints are illustrated in the pseudocode below.

```
joinSize = 0;

for (i = 0; i < InputCount; i++) {
    assert(inputTensors[i]->DimensionCount == outputTensor->DimensionCount);
    for (dim = 0; dim < outputTensor->DimensionCount; dim++) {
        if (dim == Axis) { joinSize += inputTensors[i]->Sizes[dim]; }
        else { assert(inputTensors[i]->Sizes[dim] == outputTensor-
>Sizes[dim]); }
    }
}

assert(joinSize == outputTensor->Sizes[Axis]);
```

Joining a single input tensor simply produces a copy of the input tensor.

This operator is the inverse of [DML_SPLIT_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_JOIN_OPERATOR_DESC {
    UINT             InputCount;
    const DML_TENSOR_DESC *InputTensors;
    const DML_TENSOR_DESC *OutputTensor;
    UINT             Axis;
};
```

Members

`InputCount`

Type: **UINT**

This field determines the size of the *InputTensors* array. This value must be greater than 0.

`InputTensors`

Type: `_Field_size_(InputCount) const DML_TENSOR_DESC*`

An array containing the descriptions of the tensors to join into a single output tensor. All input tensors in this array must have the same sizes except for the join axis, which may have any non-zero value.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the joined input tensors into. The output sizes must have the same sizes as all input tensors except for the join axis, which must be equal to the sum of all inputs' join axis size.

`Axis`

Type: **UINT**

The index of the dimension of the input tensors to join. All input and output tensors must have identical sizes in all dimensions except for this axis. This value must be in the range `[0, OutputTensor.DimensionCount - 1]`.

Examples

Example 1. Joining tensors with only one possible axis

In this example, the tensors may only be joined along the fourth dimension (axis 3). Joining any other axis is not possible, since the tensors' size in the fourth dimension do not match.

```
InputCount: 2
```

```
Axis: 3
```

```
InputTensors[0]: (Sizes:{1, 1, 2, 3}, DataType:FLOAT32)
```

```
[[[[ 1,  2,  3],  
   [ 4,  5,  6]]]]  
  
InputTensors[1]: (Sizes:{1, 1, 2, 4}, DataType:FLOAT32)  
[[[[ 7,  8,  9, 10],  
   [11, 12, 13, 14]]]]  
  
OutputTensor: (Sizes:{1, 1, 2, 7}, DataType:FLOAT32)  
[[[[ 1,  2,  3,  7,  8,  9, 10],  
   [ 4,  5,  6, 11, 12, 13, 14]]]]
```

Example 2. Joining tensors with multiple possible axes:

The following examples use the same input tensors. Since all inputs have the same size in all dimensions, they can be joined along any dimension.

```
InputCount: 3  
  
InputTensors[0]: (Sizes:{1, 1, 2, 2}, DataType:FLOAT32)  
[[[[1, 2],  
   [3, 4]]]]  
  
InputTensors[1]: (Sizes:{1, 1, 2, 2}, DataType:FLOAT32)  
[[[[5, 6],  
   [7, 8]]]]  
  
InputTensors[2]: (Sizes:{1, 1, 2, 2}, DataType:FLOAT32)  
[[[[9, 10],  
   [11, 12]]]]
```

Joining axis 1:

```
Axis: 1  
  
OutputTensor: (Sizes:{1, 3, 2, 2}, DataType:FLOAT32)  
[[[[1, 2],  
   [3, 4]],  
  
  [[5, 6],  
   [7, 8]],  
  
  [[9, 10],  
   [11, 12]]]]
```

Joining axis 2:

```
Axis: 2

OutputTensor: (Sizes:{1, 1, 6, 2}, DataType:FLOAT32)
[[[1, 2],
 [3, 4],
 [5, 6],
 [7, 8],
 [9, 10],
 [11, 12]]]
```

Joining axis 3:

```
Axis: 3

OutputTensor: (Sizes:{1, 1, 2, 6}, DataType:FLOAT32)
[[[1, 2, 5, 6, 9, 10],
 [3, 4, 7, 8, 11, 12]]]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensors and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensors	Array of inputs	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32,

Tensor	Kind	Supported dimension counts	Supported data types
			INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensors	Array of inputs	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensors	Array of inputs	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensors	Array of inputs	4	FLOAT32, FLOAT16, INT32, INT16, UINT32, UINT16
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, UINT32, UINT16

Requirements

 Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_LOCAL_RESPONSE_NORMALIZATION_GRAD_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes backpropagation gradients for [local response normalization](#).

The data type and size of all tensors must be the same.

Syntax

C++

```
struct DML_LOCAL_RESPONSE_NORMALIZATION_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    BOOL CrossChannel;
    UINT LocalSize;
    FLOAT Alpha;
    FLOAT Beta;
    FLOAT Bias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the input data. This tensor's `Sizes` should be `{ BatchCount, ChannelCount, Height, Width }`.

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer.

`OutputGradientTensor`

Type: **const DML_TENSOR_DESC***

An output tensor containing the backpropagated gradients.

CrossChannel

Type: **BOOL**

TRUE if the LRN layer sums across channels; **FALSE** if the LRN layer sums across spatial dimensions.

LocalSize

Type: **UINT**

The maximum number of elements to sum over per dimension (the local region is clipped so that all elements are within bounds). If *CrossChannel* is **TRUE**, then this is the width and height of the local region. If *CrossChannel* is **FALSE**, then this is the number of elements in the local region. This value must be at least 1.

Alpha

Type: **FLOAT**

The value of the scaling parameter. We recommend a value of 0.0001 as default.

Beta

Type: **FLOAT**

The value of the exponent. We recommend a value of 0.75 as default.

Bias

Type: **FLOAT**

The value of bias. We recommend a value of 1 as default.

Remarks

Availability

This operator was introduced in **DML_FEATURE_LEVEL_3_1**.

Tensor constraints

InputGradientTensor, *InputTensor*, and *OutputGradientTensor* must have the same *DataType* and *Sizes*.

Tensor support

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
InputGradientTensor	Input	4	FLOAT32, FLOAT16
OutputGradientTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs a local response normalization (LRN) function on the input. This operator performs the following computation.

```
Output = Input / (Bias + (Alpha / LocalSize) * sum(Input^2 for every Input in the local region))^Beta
```

The data type and size of the input and output tensors must be the same.

Syntax

C++

```
struct DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    BOOL CrossChannel;
    UINT LocalSize;
    FLOAT Alpha;
    FLOAT Beta;
    FLOAT Bias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the input data. This tensor's `Sizes` should be `{ BatchCount, ChannelCount, Height, Width }`.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the results to. This tensor's *Sizes* should match the *InputTensor*.

`CrossChannel`

Type: **BOOL**

TRUE if the LRN layer sums across channels; otherwise, FALSE.

`LocalSize`

Type: **UINT**

The number of elements to sum over per dimension: Width, Height, and optionally Channel (if *CrossChannel* is set). This value must be at least 1.

`Alpha`

Type: **FLOAT**

The value of the scaling parameter. A value of 0.0001 is recommended as default.

`Beta`

Type: **FLOAT**

The value of the exponent. A value of 0.75 is recommended as default.

`Bias`

Type: **FLOAT**

The value of bias. A value of 1 is recommended as default.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *Sizes*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[!\[\]\(e1c7a47b2e0f0115bf7633a17b596066_img.jpg\) Yes](#)

[!\[\]\(6441fdb06a5efde3a669a74a40f6c18d_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_LP_NORMALIZATION_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs an L_p-normalization function along the specified axis of the input tensor.

Syntax

C++

```
struct DML_LP_NORMALIZATION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT Axis;
    FLOAT Epsilon;
    UINT P;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor containing the input data.

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the results to. This tensor's *Sizes* should match the *InputTensor*.

Axis

Type: **UINT**

The axis on which to apply normalization.

Epsilon

Type: **FLOAT**

The epsilon value to use to avoid division by zero. A value of 0.00001 is recommended as default.

P

Type: [UINT](#)

The order of the normalization (either 1 or 2).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*, *DimensionCount*, and *Sizes*.

Tensor support

DML_FEATURE_LEVEL_3_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_LP_POOLING_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Computes the L_p-normalized value across the elements within the sliding window over the input tensor.

Syntax

C++

```
struct DML_LP_POOLING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
    const UINT *EndPadding;
    UINT P;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

An input tensor with `Sizes { BatchCount, ChannelCount, Height, Width }` for 4D, and `{ BatchCount, ChannelCount, Depth, Height, Width }` for 5D.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write to. The `Sizes` of the output tensor can be computed as follows.

C++

```
OutputTensor->Sizes[0] = InputTensor->Sizes[0];
OutputTensor->Sizes[1] = InputTensor->Sizes[1];
```

```
for (UINT i = 0; i < DimensionCount; ++i) {
    UINT PaddedSize = InputTensor->Sizes[i + 2] + StartPadding[i] +
EndPadding[i];
    OutputTensor->Sizes[i + 2] = (PaddedSize - WindowSizes[i]) / Strides[i] +
1;
}
```

`DimensionCount`

Type: **UINT**

The number of spatial dimensions of the input tensor *InputTensor*, which also corresponds to the number of dimensions of the sliding window *WindowSize*. This value also determines the size of the *Strides*, *StartPadding*, and *EndPadding* arrays. It should be set to 2 when *InputTensor* is 4D, and 3 when it's a 5D tensor.

`Strides`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the strides for the sliding window dimensions of sizes `{ Height, Width }` when the *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`WindowSize`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the dimensions of the sliding window in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the number of padding elements to be applied to the beginning of each spatial dimension of the input tensor *InputTensor*. The values are in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the number of padding elements to be applied to the end of each spatial dimension of the input tensor *InputTensor*. The values are in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

P

Type: [UINT](#)

The value of the `P` variable in the L_p-normalization function $Y = (X_1^P + X_2^P + \dots + X_n^P)^{1/P}$, where `X1` to `Xn` representing each of the values within the sliding window. In common use cases, this value is either set to 1 or 2, representing either the L1 or L2 normalization respectively.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)

[No](#)

DML_LSTM_OPERATOR_DESC structure (directml.h)

Article05/09/2023

Performs a one-layer long short term memory (LSTM) function on the input. This operator uses multiple gates to perform this layer. These gates are performed multiple times in a loop, dictated by the sequence length dimension and the *SequenceLengthsTensor*.

Equation for the forward direction

```
for (t = 0; t < seq_length; t++)
{
     $i_t = f(\text{clip}(X_t * W_i^T + H_{t-1} * R_i^T + P_i \odot C_{t-1} + W_{bi} + R_{bi}))$ 
    if (CoupleInputAndForget = 0)
         $f_t = f(\text{clip}(X_t * W_f^T + H_{t-1} * R_f^T + P_f \odot C_{t-1} + W_{bf} + R_{bf}))$ 
    else
         $f_t = 1.0 - i_t$ 

     $c_t = g(\text{clip}(X_t * W_c^T + H_{t-1} * R_c^T + W_{bc} + R_{bc}))$ 
     $C_t = f_t \odot C_{t-1} + i_t \odot c_t$ 
     $o_t = f(\text{clip}(X_t * W_o^T + H_{t-1} * R_o^T + P_o \odot C_t + W_{bo} + R_{bo}))$ 
     $H_t = o_t \odot h(C_t)$ 
}
```

Equation for the backward direction

```
for (t = seq_length - 1; t >= 0; t--)
{
     $i_t = f(\text{clip}(X_t * W_{Bi}^T + H_{t-1} * R_{Bi}^T + P_i \odot C_{t-1} + W_{Bbi} + R_{Bbi}))$ 
    if (CoupleInputAndForget = 0)
         $f_t = f(\text{clip}(X_t * W_{Bf}^T + H_{t-1} * R_{Bf}^T + P_f \odot C_{t-1} + W_{Bbf} + R_{Bbf}))$ 
    else
         $f_t = 1.0 - i_t$ 

     $c_t = g(\text{clip}(X_t * W_{Bc}^T + H_{t-1} * R_{Bc}^T + W_{Bbc} + R_{Bbc}))$ 
     $C_t = f_t \odot C_{t-1} + i_t \odot c_t$ 
     $o_t = f(\text{clip}(X_t * W_{Bo}^T + H_{t-1} * R_{Bo}^T + P_o \odot C_t + W_{Bbo} + R_{Bbo}))$ 
     $H_t = o_t \odot h(C_t)$ 
}
```

Equation legend

- t current LSTM layer index.
- $t - 1$ previous LSTM layer index. If backwards direction, then it means $t + 1$ index but still the previously calculated t .
- T signifies that a matrix will be transposed before use.
- $*$ signifies a matrix multiplication.
- $+$ signifies an element-wise addition of two matrices.
- \odot signifies an element-wise multiply of two matrices.
- $f(), g(), h()$ activation functions dictated by ActivationDescs.
- clip() optional clipping operation, which is controlled by UseClipThreshold.
- X_t Sub-matrix of the InputTensor, which is defined by matrix size [1, batch_size, input_size], at index * t of the seq_length dimension.
- $W_{[iofc]}^T$ Forward sub-matrix defined in WeightTensor, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- $W_{B[iofc]}^T$ Backwards sub-matrix defined in WeightTensor, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- $R_{[iofc]}^T$ Forward sub-matrix of RecurrenceTensor, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- $R_{B[iofc]}^T$ Backwards sub-matrix of RecurrenceTensor, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- H_t output of current LSTM index t .
- H_{t-1} an intermediate tensor that's defined to be the output of the previous layer. For the first index of t , $-H_{t-1}$ is replaced with HiddenInitTensor. This is defined to be size [1, batch_size, hidden_size]. A different HiddenInitTensor sub-matrix is used for each direction.
- C_t is the tensor containing the Cell Memory for index t .
- C_{t-1} an intermediate tensor that's defined to be the Cell Memory of the previous layer. For the first index of t , C_{t-1} is replaced with CellMemoryInitTensor. This is defined to be size [1, batch_size, hidden_size]. A different CellMemoryInitTensor sub-matrix is used for each direction.
- $W_{b[iofc]}, R_{b[iofc]}$ are the bias tensors for the weight tensor, and recurrence tensor, which are sub-matrices of BiasTensor. These matrices are size [1,hidden_size], and are broadcasted up to the required size, which is [1, batch_size, hidden_size].
- $W_{Bb[iofc]}, R_{Bb[iofc]}$ are the bias tensors for the backwards direction.
- i_t stands for input gate at index t .
- o_t stands for output gate at index t .
- f_t stands for forget gate at index t .
- c_t stands for cell gate at index t .

Syntax

C++

```
struct DML_LSTM_OPERATOR_DESC {
    const DML_TENSOR_DESC           *InputTensor;
    const DML_TENSOR_DESC           *WeightTensor;
    const DML_TENSOR_DESC           *RecurrenceTensor;
    const DML_TENSOR_DESC           *BiasTensor;
    const DML_TENSOR_DESC           *HiddenInitTensor;
    const DML_TENSOR_DESC           *CellMemInitTensor;
    const DML_TENSOR_DESC           *SequenceLengthsTensor;
    const DML_TENSOR_DESC           *PeepholeTensor;
    const DML_TENSOR_DESC           *OutputSequenceTensor;
    const DML_TENSOR_DESC           *OutputSingleTensor;
    const DML_TENSOR_DESC           *OutputCellsSingleTensor;
    UINT                           ActivationDescCount;
    const DML_OPERATOR_DESC          *ActivationDescs;
    DML_RECURRENT_NETWORK_DIRECTION Direction;
    float                          ClipThreshold;
    BOOL                           UseClipThreshold;
    BOOL                           CoupleInputForget;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input data, X. Packed (and potentially padded) into one 4-D tensor with the sizes of `{ 1, seq_length, batch_size, input_size }`. `seq_length` is the dimension that is mapped to the index, t. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

WeightTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the weight data, W. Concatenation of $W_{[iofc]}$ and $W_B[iofc]$ (if bidirectional). The tensor has sizes `{ 1, num_directions, 4 * hidden_size, input_size }`. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

RecurrenceTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the recurrence data, R. Concatenation of $R_{[iofc]}$ and $R_B[iofc]$ (if bidirectional). This tensor has sizes `{ 1, num_directions, 4 * hidden_size, hidden_size }`. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

BiasTensor

Type: **_Maybenull_ const DML_TENSOR_DESC***

An optional tensor containing the bias data, B. Concatenation of `{ W_b[iofc], R_b[iofc] }`, and `{ W_Bb[iofc], R_Bb[iofc] }` (if bidirectional). This tensor has sizes `{ 1, 1, num_directions, 8 * hidden_size }`. If not specified, then defaults to 0 bias. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

HiddenInitTensor

Type: **_Maybenull_ const DML_TENSOR_DESC***

An optional tensor containing the hidden node initializer data, $H_{(t-1)}$. Contents of this tensor are only used on the first loop index t. If not specified, then defaults to 0. This tensor has sizes `{ 1, num_directions, batch_size, hidden_size }`. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

CellMemInitTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the cell initializer data, $C_{(t-1)}$. Contents of this tensor are only used on the first loop index t . If not specified, then defaults to 0. This tensor has sizes `{ 1, num_directions, batch_size, hidden_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

`SequenceLengthsTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing an independent `seq_length` for each element in the batch. If not specified, then all sequences in the batch have length `seq_length`. This tensor has sizes `{ 1, 1, 1, batch_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

`PeepholeTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the weight data for peepholes, P . If not specified, then defaults to 0. Concatenation of $P_{[iof]}$ and $P_B[iof]$ (if bidirectional). This tensor has sizes `{ 1, 1, num_directions, 3 * hidden_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

`OutputSequenceTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor with which to write the concatenation of all the intermediate output values of the hidden nodes, H_t . This tensor has sizes `{ seq_length, num_directions, batch_size, hidden_size }`. `seq_length` is mapped to the loop index t .

`OutputSingleTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor with which to write the last output value of the hidden nodes, H_t . This tensor has sizes `{ 1, num_directions, batch_size, hidden_size }`.

`OutputCellSingleTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor with which to write the last output value of the cell, C_t . This tensor has sizes `{ 1, num_directions, batch_size, hidden_size }`.

`ActivationDescCount`

Type: **UINT**

This field determines the size of the *ActivationDescs* array.

`ActivationDescs`

Type: `_Field_size_(ActivationDescCount) const DML_OPERATOR_DESC*`

An array of **DML_OPERATOR_DESC** containing the descriptions of the activation operators f(), g(), and h(). f(), g(), and h() are defined independently of direction, meaning that if **DML_RECURRENT_NETWORK_DIRECTION_FORWARD** or **DML_RECURRENT_NETWORK_DIRECTION_BACKWARD** are supplied in *Direction*, then three activations must be provided. If **DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL** is defined, then six activations must be provided. For bidirectional, activations must be provided f(), g(), and h() for forward followed by f(), g(), and h() for backwards.

`Direction`

Type: `const DML_RECURRENT_NETWORK_DIRECTION*`

The direction of the operator: forward, backward, or bidirectional.

`ClipThreshold`

Type: **float**

The cell clip threshold. Clipping bounds the elements of a tensor in the range of [-`ClipThreshold`, +`ClipThreshold`], and is applied to the input of activations.

`UseClipThreshold`

Type: **BOOL**

TRUE if *ClipThreshold* should be used. Otherwise, **FALSE**.

`CoupleInputForget`

Type: **BOOL**

TRUE if the input and forget gates should be coupled. Otherwise, **FALSE**.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

BiasTensor, *CellMemInitTensor*, *HiddenInitTensor*, *InputTensor*, *OutputCellSingleTensor*, *OutputSequenceTensor*, *OutputSingleTensor*, *PeepholeTensor*, *RecurrenceTensor*, and *WeightTensor* must have the same *DataType*.

Tensor support

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
WeightTensor	Input	4	FLOAT32, FLOAT16
RecurrenceTensor	Input	4	FLOAT32, FLOAT16
BiasTensor	Optional input	4	FLOAT32, FLOAT16
HiddenInitTensor	Optional input	4	FLOAT32, FLOAT16
CellMemInitTensor	Optional input	4	FLOAT32, FLOAT16
SequenceLengthsTensor	Optional input	4	UINT32
PeepholeTensor	Optional input	4	FLOAT32, FLOAT16
OutputSequenceTensor	Optional output	4	FLOAT32, FLOAT16
OutputSingleTensor	Optional output	4	FLOAT32, FLOAT16
OutputCellSingleTensor	Optional output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_MATRIX_MULTIPLY_INTEGER_OPERATOR_DESC structure (directml.h)

Article12/02/2022

Performs a matrix multiplication function on integer data.

This operator requires the matrix multiply input tensors to be 4D, which are formatted as { BatchCount, ChannelCount, Height, Width }. The matrix multiply operator will perform BatchCount * ChannelCount number of independent matrix multiplications.

For example, if *ATensor* has Sizes of { BatchCount, ChannelCount, M, K }, and *BTensor* has Sizes of { BatchCount, ChannelCount, K, N }, and *OutputTensor* has Sizes of { BatchCount, ChannelCount, M, N }, then the matrix multiply operator will perform BatchCount * ChannelCount independent matrix multiplications of dimensions {M,K} x {K,N} = {M,N}.

Syntax

C++

```
struct DML_MATRIX_MULTIPLY_INTEGER_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *AZeroPointTensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *BZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the A data. This tensor's dimensions should be { BatchCount, ChannelCount, M, K }.

AZeroPointTensor

Type: _Maybenull_ **const DML_TENSOR_DESC***

An optional tensor containing the *ATensor* zero point data. The expected dimensions of the `AZeroPointTensor` are `{ 1, 1, 1, 1 }` if per tensor quantization is required, or `{ 1, 1, M, 1 }` if per-row quantization is required. These zero point values are used for dequantizing the *ATensor* values.

`BTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the *B* data. This tensor's dimensions should be `{ BatchCount, ChannelCount, K, N }`.

`BZeroPointTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the *BTensor* zero point data. The expected dimensions of the `BZeroPointTensor` are `{ 1, 1, 1, 1 }` if per tensor quantization is required, or `{ 1, 1, 1, N }` if per column quantization is required. These zero point values are used for dequantizing the *BTensor* values.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor with which to write the results to. This tensor's dimensions are `{ BatchCount, ChannelCount, M, N }`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *BTensor* and *BZeroPointTensor* must have the same *DataType*.
- *ATensor* and *AZeroPointTensor* must have the same *DataType*.

Tensor support

`DML_FEATURE_LEVEL_5_2` and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
ATensor	Input	{ [BatchCount], [ChannelCount], M, K }	2 to 4	INT8, UINT8
AZeroPointTensor	Optional input	{ [1], [1], AZeroPointCount, [1] }	1 to 4	INT8, UINT8
BTensor	Input	{ [BatchCount], [ChannelCount], K, N }	2 to 4	INT8, UINT8
BZeroPointTensor	Optional input	{ [1], [1], [1], BZeroPointCount }	1 to 4	INT8, UINT8
OutputTensor	Output	{ [BatchCount], [ChannelCount], M, N }	2 to 4	INT32

DML_FEATURE_LEVEL_4_0 and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
ATensor	Input	{ [BatchCount], [ChannelCount], M, K }	2 to 4	INT8, UINT8
AZeroPointTensor	Optional input	{ [1], [1], AZeroPointCount, [1] }	1 to 4	INT8, UINT8
BTensor	Input	{ [BatchCount], [ChannelCount], K, N }	2 to 4	INT8, UINT8
BZeroPointTensor	Optional input	{ [1], [1], 1, BZeroPointCount }	2 to 4	INT8, UINT8
OutputTensor	Output	{ [BatchCount], [ChannelCount], M, N }	2 to 4	INT32

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
ATensor	Input	{ BatchCount, ChannelCount, M, K }	4	INT8, UINT8
AZeroPointTensor	Optional input	{ 1, 1, AZeroPointCount, 1 }	4	INT8, UINT8
BTensor	Input	{ BatchCount, ChannelCount, K, N }	4	INT8, UINT8
BZeroPointTensor	Optional input	{ 1, 1, 1, BZeroPointCount }	4	INT8, UINT8
OutputTensor	Output	{ BatchCount, ChannelCount, M, N }	4	INT32

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

Yes

No

Provide product feedback | Get help at Microsoft Q&A

DML_MAX_POOLING_GRAD_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Computes backpropagation gradients for max pooling (see [DML_MAX_POOLING2_OPERATOR_DESC](#)).

Consider a 2x2 **DML_MAX_POOLING2_OPERATOR_DESC** without padding nor dilations and a stride of 1, which performs the following.

InputTensor	MaxPool	OutputTensor	IndicesTensor
$\begin{bmatrix} 1, 2, 3 \\ 2, 4, 2 \\ 5, 6, 7 \end{bmatrix}$	\rightarrow	$\begin{bmatrix} 4, 4 \\ 6, 7 \end{bmatrix}$	$\begin{bmatrix} 4, 4 \\ 7, 8 \end{bmatrix}$

The largest element of each 2x2 window in the input tensor produces one element of the output. Below is an example of the output of **DML_MAX_POOLING_GRAD_OPERATOR_DESC**, given similar parameters.

InputTensor	InputGradientTensor	MaxPoolGrad	OutputGradientTensor
$\begin{bmatrix} 1, 2, 3 \\ 2, 4, 2 \\ 5, 6, 7 \end{bmatrix}$	$\begin{bmatrix} 1, 2 \\ 4, 5 \end{bmatrix}$	\rightarrow	$\begin{bmatrix} 0, 0, 0 \\ 0, 3, 0 \\ 0, 4, 5 \end{bmatrix}$

In effect, this operator uses the *InputTensor* to determine the index of the largest element from each window, and distributes the values of *InputGradientTensor* into the *OutputGradientTensor* based on these indices. Where indices overlap, the values are summed. Any unreferenced output elements are zeroed.

In the case of a tie (where more than one element in a window has the same maximum value), the element with the lowest logical element index is chosen.

Syntax

C++

```
struct DML_MAX_POOLING_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
```

```
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
    const UINT *EndPadding;
    const UINT *Dilations;
};

};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input feature tensor. This is typically the same tensor that was provided as the *InputTensor* to [DML_MAX_POOLING2_OPERATOR_DESC](#) in the forward pass.

InputGradientTensor

Type: **const DML_TENSOR_DESC***

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. Typically this tensor would have the same sizes as the *output* of the corresponding [DML_MAX_POOLING2_OPERATOR_DESC](#) in the forward pass.

OutputGradientTensor

Type: **const DML_TENSOR_DESC***

An output tensor containing the backpropagated gradients. Typically this tensor would have the same sizes as the *input* of the corresponding [DML_MAX_POOLING2_OPERATOR_DESC](#) in the forward pass.

DimensionCount

Type: **UINT**

The number of elements in the *Strides*, *WindowSize*, *StartPadding*, *EndPadding*, and *Dilations* arrays. This value must equal the spatial dimension count (InputTensor's DimensionCount - 2). As this operator only supports 4D tensors, the only valid value for this parameter is 2.

Strides

Type: `_Field_size_(DimensionCount) const UINT*`

See *Strides* in [DML_MAX_POOLING2_OPERATOR_DESC](#).

`WindowSize`

Type: `_Field_size_(DimensionCount) const UINT*`

See *WindowSize* in [DML_MAX_POOLING2_OPERATOR_DESC](#).

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

See *StartPadding* in [DML_MAX_POOLING2_OPERATOR_DESC](#).

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

See *EndPadding* in [DML_MAX_POOLING2_OPERATOR_DESC](#).

`Dilations`

Type: `_Field_size_(DimensionCount) const UINT*`

See *Dilations* in [DML_MAX_POOLING2_OPERATOR_DESC](#).

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

- *InputGradientTensor*, *InputTensor*, and *OutputGradientTensor* must have the same *DataType* and *DimensionCount*.
- *InputTensor* and *OutputGradientTensor* must have the same *Sizes*.

Tensor support

DML_FEATURE_LEVEL_4_0 and above

[] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, ChannelCount, [InputDepth], InputHeight, InputWidth }	4 to 5	FLOAT32, FLOAT16
InputGradientTensor	Input	{ BatchCount, ChannelCount, [OutputDepth], OutputHeight, OutputWidth }	4 to 5	FLOAT32, FLOAT16
OutputGradientTensor	Output	{ BatchCount, ChannelCount, [InputDepth], InputHeight, InputWidth }	4 to 5	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, ChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16
InputGradientTensor	Input	{ BatchCount, ChannelCount, OutputHeight, OutputWidth }	4	FLOAT32, FLOAT16
OutputGradientTensor	Output	{ BatchCount, ChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348

Requirement	Value
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_MAX_POOLING_OPERATOR_DESC structure (directml.h)

Article01/21/2022

Computes the maximum value across the elements within the sliding window over the input tensor.

Syntax

C++

```
struct DML_MAX_POOLING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
    const UINT *EndPadding;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

An input tensor of Sizes { BatchCount, ChannelCount, Height, Width } if *InputTensor.DimensionCount* is 4, and { BatchCount, ChannelCount, Depth, Height, Weight } if *InputTensor.DimensionCount* is 5.

OutputTensor

Type: **const DML_TENSOR_DESC***

An output tensor to write the results to. The sizes of the output tensor can be computed as follows.

C++

```
OutputTensor->Sizes[0] = InputTensor->Sizes[0];
OutputTensor->Sizes[1] = InputTensor->Sizes[1];
```

```
for (UINT i = 0; i < DimensionCount; ++i) {
    UINT PaddedSize = InputTensor->Sizes[i + 2] + StartPadding[i] +
EndPadding[i];
    OutputTensor->Sizes[i + 2] = (PaddedSize - WindowSizes[i]) / Strides[i] +
1;
}
```

`DimensionCount`

Type: **UINT**

The number of spatial dimensions of the input tensor *InputTensor*, which also corresponds to the number of dimensions of the sliding window *WindowSize*. This value also determines the size of the *Strides*, *StartPadding*, and *EndPadding* arrays. It should be set to 2 when *InputTensor* is 4D, and 3 when it's a 5D tensor.

`Strides`

Type: **const UINT***

The strides for the sliding window dimensions of sizes `{ Height, Width }` when the *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`WindowSize`

Type: `_Field_size_(DimensionCount) const UINT*`

The dimensions of the sliding window in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The number of padding elements to be applied to the beginning of each spatial dimension of the input tensor *InputTensor*. The values are in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The number of padding elements to be applied to the end of each spatial dimension of the input tensor *InputTensor*. The values are in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

Remarks

A newer version of this operator, [DML_MAX_POOLING1_OPERATOR_DESC](#), was introduced in `DML_FEATURE_LEVEL_2_0`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML FEATURE LEVEL 5_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML FEATURE LEVEL 3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT8, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT8, UINT8

DML FEATURE LEVEL 1_0 and above

[\[\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[\[\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_MAX_POOLING1_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Computes the maximum value across the elements within the sliding window over the input tensor, and optionally returns the indices of the maximum values selected.

Syntax

C++

```
struct DML_MAX_POOLING1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_TENSOR_DESC *OutputIndicesTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
    const UINT *EndPadding;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

An input tensor of Sizes `{ BatchCount, ChannelCount, Height, Width }` if `InputTensor.DimensionCount` is 4, and `{ BatchCount, ChannelCount, Depth, Height, Weight }` if `InputTensor.DimensionCount` is 5.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor to write the results to. The sizes of the output tensor can be computed as follows.

C++

```
OutputTensor->Sizes[0] = InputTensor->Sizes[0];
OutputTensor->Sizes[1] = InputTensor->Sizes[1];
```

```
for (UINT i = 0; i < DimensionCount; ++i) {
    UINT PaddedSize = InputTensor->Sizes[i + 2] + StartPadding[i] +
EndPadding[i];
    OutputTensor->Sizes[i + 2] = (PaddedSize - WindowSizes[i]) / Strides[i] +
1;
}
```

OutputIndicesTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional output tensor of indices to the input tensor *InputTensor* of the maximum values produced and stored in the *OutputTensor*. These index values are zero-based and treat the input tensor as a contiguous one-dimensional array. When multiple elements within the sliding window have the same value, the later equal values are ignored and the index points to the first value encountered. Both the *OutputTensor* and *OutputIndicesTensor* have the same tensor sizes.

DimensionCount

Type: `UINT`

The number of spatial dimensions of the input tensor *InputTensor*, which also corresponds to the number of dimensions of the sliding window *WindowSize*. This value also determines the size of the *Strides*, *StartPadding*, and *EndPadding* arrays. It should be set to 2 when *InputTensor* is 4D, and 3 when it's a 5D tensor.

Strides

Type: `_Field_size_(DimensionCount) const UINT*`

The strides for the sliding window dimensions of sizes `{ Height, Width }` when the *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

WindowSize

Type: `_Field_size_(DimensionCount) const UINT*`

The dimensions of the sliding window in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

StartPadding

Type: `_Field_size_(DimensionCount) const UINT*`

The number of padding elements to be applied to the beginning of each spatial dimension of the input tensor *InputTensor*. The values are in { Height, Width } when *DimensionCount* is set to 2, or { Depth, Height, Width } when set to 3.

EndPadding

Type: _Field_size_(*DimensionCount*) const **UINT***

The number of padding elements to be applied to the end of each spatial dimension of the input tensor *InputTensor*. The values are in { Height, Width } when *DimensionCount* is set to 2, or { Depth, Height, Width } when set to 3.

Remarks

When *OutputIndicesTensor* is set to NULL, this operator is equivalent to [DML_MAX_POOLING_OPERATOR_DESC](#).

A newer version of this operator, [DML_MAX_POOLING2_OPERATOR_DESC](#), was introduced in [DML_FEATURE_LEVEL_2_1](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_0](#).

Tensor constraints

- *InputTensor*, *OutputIndicesTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *InputTensor* and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16,

Tensor	Kind	Supported dimension counts	Supported data types
			UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputIndicesTensor	Optional output	4 to 5	UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT8, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT8, UINT8
OutputIndicesTensor	Optional output	4 to 5	UINT32

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16
OutputIndicesTensor	Optional output	4 to 5	UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_MAX_POOLING2_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Computes the maximum value across the elements within the sliding window over the input tensor, and optionally returns the indices of the maximum values selected.

Syntax

C++

```
struct DML_MAX_POOLING2_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_TENSOR_DESC *OutputIndicesTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
    const UINT *EndPadding;
    const UINT *Dilations;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

An input tensor of `Sizes { BatchCount, ChannelCount, Height, Width }` if `InputTensor.DimensionCount` is 4, and `{ BatchCount, ChannelCount, Depth, Height, Weight }` if `InputTensor.DimensionCount` is 5.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor to write the results to. The sizes of the output tensor can be computed as follows.

C++

```
OutputTensor->Sizes[0] = InputTensor->Sizes[0];
OutputTensor->Sizes[1] = InputTensor->Sizes[1];

for (UINT i = 0; i < DimensionCount; ++i) {
    UINT PaddedSize = InputTensor->Sizes[i + 2] + StartPadding[i] +
EndPadding[i];
    OutputTensor->Sizes[i + 2] = (PaddedSize - WindowSizes[i]) / Strides[i] +
1;
}
```

OutputIndicesTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional output tensor of indices to the input tensor *InputTensor* of the maximum values produced and stored in the *OutputTensor*. These index values are zero-based and treat the input tensor as a contiguous one-dimensional array. When multiple elements within the sliding window have the same value, the later equal values are ignored and the index points to the first value encountered. Both the *OutputTensor* and *OutputIndicesTensor* have the same tensor sizes.

DimensionCount

Type: `UINT`

The number of spatial dimensions of the input tensor *InputTensor*, which also corresponds to the number of dimensions of the sliding window *WindowSize*. This value also determines the size of the *Strides*, *StartPadding*, and *EndPadding* arrays. It should be set to 2 when *InputTensor* is 4D, and 3 when it's a 5D tensor.

Strides

Type: `_Field_size_(DimensionCount) const UINT*`

The strides for the sliding window dimensions of sizes `{ Height, Width }` when the *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

WindowSize

Type: `_Field_size_(DimensionCount) const UINT*`

The dimensions of the sliding window in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

StartPadding

Type: `_Field_size_(DimensionCount) const UINT*`

The number of padding elements to be applied to the beginning of each spatial dimension of the input tensor *InputTensor*. The values are in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The number of padding elements to be applied to the end of each spatial dimension of the input tensor *InputTensor*. The values are in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

`Dilations`

Type: `_Field_size_(DimensionCount) const UINT*`

The values for each spatial dimension of the input tensor *InputTensor* by which an element within the sliding window is selected for every elements of that value. The values are in `{ Height, Width }` when *DimensionCount* is set to 2, or `{ Depth, Height, Width }` when set to 3.

Remarks

`DML_MAX_POOLING2_OPERATOR_DESC` supersedes the earlier version

`DML_MAX_POOLING_OPERATOR1_DESC` with an additional constant array *Dilations*. The two versions are equivalent when *Dilations* is set to `{ 1,1 }` for 4D input, or `{ 1,1,1 }` for 5D input features.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- *InputTensor*, *OutputIndicesTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *InputTensor* and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputIndicesTensor	Optional output	4 to 5	UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT8, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT8, UINT8
OutputIndicesTensor	Optional output	4 to 5	UINT32

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16
OutputIndicesTensor	Optional	4 to 5	UINT32

Tensor	Kind	Supported dimension counts	Supported data types
	output		

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_MAX_UNPOOLING_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Inverts a max-pooling operation (see [DML_MAX_POOLING_OPERATOR1_DESC](#) for details) by filling the output tensor *OutputTensor* with the values in the input tensor *InputTensor*, as obtained from a max-pooling operation, according to the index values provided in the *IndicesTensor*. The elements in the output tensor untouched by this process are left with zero values.

Syntax

C++

```
struct DML_MAX_UNPOOLING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: [const DML_TENSOR_DESC*](#)

An input tensor of *Sizes* { `Batch`, `Channel`, `Height`, `Width` }. The tensor values are obtained from the values in the *OutputTensor* of a max-pooling operation.

IndicesTensor

Type: [const DML_TENSOR_DESC*](#)

A tensor of indices to the output tensor *OutputTensor* for the values given in the input tensor *InputTensor*. These index values are zero-based, and treat the output tensor as a contiguous one-dimensional array. Both the *InputTensor* and *IndicesTensor* have the same tensor sizes. The tensor values are obtained from the *OutputIndicesTensor* of a max-pooling operation.

OutputTensor

Type: `const DML_TENSOR_DESC*`

An output tensor of the same number of dimensions as the input tensor.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
IndicesTensor	Input	4	UINT64, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
IndicesTensor	Input	4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC structure (directml.h)

Article05/09/2023

Performs a mean variance normalization function on the input tensor. This operator will calculate the mean and variance of the input tensor to perform normalization. This operator performs the following computation.

```
Output = FusedActivation(Scale * ((Input - Mean) / sqrt(Variance + Epsilon))  
+ Bias).
```

Syntax

C++

```
struct DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC {  
    const DML_TENSOR_DESC    *InputTensor;  
    const DML_TENSOR_DESC    *ScaleTensor;  
    const DML_TENSOR_DESC    *BiasTensor;  
    const DML_TENSOR_DESC    *OutputTensor;  
    BOOL                    CrossChannel;  
    BOOL                    NormalizeVariance;  
    FLOAT                  Epsilon;  
    const DML_OPERATOR_DESC *FusedActivation;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the Input data. This tensor's dimensions should be `{ BatchCount, ChannelCount, Height, Width }`.

`ScaleTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the Scale data. This tensor's dimensions should be `{BatchCount, ChannelCount, Height, Width}`. Any dimension can be replaced with 1 to broadcast in that dimension. If **DML_FEATURE_LEVEL** is less than **DML_FEATURE_LEVEL_5_2**, then this tensor is required if *BiasTensor* is present. If **DML_FEATURE_LEVEL** is greater than or equal to **DML_FEATURE_LEVEL_5_2**, then this tensor can be null regardless of the value of *BiasTensor*.

`BiasTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the bias data. This tensor's dimensions should be `{BatchCount, ChannelCount, Height, Width}`. Any dimension can be replaced with 1 to broadcast in that dimension. If **DML_FEATURE_LEVEL** is less than **DML_FEATURE_LEVEL_5_2**, then this tensor is required if *ScaleTensor* is present. If **DML_FEATURE_LEVEL** is greater than or equal to **DML_FEATURE_LEVEL_5_2**, then this tensor can be null regardless of the value of *ScaleTensor*.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor to write the results to. This tensor's dimensions are `{BatchCount, ChannelCount, Height, Width}`.

`CrossChannel`

Type: `BOOL`

When **TRUE**, the MeanVariance layer includes channels in the Mean and Variance calculations, meaning they are normalized across axes `{ChannelCount, Height, Width}`.

When **FALSE**, Mean and Variance calculations are normalized across axes `{Height, Width}` with each channel being independent.

`NormalizeVariance`

Type: `BOOL`

TRUE if the Normalization layer includes Variance in the normalization calculation. Otherwise, **FALSE**. If **FALSE**, then normalization equation is `Output = FusedActivation(Scale * (Input - Mean) + Bias)`.

`Epsilon`

Type: `FLOAT`

The epsilon value to use to avoid division by zero. A value of 0.00001 is recommended as default.

FusedActivation

Type: `_Maybenull_ const DML_OPERATOR_DESC*`

An optional fused activation layer to apply after the normalization. For more info, see [Using fused operators for improved performance](#).

Remarks

A newer version of this operator,

[DML_MEAN_VARIANCE_NORMALIZATION1_OPERATOR_DESC](#), was introduced in [DML_FEATURE_LEVEL_2_1](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

- *InputTensor* and *OutputTensor* must have the same *Sizes*.
- *BiasTensor*, *InputTensor*, *OutputTensor*, and *ScaleTensor* must have the same *DataType*.

Tensor support

[+] [Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, ChannelCount, Height, Width }	4	FLOAT32, FLOAT16
ScaleTensor	Optional input	{ ScaleBatchCount, ScaleChannelCount, ScaleHeight, ScaleWidth }	4	FLOAT32, FLOAT16

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
BiasTensor	Optional input	{ BiasBatchCount, BiasChannelCount, BiasHeight, BiasWidth }	4	FLOAT32, FLOAT16
OutputTensor	Output	{ BatchCount, ChannelCount, Height, Width }	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

- [Using fused operators for improved performance](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_MEAN_VARIANCE_NORMALIZATION1_OPERATOR_DESC structure (directml.h)

Article05/02/2023

Performs a mean variance normalization function on the input tensor. This operator will calculate the mean and variance of the input tensor to perform normalization. This operator performs the following computation.

```
Output = FusedActivation(Scale * ((Input - Mean) / sqrt(Variance + Epsilon))  
+ Bias).
```

Syntax

C++

```
struct DML_MEAN_VARIANCE_NORMALIZATION1_OPERATOR_DESC {  
    const DML_TENSOR_DESC    *InputTensor;  
    const DML_TENSOR_DESC    *ScaleTensor;  
    const DML_TENSOR_DESC    *BiasTensor;  
    const DML_TENSOR_DESC    *OutputTensor;  
    UINT                     AxisCount;  
    const UINT                *Axes;  
    BOOL                     NormalizeVariance;  
    FLOAT                    Epsilon;  
    const DML_OPERATOR_DESC  *FusedActivation;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the Input data. This tensor's dimensions should be { BatchCount, ChannelCount, Height, Width }.

ScaleTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the Scale data.

If `DML_FEATURE_LEVEL` is less than `DML_FEATURE_LEVEL_4_0`, then this tensor's dimensions should be `{ ScaleBatchCount, ChannelCount, ScaleHeight, ScaleWidth }`.

The dimensions `ScaleBatchCount`, `ScaleHeight`, and `ScaleWidth` should either match `InputTensor`, or be set to 1 to automatically broadcast those dimensions across the input.

If `DML_FEATURE_LEVEL` is greater than or equal to `DML_FEATURE_LEVEL_4_0`, then any dimension can be set to 1, and be automatically broadcast to match `InputTensor`.

If `DML_FEATURE_LEVEL` is less than `DML_FEATURE_LEVEL_5_2`, then this tensor is required if `BiasTensor` is present. If `DML_FEATURE_LEVEL` is greater than or equal to `DML_FEATURE_LEVEL_5_2`, then this tensor can be null regardless of the value of `BiasTensor`.

`BiasTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the Bias data.

If `DML_FEATURE_LEVEL` is less than `DML_FEATURE_LEVEL_4_0`, then this tensor's dimensions should be `{ BiasBatchCount, ChannelCount, BiasHeight, BiasWidth }`. The dimensions `BiasBatchCount`, `BiasHeight`, and `BiasWidth` should either match `InputTensor`, or be set to 1 to automatically broadcast those dimensions across the input.

If `DML_FEATURE_LEVEL` is greater than or equal to `DML_FEATURE_LEVEL_4_0`, then any dimension can be set to 1, and be automatically broadcast to match `InputTensor`.

If `DML_FEATURE_LEVEL` is less than `DML_FEATURE_LEVEL_5_2`, then this tensor is required if `ScaleTensor` is present. If `DML_FEATURE_LEVEL` is greater than or equal to `DML_FEATURE_LEVEL_5_2`, then this tensor can be null regardless of the value of `ScaleTensor`.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor to write the results to. This tensor's dimensions are `{ BatchCount, ChannelCount, Height, Width }`.

`AxisCount`

Type: `UINT`

The number of axes. This field determines the size of the `Axes` array.

Axes

Type: `_Field_size_(AxisCount) const UINT*`

The axes along which to calculate the Mean and Variance.

NormalizeVariance

Type: `BOOL`

TRUE if the Normalization layer includes Variance in the normalization calculation.

Otherwise, **FALSE**. If **FALSE**, then normalization equation is `Output =`

`FusedActivation(Scale * (Input - Mean) + Bias)`.

Epsilon

Type: `FLOAT`

The epsilon value to use to avoid division by zero. A value of 0.00001 is recommended as default.

FusedActivation

Type: `_Maybenull_ const DML_OPERATOR_DESC*`

An optional fused activation layer to apply after the normalization.

Remarks

`DML_MEAN_VARIANCE_NORMALIZATION1_OPERATOR_DESC` is a superset of functionality of `DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC`. Here, setting the `Axes` array to `{ 2, 3 }` is the equivalent of setting `CrossChannel` to **FALSE** in `DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC`; while setting the `Axes` array to `{ 1, 2, 3 }` is equivalent of setting `CrossChannel` to **TRUE**.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

BiasTensor, *InputTensor*, *OutputTensor*, and *ScaleTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
ScaleTensor	Optional input	1 to 8	FLOAT32, FLOAT16
BiasTensor	Optional input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
ScaleTensor	Optional input	4	FLOAT32, FLOAT16
BiasTensor	Optional input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

[Using fused operators for improved performance](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_MULTIHEAD_ATTENTION_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Performs a multi-head attention operation (for more info, see [Attention is all you need](#)). Exactly one *Query*, *Key* and *Value* tensor must be present, whether or not they're stacked. For example, if *StackedQueryKey* is provided, both the *Query* and *Key* tensors must be null, since they're already provided in a stacked layout. The same goes for *StackedKeyValue* and *StackedQueryKeyValue*. The stacked tensors always have five dimensions, and are always stacked on the fourth dimension.

Logically, the algorithm can be decomposed into the following operations (operations in brackets are optional):

```
[Add Bias to query/key/value] -> GEMM(Query, Transposed(Key)) * Scale ->
[Add RelativePositionBias] -> [Add Mask] -> Softmax -> GEMM(SoftmaxResult,
Value);
```

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) version 1.12 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_MULTIHEAD_ATTENTION_OPERATOR_DESC
{
    _Maybenull_ const DML_TENSOR_DESC* QueryTensor;
    _Maybenull_ const DML_TENSOR_DESC* KeyTensor;
    _Maybenull_ const DML_TENSOR_DESC* ValueTensor;
    _Maybenull_ const DML_TENSOR_DESC* StackedQueryKeyTensor;
    _Maybenull_ const DML_TENSOR_DESC* StackedKeyValueTensor;
    _Maybenull_ const DML_TENSOR_DESC* StackedQueryKeyValueTensor;
    _Maybenull_ const DML_TENSOR_DESC* BiasTensor;
    _Maybenull_ const DML_TENSOR_DESC* MaskTensor;
    _Maybenull_ const DML_TENSOR_DESC* RelativePositionBiasTensor;
    _Maybenull_ const DML_TENSOR_DESC* PastKeyTensor;
    _Maybenull_ const DML_TENSOR_DESC* PastValueTensor;
    const DML_TENSOR_DESC* OutputTensor;
```

```
_Maybenull_ const DML_TENSOR_DESC* OutputPresentKeyTensor;
_Maybenull_ const DML_TENSOR_DESC* OutputPresentValueTensor;
FLOAT Scale;
FLOAT MaskFilterValue;
UINT HeadCount;
DML_MULTIHEAD_ATTENTION_MASK_TYPE MaskType;
};
```

Members

QueryTensor

Type: _Maybenull_ const [DML_TENSOR_DESC*](#)

Query with shape [batchSize, sequenceLength, hiddenSize], where hiddenSize = headCount * headSize. This tensor is mutually exclusive with *StackedQueryKeyTensor* and *StackedQueryKeyValueTensor*. The tensor can also have 4 or 5 dimensions, as long as the leading dimensions are 1's.

KeyTensor

Type: _Maybenull_ const [DML_TENSOR_DESC*](#)

Key with shape [batchSize, keyValueSequenceLength, hiddenSize], where hiddenSize = headCount * headSize. This tensor is mutually exclusive with *StackedQueryKeyTensor*, *StackedKeyValueTensor*, and *StackedQueryKeyValueTensor*. The tensor can also have 4 or 5 dimensions, as long as the leading dimensions are 1's.

ValueTensor

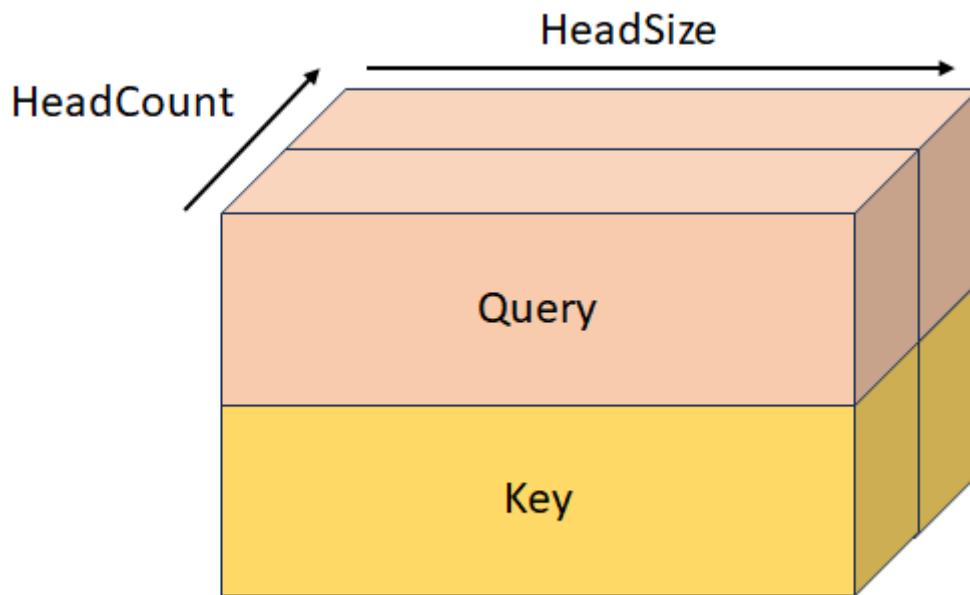
Type: _Maybenull_ const [DML_TENSOR_DESC*](#)

Value with shape [batchSize, keyValueSequenceLength, valueHiddenSize], where valueHiddenSize = headCount * valueHeadSize. This tensor is mutually exclusive with *StackedKeyValueTensor*, and *StackedQueryKeyValueTensor*. The tensor can also have 4 or 5 dimensions, as long as the leading dimensions are 1's.

StackedQueryKeyTensor

Type: _Maybenull_ const [DML_TENSOR_DESC*](#)

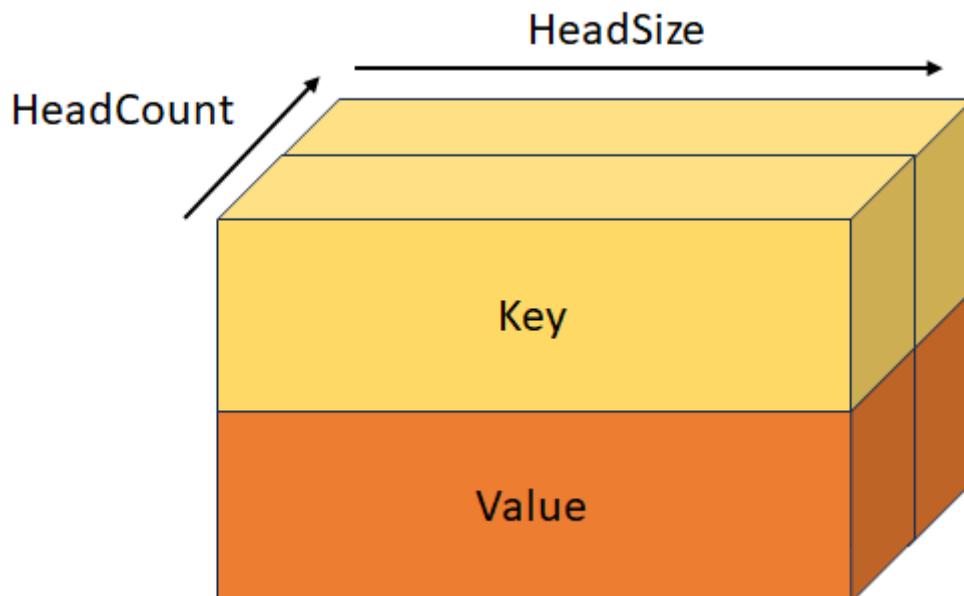
Stacked query and key with shape [batchSize, sequenceLength, headCount, 2, headSize]. This tensor is mutually exclusive with *QueryTensor*, *KeyTensor*, *StackedKeyValueTensor*, and *StackedQueryKeyValueTensor*.



`StackedKeyValueTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

Stacked key and value with shape `[batchSize, keyValueSequenceLength, headCount, 2, headSize]`. This tensor is mutually exclusive with `KeyTensor`, `ValueTensor`, `StackedQueryKeyTensor`, and `StackedQueryKeyValueTensor`.

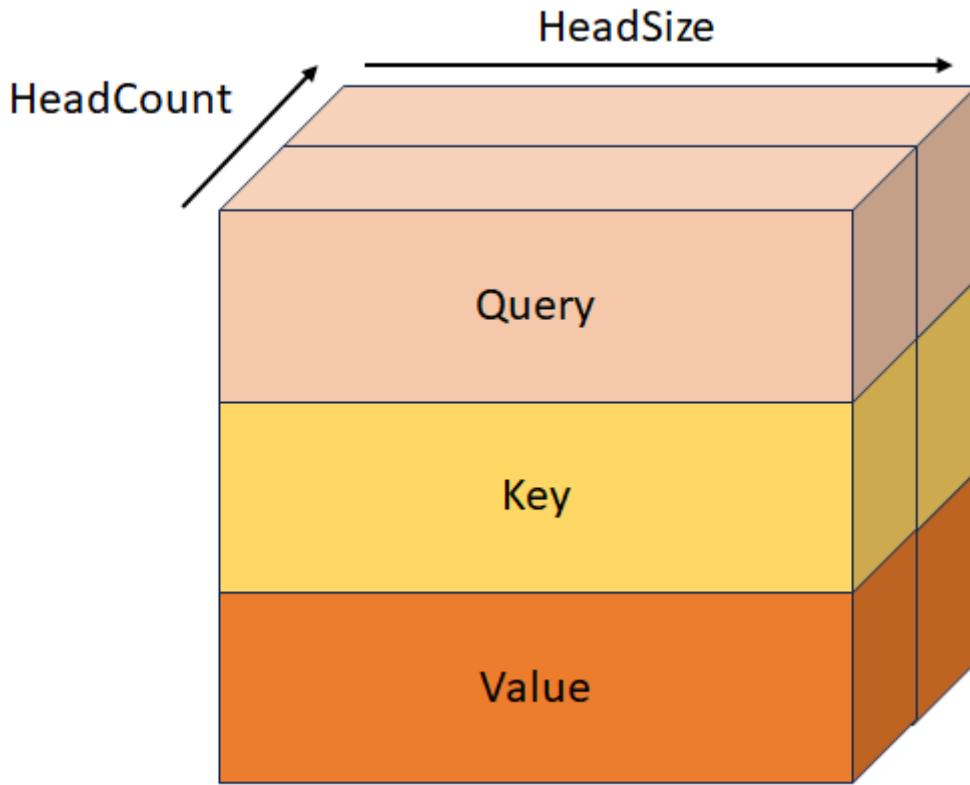


`StackedQueryKeyValueTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

Stacked query, key, and value with shape `[batchSize, sequenceLength, headCount, 3, headSize]`. This tensor is mutually exclusive with `QueryTensor`, `KeyTensor`, `ValueTensor`, `StackedQueryKeyTensor`, and `StackedQueryKeyValueTensor`.

StackedQueryKeyTensor, and *StackedKeyValueTensor*.



BiasTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

This is the bias, of shape `[hiddenSize + hiddenSize + valueHiddenSize]`, which gets added to *Query/Key/Value* before the first GEMM operation. This tensor can also have 2, 3, 4 or 5 dimensions, as long as the leading dimensions are 1's.

MaskTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

This is the mask that determines which elements get their value set to *MaskFilterValue* after the QxK GEMM operation. The behavior of this mask depends on the value of *MaskType*, and gets applied after *RelativePositionBiasTensor*, or after the first GEMM operation if *RelativePositionBiasTensor* is null. See the definition of *MaskType* for more information.

RelativePositionBiasTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

This is the bias that gets added to the result of the first GEMM operation.

PastKeyTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

Key tensor from the previous iteration with shape `[batchSize, headCount, pastSequenceLength, headSize]`. When this tensor is not null, it gets concatenated with the key tensor, which results in a tensor of shape `[batchSize, headCount, pastSequenceLength + keyValueSequenceLength, headSize]`.

PastValueTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

Value tensor from the previous iteration with shape `[batchSize, headCount, pastSequenceLength, headSize]`. When this tensor is not null, it gets concatenated with `ValueDesc` which results in a tensor of shape `[batchSize, headCount, pastSequenceLength + keyValueSequenceLength, headSize]`.

OutputTensor

Type: `const DML_TENSOR_DESC*`

Output, of shape `[batchSize, sequenceLength, valueHiddenSize]`.

OutputPresentKeyTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

Present state for cross attention key, with shape `[batchSize, headCount, keyValueSequenceLength, headSize]` or present state for self attention with shape `[batchSize, headCount, pastSequenceLength + keyValueSequenceLength, headSize]`. It contains either the content of the key tensor or the content of the concatenated `PastKey + Key` tensor to pass to the next iteration.

OutputPresentValueTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

Present state for cross attention value, with shape `[batchSize, headCount, keyValueSequenceLength, headSize]` or present state for self attention with shape `[batchSize, headCount, pastSequenceLength + keyValueSequenceLength, headSize]`. It contains either the content of the value tensor or the content of the concatenated `PastValue + Value` tensor to pass to the next iteration.

Scale

Type: **FLOAT**

Scale to multiply the result of the QxK GEMM operation, but before the Softmax operation. This value is usually $1/\sqrt{\text{headSize}}$.

`MaskFilterValue`

Type: **FLOAT**

Value that gets added to the result of QxK GEMM operation to the positions that the mask defined as padding elements. This value should be a very large negative number (usually -10000.0f).

`HeadCount`

Type: **UINT**

Number of attention heads.

`MaskType`

Type: **DML_MULTIHEAD_ATTENTION_MASK_TYPE**

Describes the behavior of *MaskTensor*.

DML_MULTIHEAD_ATTENTION_MASK_TYPE_BOOLEAN. When the mask contains a value of 0, *MaskFilterValue* gets added; but when it contains a value of 1, nothing gets added.

DML_MULTIHEAD_ATTENTION_MASK_TYPE_KEY_SEQUENCE_LENGTH. The mask, of shape `[1, batchSize]`, contains the sequence lengths of the unpadded area for each batch, and all elements after the sequence length get their value set to *MaskFilterValue*.

DML_MULTIHEAD_ATTENTION_MASK_TYPE_KEY_SEQUENCE_END_START. The mask, of shape `[2, batchSize]`, contains the end (exclusive) and start (inclusive) indices of the unpadded area, and all elements outside of the area get their value set to *MaskFilterValue*.

DML_MULTIHEAD_ATTENTION_MASK_TYPE_KEY_SEQUENCE_LENGTH_START-END. The mask, of shape `[batchSize * 3 + 2]`, has the following values: `[keyLength[0], ..., keyLength[batchSize - 1], queryStart[0], ..., queryStart[batchSize - 1], queryEnd[batchSize - 1], keyStart[0], ..., keyStart[batchSize - 1], keyEnd[batchSize - 1]]`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_6_1`.

Tensor constraints

`BiasTensor`, `KeyTensor`, `OutputPresentKeyTensor`, `OutputPresentValueTensor`, `OutputTensor`, `PastKeyTensor`, `PastValueTensor`, `QueryTensor`, `RelativePositionBiasTensor`, `StackedKeyValueTensor`, `StackedQueryKeyTensor`, `StackedQueryKeyValueTensor`, and `ValueTensor` must have the same `Data Type`.

Tensor support

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
QueryTensor	Optional input	3 to 5	FLOAT32, FLOAT16
KeyTensor	Optional input	3 to 5	FLOAT32, FLOAT16
ValueTensor	Optional input	3 to 5	FLOAT32, FLOAT16
StackedQueryKeyTensor	Optional input	5	FLOAT32, FLOAT16
StackedKeyValueTensor	Optional input	5	FLOAT32, FLOAT16
StackedQueryKeyValueTensor	Optional input	5	FLOAT32, FLOAT16
BiasTensor	Optional input	1 to 5	FLOAT32, FLOAT16
MaskTensor	Optional input	1 to 5	INT32
RelativePositionBiasTensor	Optional input	4 to 5	FLOAT32, FLOAT16
PastKeyTensor	Optional input	4 to 5	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
PastValueTensor	Optional input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	3 to 5	FLOAT32, FLOAT16
OutputPresentKeyTensor	Optional output	4 to 5	FLOAT32, FLOAT16
OutputPresentValueTensor	Optional output	4 to 5	FLOAT32, FLOAT16

Requirements

[+] Expand table

Header	directml.h
--------	------------

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_NONZERO_COORDINATES_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Computes the N-dimensional coordinates of all non-zero elements of the input tensor.

This operator produces an MxN matrix of values, where each row M contains an N-dimensional coordinate of a non-zero value from the input. When using **FLOAT32** or **FLOAT16** inputs, both negative and positive 0 (0.0f and -0.0f) are treated as zero for the purposes of this operator.

The operator requires that the *OutputCoordinatesTensor* has a size large enough to accommodate a worst-case scenario where every element of the input is non-zero. This operator returns the count of non-zero elements via the *OutputCountTensor*, which callers can inspect to determine the number of coordinates written to the *OutputCoordinatesTensor*.

Syntax

C++

```
struct DML_NONZERO_COORDINATES_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputCountTensor;
    const DML_TENSOR_DESC *OutputCoordinatesTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

An input tensor.

OutputCountTensor

Type: **const DML_TENSOR_DESC***

An output tensor that holds the count of non-zero elements in the input tensor. This tensor must be a scalar—that is, the Sizes of this tensor must all be 1. The type of this

tensor must be **UINT32**.

`OutputCoordinatesTensor`

Type: **const DML_TENSOR_DESC***

An output tensor that holds the N-dimensional coordinates of the input elements which are non-zero.

This tensor must have *Sizes* of `{1,1,M,N}` (if *DimensionCount* is 4), or `{1,1,1,M,N}` (if *DimensionCount* is 5), where M is the total number of elements in the *InputTensor*, and N is greater or equal to the *effective rank* of *InputTensor*, up to the *DimensionCount* of the input.

The effective rank of a tensor is determined by the *DimensionCount* of that tensor excluding leading dimensions of size 1. For example a tensor with sizes of `{1,2,3,4}` has effective rank 3, as does a tensor with sizes of `{1,1,5,5,5}`. A tensor with sizes `{1,1,1,1}` has effective rank 0.

Consider an *InputTensor* with *Sizes* of `{1,1,12,5}`. This input tensor contains 60 elements, and has an effective rank of 2. In this example all valid sizes of *OutputCoordinatesTensor* are of the form `{1,1,60,N}`, where $N \geq 2$ but no greater than the *DimensionCount* (4 in this example).

The coordinates written into this tensor are guaranteed to be ordered by ascending element index. For example if the input tensor has 3 non-zero values at coordinates `{1,0}`, `{1,2}`, and `{0,5}`, the values written to the *OutputCoordinatesTensor* will be `[[0,5], [1,0], [1,2]]`.

While this tensor requires its dimension M to equal the number of elements in the input tensor, this operator will only write a maximum of *OutputCount* elements to this tensor. The *OutputCount* is returned through the scalar *OutputCountTensor*.

Note

The remaining elements of this tensor beyond the *OutputCount* are undefined once this operator completes. You shouldn't rely on the values of these elements.

Example

```

InputTensor: (Sizes:{1,1,2,4}, DataType:FLOAT32)
[[1.0f, 0.0f, 0.0f, 2.0f],
 [-0.0f, 3.5f, 0.0f, -5.2f]]

OutputCountTensor: (Sizes:{1,1,1,1}, DataType:UINT32)
[4]

OutputCoordinatesTensor: (Sizes:{1,1,8,3}, DataType:UINT32)
[[0, 0, 0],
 [0, 0, 3],
 [0, 1, 1],
 [0, 1, 3],
 [0, 0, 0], //
 [0, 0, 0], // Values in rows >= OutputCountTensor (4 in
 [0, 0, 0], // this case) are left undefined
 [0, 0, 0]] //

```

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor support

`DML_FEATURE_LEVEL_4_0` and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ [D0], [D1], [D2], [D3], [D4], [D5], [D6], D7 }	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputCountTensor	Output	{ [1], [1], [1], [1], [1], [1], [1], 1 }	1 to 8	UINT32
OutputCoordinatesTensor	Output	{ [1], [1], [1], [1], [1], [1], M, N }	2 to 8	UINT32

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ [D0], D1, D2, D3, D4 }	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputCountTensor	Output	{ [1], 1, 1, 1, 1 }	4 to 5	UINT32
OutputCoordinatesTensor	Output	{ [1], 1, 1, M, N }	4 to 5	UINT32

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ONE_HOT_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Produces a tensor filled with *one-hot encoded* values. This operator produces an output tensor where, for all sequences in a chosen axis, all but one element in that sequence is set to *OffValue*, and the remaining single element is set to *OnValue*. A *sequence* refers to one of the sets of elements that exist along the *Axis* dimension of the *OutputTensor*.

The location of the *OnValue* for each sequence and the choice of *OnValue/OffValue* are determined by the *IndicesTensor* and *ValuesTensor*, respectively.

Syntax

C++

```
struct DML_ONE_HOT_OPERATOR_DESC {
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *ValuesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
};
```

Members

IndicesTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the index in elements of the *OnValue*, for each sequence along the *Axis*. Indices are measured relative to the beginning of their sequence (as opposed to the beginning of the tensor). For example, an index of 0 always refers to the first element for all sequences in an axis.

If an index value for a sequence exceeds the number of elements along the *Axis* dimension in the *OutputTensor*, then that index value is ignored, and all elements in that sequence will be set to *OffValue*.

Starting with `DML_FEATURE_LEVEL_3_0`, this operator supports negative index values when using a signed integral type with this tensor. Negative indices are interpreted as being

relative to the end of the sequence. For example, an index of -1 refers to the last element in the sequence.

This tensor must have dimension count and sizes equal to the *OutputTensor*, except for the dimension specified by the *Axis* parameter. The size of the *Axis* dimension must be 1. For example if the *OutputTensor* has sizes of `{2,3,4,5}` and *Axis* is 1, the sizes of the *IndicesTensor* must be `{2,1,4,5}`.

`ValuesTensor`

Type: **const DML_TENSOR_DESC***

This tensor may have any size, so long as it contains at least two elements. The 0th element of this tensor is interpreted as the *OffValue*, and the 1st element along the fastest-changing dimension of size >1 is interpreted as the *OnValue*.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to. This tensor must have dimension count and sizes equal to the *IndicesTensor*, except for the dimension specified by the *Axis* parameter. The size of the *Axis* dimension in this tensor may have any value greater than 0.

`Axis`

Type: **UINT**

The index of the dimension to produce one-hot encoded sequences along. This value must be less than the DimensionCount of the *IndicesTensor*.

Examples

Example 1

```
IndicesTensor: (Sizes:{1,1,3,1}, DataType:UINT32)
[[[0],
 [3],
 [2]]]

ValuesTensor: (Sizes:{1,1,1,2}, DataType:FLOAT32)
[[[0, 1]]]
```

```
Axis: 3

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[1, 0, 0, 0],    // The one-hot encoding is formed across the rows
   [0, 0, 0, 1],
   [0, 0, 1, 0]]]]
```

Example 2. Using a different axis

```
IndicesTensor: (Sizes:{1,1,1,4}, DataType:UINT32)
[[[[0, 2, 1, 0]]]]

ValuesTensor: (Sizes:{1,1,1,2}, DataType:FLOAT32)
[[[[0, 1]]]]

Axis: 2

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[1, 0, 0, 1],    // The one-hot encoding is formed across the columns
   [0, 0, 1, 0],
   [0, 1, 0, 0]]]]
```

Example 3. Using different on/off values

```
IndicesTensor: (Sizes:{1,1,3,1}, DataType:UINT32)
[[[[0],
   [3],
   [2]]]]

ValuesTensor: (Sizes:{1,1,3,1}, DataType:FLOAT32)
[[[[4],    // off value
   [2],    // on value
   [9]]]] // unused

Axis: 3

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[2, 4, 4, 4],
   [4, 4, 4, 2],
   [4, 4, 2, 4]]]]
```

Example 4. Negative and out-of-bounds indices

```

IndicesTensor: (Sizes:{1,1,3,1}, DataType:INT32)
[[[[ -3],
   [100],
   [ 3]]]]

ValuesTensor: (Sizes:{1,1,1,2}, DataType:FLOAT32)
[[[ [0, 1]]]]

Axis: 3

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[0, 1, 0, 0],    // negative indices count from the end
   [0, 0, 0, 0],    // out-of-bounds indices are ignored; all elements are
   set to OffValue
   [0, 0, 0, 1]]]]

```

Remarks

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

- *IndicesTensor*, *OutputTensor*, and *ValuesTensor* must have the same *DimensionCount*.
- *OutputTensor* and *ValuesTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
ValuesTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
ValuesTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
IndicesTensor	Input	4	UINT32
ValuesTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
IndicesTensor	Input	4	UINT32
ValuesTensor	Input	4	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

A generic container for an operator description. You construct DirectML operators using the parameters specified in this struct. See [IDMLDevice::CreateOperator](#) for additional details.

Syntax

C++

```
struct DML_OPERATOR_DESC {
    DML_OPERATOR_TYPE Type;
    const void        *Desc;
};
```

Members

Type

Type: [DML_OPERATOR_TYPE](#)

The type of the operator description. See [DML_OPERATOR_TYPE](#) for the available types.

Desc

Type: `const void*`

A pointer to the operator description. The type of the pointed-to struct must match the value specified in *Type*.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_OPERATOR_GRAPH_NODE_DESC structure (directml.h)

Article 02/22/2024

Describes a node within a graph of DirectML operators defined by [DML_GRAPH_DESC](#) and passed to [IDMLDevice1::CompileGraph](#).

The behavior of this node is defined by a DirectML operator.

Syntax

C++

```
struct DML_OPERATOR_GRAPH_NODE_DESC {
    IDMLOperator *Operator;
    const char    *Name;
};
```

Members

Operator

Type: [IDMLOperator*](#)

A DirectML operator defining the behavior of the node.

Name

Type: _Field_z_ _Maybenull_ const char*

An optional name for the graph connection. If provided, this might be used within certain error messages emitted by the DirectML debug layer.

Availability

This API was introduced in DirectML version [1.1.0](#).

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method](#)
- [DML_GRAPH_DESC structure](#)
- [DML_GRAPH_NODE_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_OUTPUT_GRAPH_EDGE_DESC structure (directml.h)

Article 10/20/2021

Describes a connection within a graph of DirectML operators defined by [DML_GRAPH_DESC](#) and passed to [IDMLDevice1::CompileGraph](#). This structure is used to define a connection from an output of an internal node to a graph output.

Syntax

C++

```
struct DML_OUTPUT_GRAPH_EDGE_DESC {
    UINT      FromNodeIndex;
    UINT      FromNodeOutputIndex;
    UINT      GraphOutputIndex;
    const char *Name;
};
```

Members

FromNodeIndex

FromNodeOutputIndex

GraphOutputIndex

Type: [UINT](#)

The index of the graph output to which a connection from an internal node output is specified.

Name

Type: `_Field_z_ _Maybenull_ const char*`

An optional name for the graph connection. If provided, this might be used within certain error messages emitted by the DirectML debug layer.

Availability

This API was introduced in DirectML version 1.1.0.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method](#)
- [DML_GRAPH_DESC structure](#)
- [DML_GRAPH_EDGE_DESC structure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_PADDING_OPERATOR_DESC structure (directml.h)

Article 08/19/2022

Inflates the input tensor with constant or mirrored values on the edges, and writes the result to the output.

Syntax

C++

```
struct DML_PADDING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_PADDING_MODE      PaddingMode;
    FLOAT                 PaddingValue;
    UINT                  DimensionCount;
    const UINT            *StartPadding;
    const UINT            *EndPadding;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the input data.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the output data. For each dimension `i`, `OutputTensor.Sizes[i] = InputTensor.Sizes[i] + StartPadding[i] + EndPadding[i]`.

`PaddingMode`

Type: `DML_PADDING_MODE`

The padding mode to use when filling the padding regions.

- **DML_PADDING_MODE_CONSTANT**. Uses a single constant value defined by `PaddingValue` for all padding values (see [Example 1](#)).
- **DML_PADDING_MODE_EDGE**. For each dimension, use the edge values of that dimension for all padding values (see [Example 2](#)).
- **DML_PADDING_MODE_REFLECTION**. Mirror the values of the tensor as if we folded it right on the edges, which means that edges are not mirrored. Note that `StartPadding[i] >= InputTensor.Sizes[i]`, and `EndPadding[i] >= InputTensor.Sizes[i]` is valid, which means that we can mirror new padding regions periodically by folding them over previous padding regions (see [Example 3](#)).
- **DML_PADDING_MODE_SYMMETRIC**. Similar to **DML_PADDING_MODE_REFLECTION**, but edges are also mirrored. Note that `StartPadding[i] > InputTensor.Sizes[i]`, and `EndPadding[i] > InputTensor.Sizes[i]` is valid, which means that we can mirror new padding regions periodically by folding them over previous padding regions (see [Example 4](#)). This mode was introduced in feature level `DML_FEATURE_LEVEL_3_0`.

`PaddingValue`

Type: [FLOAT](#)

The padding value to use when `PaddingMode == DML_PADDING_MODE_CONSTANT`. This value is ignored for other padding modes. Note that if the *DataType* of the tensors is not [DML_TENSOR_DATA_TYPE_FLOAT16](#) or [DML_TENSOR_DATA_TYPE_FLOAT32](#), then the value might be truncated (for example, 10.6 will become 10).

`DimensionCount`

Type: [UINT](#)

The size of the arrays pointed to by `StartPadding` and `EndPadding`. This value must be the same value as the dimension count of *InputTensor* and *OutputTensor*.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The sizes of the padding regions to add at the beginning of each dimension. For each dimension `i`, `StartPadding[i] = OutputTensor.Sizes[i] - InputTensor.Sizes[i] - EndPadding[i]`.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The sizes of the padding regions to add at the end of each dimension. For each dimension `i`, `EndPadding[i] = OutputTensor.Sizes[i] - InputTensor.Sizes[i] - StartPadding[i]`.

Examples

Example 1

```
PaddingMode: DML_PADDING_MODE_CONSTANT
PaddingValue: 9
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [1, 2, 3, 4],
 [5, 6, 7, 8]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[9, 9, 9, 9, 9, 9, 9, 9, 9]
 [9, 9, 1, 2, 3, 4, 9, 9, 9],
 [9, 9, 5, 6, 7, 8, 9, 9, 9],
 [9, 9, 1, 2, 3, 4, 9, 9, 9],
 [9, 9, 5, 6, 7, 8, 9, 9, 9],
 [9, 9, 9, 9, 9, 9, 9, 9, 9],
 [9, 9, 9, 9, 9, 9, 9, 9, 9],
 [9, 9, 9, 9, 9, 9, 9, 9, 9]]]]
```

Example 2

```
PaddingMode: DML_PADDING_MODE_EDGE
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [1, 2, 3, 4],
 [5, 6, 7, 8]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[1, 1, 1, 2, 3, 4, 4, 4, 4]]]
```

```
[1, 1, 1, 2, 3, 4, 4, 4, 4, 4],  
[5, 5, 5, 6, 7, 8, 8, 8, 8, 8],  
[1, 1, 1, 2, 3, 4, 4, 4, 4, 4],  
[5, 5, 5, 6, 7, 8, 8, 8, 8, 8],  
[5, 5, 5, 6, 7, 8, 8, 8, 8, 8],  
[5, 5, 5, 6, 7, 8, 8, 8, 8, 8],  
[5, 5, 5, 6, 7, 8, 8, 8, 8, 8]]]
```

Example 3

```
PaddingMode: DML_PADDING_MODE_REFLECTION
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[[1, 2, 3, 4],
   [5, 6, 7, 8],
   [1, 2, 3, 4],
   [5, 6, 7, 8]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[[7, 6, 5, 6, 7, 8, 7, 6, 5, 6]
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2],
   [7, 6, 5, 6, 7, 8, 7, 6, 5, 6],
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2],
   [7, 6, 5, 6, 7, 8, 7, 6, 5, 6],
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2],
   [7, 6, 5, 6, 7, 8, 7, 6, 5, 6],
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2]]]
```

Example 4 (starting from DML_FEATURE_LEVEL_3_0)

```
PaddingMode: DML_PADDING_MODE_SYMMETRIC
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[ [1, 2, 3, 4],
      [5, 6, 7, 8],
      [1, 2, 3, 4],
      [5, 6, 7, 8]]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[ [2, 1, 1, 2, 3, 4, 4, 3, 2, 1]
      [2, 1, 1, 2, 3, 4, 4, 3, 2, 1],
```

```
[6, 5, 5, 6, 7, 8, 8, 7, 6, 5],  
[2, 1, 1, 2, 3, 4, 4, 3, 2, 1],  
[6, 5, 5, 6, 7, 8, 8, 7, 6, 5],  
[6, 5, 5, 6, 7, 8, 8, 7, 6, 5],  
[2, 1, 1, 2, 3, 4, 4, 3, 2, 1],  
[6, 5, 5, 6, 7, 8, 8, 7, 6, 5]]]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_1 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_PADDING1_OPERATOR_DESC structure (directml.h)

Article 08/19/2022

Inflates the input tensor with constant or mirrored values on the edges, and writes the result to the output.

Syntax

C++

```
struct DML_PADDING1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_PADDING_MODE      PaddingMode;
    DML_TENSOR_DATA_TYPE PaddingValueDataType;
    DML_SCALAR_UNION     PaddingValue;
    UINT                 DimensionCount;
    const UINT            *StartPadding;
    const UINT            *EndPadding;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the input data.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the output data. For each dimension i , `OutputTensor.Sizes[i] = InputTensor.Sizes[i] + StartPadding[i] + EndPadding[i]`.

`PaddingMode`

Type: `DML_PADDING_MODE*`

The padding mode to use when filling the padding regions.

- **DML_PADDING_MODE_CONSTANT**. Uses a single constant value defined by *PaddingValue* for all padding values (see [Example 1](#)).
- **DML_PADDING_MODE_EDGE**. For each dimension, use the edge values of that dimension for all padding values (see [Example 2](#)).
- **DML_PADDING_MODE_REFLECTION**. Mirror the values of the tensor as if we folded it right on the edges, which means that edges are not mirrored. Note that `StartPadding[i] >= InputTensor.Sizes[i]`, and `EndPadding[i] >= InputTensor.Sizes[i]` is valid, which means that we can mirror new padding regions periodically by folding them over previous padding regions (see [Example 3](#)).
- **DML_PADDING_MODE_SYMMETRIC**. Similar to **DML_PADDING_MODE_REFLECTION**, but edges are also mirrored. Note that `StartPadding[i] > InputTensor.Sizes[i]`, and `EndPadding[i] > InputTensor.Sizes[i]` is valid, which means that we can mirror new padding regions periodically by folding them over previous padding regions (see [Example 4](#)). This mode was introduced in feature level `DML_FEATURE_LEVEL_3_0`.

`PaddingValueType`

Type: [DML_TENSOR_DATA_TYPE](#)

The data type of the *PaddingValue* member, which must match *OutputTensor.DataType*.

`PaddingValue`

Type: [DML_SCALAR_UNION](#)

The padding value to use when `PaddingMode == DML_PADDING_MODE_CONSTANT`, with *PaddingValueType* determining how to interpret the field. This value is ignored for other padding modes.

`DimensionCount`

Type: [UINT](#)

The size of the arrays pointed to by *StartPadding* and *EndPadding*. This value must be the same value as the dimension count of *InputTensor* and *OutputTensor*.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The sizes of the padding regions to add at the beginning of each dimension. For each dimension *i*, `StartPadding[i] = OutputTensor.Sizes[i] - InputTensor.Sizes[i] -`

```
EndPadding[i].
```

```
EndPadding
```

Type: _Field_size_(DimensionCount) const **UINT***

The sizes of the padding regions to add at the end of each dimension. For each dimension i , `EndPadding[i] = OutputTensor.Sizes[i] - InputTensor.Sizes[i] - StartPadding[i]`.

Remarks

Examples

Example 1

```
PaddingMode: DML_PADDING_MODE_CONSTANT
PaddingValueType: FLOAT32
PaddingValue: 9
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[[1, 2, 3, 4],
   [5, 6, 7, 8],
   [1, 2, 3, 4],
   [5, 6, 7, 8]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[[9, 9, 9, 9, 9, 9, 9, 9, 9]
   [9, 9, 1, 2, 3, 4, 9, 9, 9],
   [9, 9, 5, 6, 7, 8, 9, 9, 9],
   [9, 9, 1, 2, 3, 4, 9, 9, 9],
   [9, 9, 5, 6, 7, 8, 9, 9, 9],
   [9, 9, 9, 9, 9, 9, 9, 9, 9],
   [9, 9, 9, 9, 9, 9, 9, 9, 9],
   [9, 9, 9, 9, 9, 9, 9, 9, 9]]]]
```

Example 2

```
PaddingMode: DML_PADDING_MODE_EDGE
PaddingValueDataType: FLOAT32
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[[1, 2, 3, 4],
   [5, 6, 7, 8],
   [1, 2, 3, 4],
   [5, 6, 7, 8]]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[[1, 1, 2, 3, 4, 4, 4, 4, 4]
   [1, 1, 1, 2, 3, 4, 4, 4, 4],
   [5, 5, 5, 6, 7, 8, 8, 8, 8],
   [1, 1, 1, 2, 3, 4, 4, 4, 4],
   [5, 5, 5, 6, 7, 8, 8, 8, 8],
   [5, 5, 5, 6, 7, 8, 8, 8, 8],
   [5, 5, 5, 6, 7, 8, 8, 8, 8],
   [5, 5, 5, 6, 7, 8, 8, 8, 8]]]]
```

Example 3

```
PaddingMode: DML_PADDING_MODE_REFLECTION
PaddingValueDataType: FLOAT32
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[[1, 2, 3, 4],
   [5, 6, 7, 8],
   [1, 2, 3, 4],
   [5, 6, 7, 8]]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[[7, 6, 5, 6, 7, 8, 7, 6, 5, 6]
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2],
   [7, 6, 5, 6, 7, 8, 7, 6, 5, 6],
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2],
   [7, 6, 5, 6, 7, 8, 7, 6, 5, 6],
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2],
   [7, 6, 5, 6, 7, 8, 7, 6, 5, 6],
   [3, 2, 1, 2, 3, 4, 3, 2, 1, 2]]]]
```

Example 4 (starting from DML_FEATURE_LEVEL_3_0)

```

PaddingMode: DML_PADDING_MODE_SYMMETRIC
PaddingValueDataType: FLOAT32
StartPadding: {0, 0, 1, 2}
EndPadding: {0, 0, 3, 4}

InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [1, 2, 3, 4],
 [5, 6, 7, 8]]]

OutputTensor: (Sizes:{1, 1, 8, 10}, DataType:FLOAT32)
[[[2, 1, 1, 2, 3, 4, 4, 3, 2, 1]
 [2, 1, 1, 2, 3, 4, 4, 3, 2, 1],
 [6, 5, 5, 6, 7, 8, 8, 7, 6, 5],
 [2, 1, 1, 2, 3, 4, 4, 3, 2, 1],
 [6, 5, 5, 6, 7, 8, 8, 7, 6, 5],
 [6, 5, 5, 6, 7, 8, 8, 7, 6, 5],
 [2, 1, 1, 2, 3, 4, 4, 3, 2, 1],
 [6, 5, 5, 6, 7, 8, 8, 7, 6, 5]]]

```

Availability

This operator was introduced in **DML_FEATURE_LEVEL_5_0**.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

Requirements

Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_QUANTIZED_LINEAR_CONVOLUTION_OPERATOR_DESC structure (directml.h)

Article 08/21/2024

Performs a convolution of the *FilterTensor* with the *InputTensor*. This operator performs forward convolution on quantized data. This operator is mathematically equivalent to dequantizing the inputs, convolving, and then quantizing the output.

The quantize linear functions used by this operator are the linear quantization functions

Dequantize function

```
f(Input, Scale, ZeroPoint) = (Input - ZeroPoint) * Scale
```

Quantize function

```
f(Input, Scale, ZeroPoint) = clamp(round(Input / Scale) + ZeroPoint, Min, Max)
```

Syntax

C++

```
struct DML_QUANTIZED_LINEAR_CONVOLUTION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *InputScaleTensor;
    const DML_TENSOR_DESC *InputZeroPointTensor;
    const DML_TENSOR_DESC *FilterTensor;
    const DML_TENSOR_DESC *FilterScaleTensor;
    const DML_TENSOR_DESC *FilterZeroPointTensor;
    const DML_TENSOR_DESC *BiasTensor;
    const DML_TENSOR_DESC *OutputScaleTensor;
    const DML_TENSOR_DESC *OutputZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT DimensionCount;
    const UINT *Strides;
```

```
const UINT          *Dilations;
const UINT          *StartPadding;
const UINT          *EndPadding;
UINT               GroupCount;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input data. The expected dimensions of the *InputTensor* are {
InputBatchCount, InputChannelCount, InputHeight, InputWidth }.

InputScaleTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input scale data. The expected dimensions of the
InputScaleTensor are { 1, 1, 1, 1 }. This scale value is used for dequantizing the
input values.

ⓘ Note

A scale value of 0 results in undefined behavior.

InputZeroPointTensor

Type: _Maybenull_ **const DML_TENSOR_DESC***

An optional tensor containing the input zero point data. The expected dimensions of the
InputZeroPointTensor are { 1, 1, 1, 1 }. This zero point value is used for dequantizing
the input values.

FilterTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the filter data. The expected dimensions of the *FilterTensor* are {
FilterBatchCount, FilterChannelCount, FilterHeight, FilterWidth }.

FilterScaleTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the filter scale data. The expected dimensions of the `FilterScaleTensor` are `{ 1, 1, 1, 1 }` if per tensor quantization is required, or `{ 1, OutputChannelCount, 1, 1 }` if per channel quantization is required. This scale value is used for dequantizing the filter values.

ⓘ Note

A scale value of 0 results in undefined behavior.

`FilterZeroPointTensor`

Type: **_Maybenull_ const DML_TENSOR_DESC***

An optional tensor containing the filter zero point data. The expected dimensions of the `FilterZeroPointTensor` are `{ 1, 1, 1, 1 }` if per tensor quantization is required, or `{ 1, OutputChannelCount, 1, 1 }` if per channel quantization is required. This zero point value is used for dequantizing the filter values.

`BiasTensor`

Type: **_Maybenull_ const DML_TENSOR_DESC***

A tensor containing the bias data. The bias tensor is a tensor containing data which is broadcasted across the output tensor at the end of the convolution which is added to the result. The expected dimensions of the `BiasTensor` are `{ 1, OutputChannelCount, 1, 1 }` for 4D.

`OutputScaleTensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the output scale data. The expected dimensions of the `OutputScaleTensor` are `{ 1, 1, 1, 1 }`. This input scale value is used for quantizing the convolution output values.

ⓘ Note

A scale value of 0 results in undefined behavior.

`OutputZeroPointTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the filter zero point data. The expected dimensions of the `OutputZeroPointTensor` are `{ 1, 1, 1, 1 }`. This input zero point value is used for quantizing the convolution the output values.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor to write the results to. The expected dimensions of the `OutputTensor` are `{ OutputBatchCount, OutputChannelCount, OutputHeight, OutputWidth }`.

`DimensionCount`

Type: `UINT`

The number of spatial dimensions for the convolution operation. Spatial dimensions are the lower dimensions of the convolution filter tensor `FilterTensor`. This value also determines the size of the `Strides`, `Dilations`, `StartPadding`, and `EndPadding` arrays. Only a value of 2 is supported.

`Strides`

Type: `_Field_size_(DimensionCount) const UINT*`

The strides of the convolution operation. These strides are applied to the convolution filter. They are separate from the tensor strides included in `DML_TENSOR_DESC`.

`Dilations`

Type: `_Field_size_(DimensionCount) const UINT*`

The Dilations of the convolution operation. Dilations are strides applied to the elements of the filter kernel. This has the effect of simulating a larger filter kernel by padding the internal filter kernel elements with zeros.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The padding values to be applied to the beginning of each spatial dimension of the filter and input tensor of the convolution operation.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

The padding values to be applied to the end of each spatial dimension of the filter and input tensor of the convolution operation.

GroupCount

Type: **UINT**

The number of groups which to divide the convolution operation into. *GroupCount* can be used to achieve depth-wise convolution by setting the *GroupCount* equal to the input channel count. This divides the convolution up into a separate convolution per input channel.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_1](#).

Tensor constraints

- *BiasTensor*, *FilterTensor*, *InputTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *OutputTensor* and *OutputZeroPointTensor* must have the same *DataType*.
- *InputTensor* and *InputZeroPointTensor* must have the same *DataType*.
- *FilterTensor* and *FilterZeroPointTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_2 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	3 to 4	INT8, UINT8
InputScaleTensor	Input	1 to 4	FLOAT32
InputZeroPointTensor	Optional input	1 to 4	INT8, UINT8
FilterTensor	Input	3 to 4	INT8, UINT8
FilterScaleTensor	Input	1 to 4	FLOAT32

Tensor	Kind	Supported dimension counts	Supported data types
FilterZeroPointTensor	Optional input	1 to 4	INT8, UINT8
BiasTensor	Optional input	3 to 4	INT32
OutputScaleTensor	Input	1 to 4	FLOAT32
OutputZeroPointTensor	Optional input	1 to 4	INT8, UINT8
OutputTensor	Output	3 to 4	INT8, UINT8

DML_FEATURE_LEVEL_4_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	3 to 4	INT8, UINT8
InputScaleTensor	Input	1 to 4	FLOAT32
InputZeroPointTensor	Optional input	1 to 4	INT8, UINT8
FilterTensor	Input	3 to 4	INT8, UINT8
FilterScaleTensor	Input	3 to 4	FLOAT32
FilterZeroPointTensor	Optional input	1 to 4	INT8, UINT8
BiasTensor	Optional input	3 to 4	INT32
OutputScaleTensor	Input	1 to 4	FLOAT32
OutputZeroPointTensor	Optional input	1 to 4	INT8, UINT8
OutputTensor	Output	3 to 4	INT8, UINT8

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	INT8, UINT8
InputScaleTensor	Input	4	FLOAT32
InputZeroPointTensor	Optional input	4	INT8, UINT8
FilterTensor	Input	4	INT8, UINT8
FilterScaleTensor	Input	4	FLOAT32
FilterZeroPointTensor	Optional input	4	INT8, UINT8
BiasTensor	Optional input	4	INT32
OutputScaleTensor	Input	4	FLOAT32
OutputZeroPointTensor	Optional input	4	INT8, UINT8
OutputTensor	Output	4	INT8, UINT8

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)[No](#)[Provide product feedback](#) | Get help at Microsoft Q&A

DML_QUANTIZED_LINEAR_MATRIX_MULTIPLY_OPERATOR_DESC structure (directml.h)

Article 08/22/2024

Performs a matrix multiplication function on quantized data. This operator is mathematically equivalent to dequantizing the inputs, then performing matrix multiply, and then quantizing the output.

This operator requires the matrix multiply input tensors to be 4D which are formatted as `{ BatchCount, ChannelCount, Height, Width }`. The matrix multiply operator will perform `BatchCount * ChannelCount` number of independent matrix multiplications.

For example, if `ATensor` has `Sizes` of `{ BatchCount, ChannelCount, M, K }`, and `BTensor` has `Sizes` of `{ BatchCount, ChannelCount, K, N }`, and `OutputTensor` has `Sizes` of `{ BatchCount, ChannelCount, M, N }`, then the matrix multiply operator will perform `BatchCount * ChannelCount` independent matrix multiplications of dimensions $\{M,K\} \times \{K,N\} = \{M,N\}$.

Dequantize function

```
f(Input, Scale, ZeroPoint) = (Input - ZeroPoint) * Scale
```

Quantize function

```
f(Input, Scale, ZeroPoint) = clamp(round(Input / Scale) + ZeroPoint, Min, Max)
```

Syntax

C++

```
struct DML_QUANTIZED_LINEAR_MATRIX_MULTIPLY_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
```

```
    const DML_TENSOR_DESC *AScaleTensor;
    const DML_TENSOR_DESC *AZeroPointTensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *BScaleTensor;
    const DML_TENSOR_DESC *BZeroPointTensor;
    const DML_TENSOR_DESC *OutputScaleTensor;
    const DML_TENSOR_DESC *OutputZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
};


```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A tensor containing the A data. This tensor's dimensions should be { BatchCount, ChannelCount, M, K }.

AScaleTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the ATensor scale data. The expected dimensions of the AScaleTensor are { 1, 1, 1, 1 } if per tensor quantization is required, or { 1, 1, M, 1 } if per row quantization is required. These scale values are used for dequantizing the A values.

ⓘ Note

A scale value of 0 results in undefined behavior.

AZeroPointTensor

Type: _Maybenull_ **const DML_TENSOR_DESC***

An optional tensor containing the ATensor zero point data. The expected dimensions of the AZeroPointTensor are { 1, 1, 1, 1 } if per tensor quantization is required, or { 1, 1, M, 1 } if per row quantization is required. These zero point values are used for dequantizing the ATensor values.

BTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the *B* data. This tensor's dimensions should be `{ BatchCount, ChannelCount, K, N }`.

`BScaleTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the *BTensor* scale data. The expected dimensions of the `BScaleTensor` are `{ 1, 1, 1, 1 }` if per tensor quantization is required, or `{ 1, 1, 1, N }` if per column quantization is required. These scale values are used for dequantizing the *BTensor* values.

Note

A scale value of 0 results in undefined behavior.

`BZeroPointTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the *BTensor* zero point data. The expected dimensions of the `BZeroPointTensor` are `{ 1, 1, 1, 1 }` if per tensor quantization is required, or `{ 1, 1, 1, N }` if per column quantization is required. These zero point values are used for dequantizing the *BTensor* values.

`OutputScaleTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the *OutputTensor* scale data. The expected dimensions of the `OutputScaleTensor` are `{ 1, 1, 1, 1 }` if per-tensor quantization is required, or `{ 1, 1, M, 1 }` if per-row quantization is required. This scale value is used for dequantizing the *OutputTensor* values.

Note

A scale value of 0 results in undefined behavior.

`OutputZeroPointTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the *OutputTensor* zero point data. The expected dimensions of the *OutputZeroPointTensor* are `{ 1, 1, 1, 1 }` if per-tensor quantization is required, or `{ 1, 1, M, 1 }` if per-row quantization is required. This zero point value is used for dequantizing the *OutputTensor* values.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A tensor to write the results to. This tensor's dimensions are `{ BatchCount, ChannelCount, M, N }`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- *AScaleTensor*, *AZeroPointTensor*, *BScaleTensor*, *BZeroPointTensor*, *OutputScaleTensor*, and *OutputZeroPointTensor* must have the same *DimensionCount*.
- *ATensor*, *BTensor*, and *OutputTensor* must have the same *DimensionCount*.
- *BTensor* and *BZeroPointTensor* must have the same *DataType*.
- *OutputTensor* and *OutputZeroPointTensor* must have the same *DataType*.
- *AScaleTensor*, *AZeroPointTensor*, *BScaleTensor*, *BZeroPointTensor*, *OutputScaleTensor*, and *OutputZeroPointTensor* must have the same *DimensionCount*.
- *ATensor* and *AZeroPointTensor* must have the same *DataType*.

Tensor support

`DML_FEATURE_LEVEL_4_0` and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	2 to 4	INT8, UINT8
AScaleTensor	Input	1 to 4	FLOAT32
AZeroPointTensor	Optional	1 to 4	INT8, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
	input		
BTensor	Input	2 to 4	INT8, UINT8
BScaleTensor	Input	1 to 4	FLOAT32
BZeroPointTensor	Optional input	1 to 4	INT8, UINT8
OutputScaleTensor	Input	1 to 4	FLOAT32
OutputZeroPointTensor	Optional input	1 to 4	INT8, UINT8
OutputTensor	Output	2 to 4	INT8, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
ATensor	Input	4	INT8, UINT8
AScaleTensor	Input	4	FLOAT32
AZeroPointTensor	Optional input	4	INT8, UINT8
BTensor	Input	4	INT8, UINT8
BScaleTensor	Input	4	FLOAT32
BZeroPointTensor	Optional input	4	INT8, UINT8
OutputScaleTensor	Input	4	FLOAT32
OutputZeroPointTensor	Optional input	4	INT8, UINT8
OutputTensor	Output	4	INT8, UINT8

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_RANDOM_GENERATOR_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Fills an output tensor with deterministically-generated, pseudo-random, uniformly-distributed bits. This operator optionally may also output an updated internal generator state, which can be used during subsequent executions of the operator.

This operator is deterministic, and behaves as if it were a pure function—that is, its execution produces no side-effects, and uses no global state. The pseudo-random output produced by this operator is solely dependent on the values supplied in the *InputStateTensor*.

The generators implemented by this operator are not cryptographically secure; therefore, this operator should not be used for encryption, key generation, or other applications which require cryptographically secure random number generation.

Syntax

C++

```
struct DML_RANDOM_GENERATOR_OPERATOR_DESC {
    const DML_TENSOR_DESC      *InputStateTensor;
    const DML_TENSOR_DESC      *OutputTensor;
    const DML_TENSOR_DESC      *OutputStateTensor;
    DML_RANDOM_GENERATOR_TYPE Type;
};
```

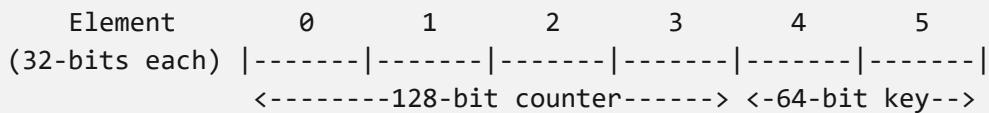
Members

InputStateTensor

Type: **const DML_TENSOR_DESC***

An input tensor containing the internal generator state. The size and format of this input tensor depends on the [DML_RANDOM_GENERATOR_TYPE](#).

For [DML_RANDOM_GENERATOR_TYPE_PHILOX_4X32_10](#), this tensor must be of type [UINT32](#), and have sizes `{1,1,1,6}`. The first four 32-bit values contain a monotonically increasing 128-bit counter. The last two 32-bit values contain the 64-bit key value for the generator.



When initializing the generator's input state for the first time, typically the 128-bit counter (the first four 32-bit `UINT32` values) should be initialized to 0. The key can be arbitrarily chosen; different key values will produce different sequences of numbers. The value for the key is usually generated using a user-provided *seed*.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor which holds the resulting pseudo-random values. This tensor can be of any size.

`OutputStateTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional output tensor THAT holds the updated internal generator state. If supplied, this operator uses the *InputStateTensor* to compute an appropriate updated generator state, and writes the result into the *OutputStateTensor*. Typically, callers would save this result and supply it as the input state on a subsequent execution of this operator.

`Type`

Type: `DML_RANDOM_GENERATOR_TYPE`

One of the values from the `DML_RANDOM_GENERATOR_TYPE` enum, indicating the type of generator to use. Currently the only valid value is `DML_RANDOM_GENERATOR_TYPE_PHILOX_4X32_10`, which generates pseudo-random numbers according to the [Philox 4x32-10 algorithm](#).

Remarks

On each execution of this operator, the 128-bit counter should be incremented. If the *OutputStateTensor* is supplied, THEN this operator increments the counter with the correct value on your behalf, and writes the result into the *OutputStateTensor*. The amount it should be incremented by depends on the number of 32-bit output elements generated, and the type of the generator.

For `DML_RANDOM_GENERATOR_TYPE_PHILOX_4X32_10`, the 128-bit counter should be incremented by `ceil(outputSize / 4)` on each execution, where `outputSize` is the number of elements in the `OutputTensor`.

Consider an example where the value of the 128-bit counter is currently `0x48656c6c'6f46726f'6d536561'74746c65`, and the `OutputTensor`'s size is `{3, 3, 20, 7219}`. After executing this operator once, the counter should be incremented by 324,855 (the number of output elements generated divided by 4, rounded up) resulting in a counter value of `0x48656c6c'6f46726f'6d536561'746f776e`. This updated value should then be supplied as an input for the next execution of this operator.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

`InputStateTensor` and `OutputStateTensor` must have the same `DimensionCount` and `Sizes`.

Tensor support

DML_FEATURE_LEVEL_4_0 and above

[+] Expand table

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputStateTensor	Input	{ [1], [1], [1], [1], [1], [1], [1], 6 }	1 to 8	UINT32
OutputTensor	Output	{ [D0], [D1], [D2], [D3], [D4], [D5], [D6], D7 }	1 to 8	UINT32
OutputStateTensor	Optional output	{ [1], [1], [1], [1], [1], [1], [1], 6 }	1 to 8	UINT32

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputStateTensor	Input	{ 1, 1, 1, 6 }	4	UINT32
OutputTensor	Output	{ D0, D1, D2, D3 }	4	UINT32
OutputStateTensor	Optional output	{ 1, 1, 1, 6 }	4	UINT32

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)[No](#)[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_REDUCE_OPERATOR_DESC structure (directml.h)

Article01/21/2022

Outputs the reduction of elements (sum, product, minimum, and so on) within one or more dimensions of the input tensor.

Each output element is the result of applying a reduction function on a subset of the input tensor. A reduction function, such as *sum*, maps N input elements to a single output element. The input elements involved in each reduction are determined by the provided input axes: N is equal to the product of the sizes of the reduced axes. If all input axes are specified, then the operator performs a reduction on the entire input tensor and produces a single output element.

Syntax

C++

```
struct DML_REDUCE_OPERATOR_DESC {
    DML_REDUCE_FUNCTION    Function;
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                  AxisCount;
    const UINT             *Axes;
};
```

Members

Function

Type: [DML_REDUCE_FUNCTION](#)

Specifies the reduction function to apply to the input.

InputTensor

Type: [const DML_TENSOR_DESC*](#)

The tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the results to. Each output element is the result of a reduction on a subset of elements from the *InputTensor*.

- *DimensionCount* must match *InputTensor.DimensionCount* (the rank of the input tensor is preserved).
- *Sizes* must match *InputTensor.Sizes*, except for dimensions included in the reduced *Axes*, which must be size 1.

AxisCount

Type: **UINT**

The number of axes to reduce. This field determines the size of the *Axes* array.

Axes

Type: **_Field_size_(AxisCount) const UINT***

The axes along which to reduce. Values must be in the range `[0, InputTensor.DimensionCount - 1]`.

Examples

The following examples all use this same two-dimensional input tensor.

```
InputTensor: (Sizes:{3, 3}, DataType:FLOAT32)
[[1, 2, 3],
 [3, 0, 4],
 [2, 4, 2]]
```

Example 1. Applying *sum* to columns

```
Function: DML_REDUCE_FUNCTION_SUM
AxisCount: 1
Axes: {0}
OutputTensor: (Sizes:{1, 3}, DataType:FLOAT32)
[[6, // sum({1, 3, 2})
 6, // sum({2, 0, 4})
 9]] // sum({3, 4, 2})
```

Example 2. Applying *sum* to rows

```
Function: DML_REDUCE_FUNCTION_SUM
AxisCount: 1
Axes: {1}
OutputTensor: (Sizes:{3, 1}, DataType:FLOAT32)
[[6], // sum({1, 2, 3})
 [7], // sum({3, 0, 4})
 [8]] // sum({2, 4, 2})
```

Example 3. Applying *sum* to all axes (the entire tensor)

```
Function: DML_REDUCE_FUNCTION_SUM
AxisCount: 2
Axes: {0, 1}
OutputTensor: (Sizes:{1, 1}, DataType:FLOAT32)
[[21]] // sum({1, 2, 3, 3, 0, 4, 2, 5, 2})
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- The input and output tensor data types must match except when using the ARGMAX and ARGMIN functions, which always output an integral data type.
- The output sizes must be the same as the input sizes except for the reduced axes, which must be 1.

Tensor support according to function

ARGMIN and ARGMAX

`DML_FEATURE_LEVEL_4_1` and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	UINT32

AVERAGE, L2, LOG_SUM, and LOG_SUM_EXP

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

L1 and SUM_SQUARE

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

MIN and MAX

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

MULTIPLY and SUM

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, UINT64, UINT32

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, UINT32

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

Feature level 3_0 introduced these stand-alone operators that supersede the functionality available with [DML_REDUCE_FUNCTION_ARGMAX](#) and [DML_REDUCE_FUNCTION_ARGMIN](#).

- [DML_ARGMAX_OPERATOR_DESC](#)
- [DML_ARGMIN_OPERATOR_DESC](#)

Feedback

Was this page helpful?

[!\[\]\(b40158015f37035960572efaeec60784_img.jpg\) Yes](#)[!\[\]\(91b81f0b3310f005de7cd36408794f27_img.jpg\) No](#)

DML_RESAMPLE_GRAD_OPERATOR_DESC structure (directml.h)

Article 07/20/2022

Computes backpropagation gradients for Resample (see [DML_RESAMPLE1_OPERATOR_DESC](#)).

DML_RESAMPLE1_OPERATOR_DESC rescales arbitrary dimensions of the input tensor using either nearest-neighbor sampling or bilinear interpolation. Given an *InputGradientTensor* with the same sizes as the *output* of an equivalent **DML_RESAMPLE1_OPERATOR_DESC**, this operator produces an *OutputGradientTensor* with the same sizes as the *input* of the **DML_RESAMPLE1_OPERATOR_DESC**.

As an example, consider a **DML_RESAMPLE1_OPERATOR_DESC** that performs a nearest-neighbor scaling of 1.5x in the width, and 0.5x in the height.

```
InputTensor          OutputTensor
[[1, 2],    Resample   [1, 1, 2]
 [3, 4]]      -->
```

Notice how the 0th element of the input tensor (with value 1) contributes to two elements in the output, the 1st element (with value 2) contributes to one element in the output, and the 2nd and 3rd elements (with values 3 and 4) contribute to no elements of the output.

The corresponding **DML_RESAMPLE_GRAD_OPERATOR_DESC** would perform the following.

```
InputGradientTensor      OutputGradientTensor
[4, 5, 6]    ResampleGrad   [[9, 6],
                             -->           [0, 0]]
```

Notice that the values in the *OutputGradientTensor* represent the weighted contributions of that element to the *OutputTensor* during the original **DML_RESAMPLE1_OPERATOR_DESC** operator.

Syntax

C++

```
struct DML_RESAMPLE_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    DML_INTERPOLATION_MODE InterpolationMode;
    UINT DimensionCount;
    const FLOAT *Scales;
    const FLOAT *InputPixelOffsets;
    const FLOAT *OutputPixelOffsets;
};
```

Members

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. Typically this tensor would have the same sizes as the *output* of the corresponding `DML_RESAMPLE1_OPERATOR_DESC` in the forward pass.

`OutputGradientTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor containing the backpropagated gradients. Typically this tensor would have the same sizes as the *input* of the corresponding `DML_RESAMPLE1_OPERATOR_DESC` in the forward pass.

`InterpolationMode`

Type: `DML_INTERPOLATION_MODE`

See *InterpolationMode* in `DML_RESAMPLE1_OPERATOR_DESC`.

`DimensionCount`

Type: `UINT`

The number of elements in the *Scales*, *InputPixelOffsets*, and *OutputPixelOffsets* arrays. This value must equal the *DimensionCount* provided in the *InputGradientTensor* and *OutputGradientTensor*.

`Scales`

Type: _Field_size_(DimensionCount) const **FLOAT***

See *Scales* in [DML_RESAMPLE1_OPERATOR_DESC](#).

`InputPixelOffsets`

Type: _Field_size_(DimensionCount) const **FLOAT***

See *InputPixelOffsets* in [DML_RESAMPLE1_OPERATOR_DESC](#).

`OutputPixelOffsets`

Type: _Field_size_(DimensionCount) const **FLOAT***

See *OutputPixelOffsets* in [DML_RESAMPLE1_OPERATOR_DESC](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_3_0](#).

Tensor constraints

InputGradientTensor and *OutputGradientTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_5_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputGradientTensor	Input	1 to 4	FLOAT32, FLOAT16
OutputGradientTensor	Output	1 to 4	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputGradientTensor	Input	4	FLOAT32, FLOAT16
OutputGradientTensor	Output	4	FLOAT32, FLOAT16

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_RESAMPLE_GRAD1_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Computes backpropagation gradients for [DML_RESAMPLE2_OPERATOR_DESC](#).

DML_RESAMPLE2_OPERATOR_DESC rescales arbitrary dimensions of the input tensor using either nearest-neighbor sampling or bilinear interpolation. Given an *InputGradientTensor* with the same sizes as the *output* of an equivalent **DML_RESAMPLE2_OPERATOR_DESC**, this operator produces an *OutputGradientTensor* with the same sizes as the *input* of the **DML_RESAMPLE2_OPERATOR_DESC**.

As an example, consider a **DML_RESAMPLE2_OPERATOR_DESC** that performs a nearest-neighbor scaling of 1.5x in the width, and 0.5x in the height:

```
InputTensor          OutputTensor
[[1, 2],    Resample    [1, 1, 2]
 [3, 4]]      -->
```

Notice how the 0th element of the input tensor (with value 1) contributes to two elements in the output; the 1st element (with value 2) contributes to one element in the output; and the 2nd and 3rd elements (with values 3 and 4) don't contribute to any elements of the output.

The corresponding **DML_RESAMPLE_GRAD1_OPERATOR_DESC** would perform the following:

```
InputGradientTensor      OutputGradientTensor
[4, 5, 6]    ResampleGrad    [[9, 6],
                               -->           [0, 0]]
```

Notice that the values in the *OutputGradientTensor* represent the weighted contributions of that element to the *OutputTensor* during the original **DML_RESAMPLE2_OPERATOR_DESC** operator.

 **Important**

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) version 1.9 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_RESAMPLE_GRAD1_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputGradientTensor;
    const DML_TENSOR_DESC* OutputGradientTensor;
    DML_INTERPOLATION_MODE InterpolationMode;
    DML_AXIS_DIRECTION RoundingDirection;
    UINT DimensionCount;
    _Field_size_(DimensionCount) const FLOAT* Scales;
    _Field_size_(DimensionCount) const FLOAT* InputPixelOffsets;
    _Field_size_(DimensionCount) const FLOAT* OutputPixelOffsets;
};
```

Members

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. Typically this tensor would have the same sizes as the *output* of the corresponding `DML_RESAMPLE2_OPERATOR_DESC` in the forward pass.

`OutputGradientTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor containing the backpropagated gradients. Typically this tensor would have the same sizes as the *input* of the corresponding `DML_RESAMPLE2_OPERATOR_DESC` in the forward pass.

`InterpolationMode`

Type: `DML_INTERPOLATION_MODE`

See `DML_RESAMPLE2_OPERATOR_DESC::InterpolationMode`.

RoundingDirection

Type: [DML_AXIS_DIRECTION](#)

See [DML_RESAMPLE2_OPERATOR_DESC::RoundingDirection](#).

DimensionCount

Type: [UINT](#)

The number of elements in the *Scales*, *InputPixelOffsets*, and *OutputPixelOffsets* arrays. This value must equal the *DimensionCount* provided in the *InputGradientTensor* and *OutputGradientTensor*.

Scales

Type: `_Field_size_(DimensionCount) const FLOAT*`

See [DML_RESAMPLE2_OPERATOR_DESC::Scales](#).

InputPixelOffsets

Type: `_Field_size_(DimensionCount) const FLOAT*`

See [DML_RESAMPLE2_OPERATOR_DESC::InputPixelOffsets](#).

OutputPixelOffsets

Type: `_Field_size_(DimensionCount) const FLOAT*`

See [DML_RESAMPLE2_OPERATOR_DESC::OutputPixelOffsets](#).

Remarks

This operator is equivalent to [DML_RESAMPLE_GRAD_OPERATOR_DESC](#) when *InterpolationMode* is set to [DML_INTERPOLATION_MODE_LINEAR](#); or when *InterpolationMode* is set to [DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR](#), and *RoundingDirection* to [DML_AXIS_DIRECTION_DECREASING](#), and *OutputPixelOffsets* are adjusted an additional -0.5.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_5_1](#).

Tensor constraints

InputGradientTensor and *OutputGradientTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputGradientTensor	Input	1 to 4	FLOAT32, FLOAT16
OutputGradientTensor	Output	1 to 4	FLOAT32, FLOAT16

Requirements

[] Expand table

Header	directml.h
--------	------------

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_RESAMPLE_OPERATOR_DESC structure (directml.h)

Article01/09/2024

Resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size. You can use a linear or nearest-neighbor interpolation mode. The operator supports interpolation across multiple dimensions, not just 2D. So you can keep the same spatial size, but interpolate across channels or across batches.

Syntax

C++

```
struct DML_RESAMPLE_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_INTERPOLATION_MODE InterpolationMode;
    UINT ScaleCount;
    const FLOAT *Scales;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor containing the input data.

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the output data to.

InterpolationMode

Type: **DML_INTERPOLATION_MODE**

This field determines the kind of interpolation used to choose output pixels.

- **DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR**. Uses the *Nearest Neighbor* algorithm, which chooses the input element nearest to the corresponding pixel center for each output element.
- **DML_INTERPOLATION_MODE_LINEAR**. Uses the *Linear Interpolation* algorithm, which computes the output element by computing the weighted average of the 2 nearest neighboring input elements per dimension. Resampling is supported up to 4 dimensions (quadrilinear), where the weighted average is computed on a total of 16 input elements for each output element.

`ScaleCount`

Type: **UINT**

The number of values in the array `Scales` points to. This value must match the dimension count of *InputTensor* and *OutputTensor*.

`Scales`

Type: `_Field_size_(ScaleCount) const FLOAT*`

The scales to apply when resampling the input, where `scales > 1` scale up the image and `scales < 1` scale down the image for that dimension. Note that the scales don't need to be exactly `OutputSize / InputSize`. If the input after scaling is larger than the output bound, then we crop it to the output size. On the other hand, if the input after scaling is smaller than the output bound, the output edges are clamped.

Remarks

A newer version of this operator, [DML_RESAMPLE1_OPERATOR_DESC](#), was introduced in `DML_FEATURE_LEVEL_2_1`.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_6_2 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 4	FLOAT32, FLOAT16, INT8, UINT8
OutputTensor	Output	1 to 4	FLOAT32, FLOAT16, INT8, UINT8

DML_FEATURE_LEVEL_5_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 4	FLOAT32, FLOAT16
OutputTensor	Output	1 to 4	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_RESAMPLE1_OPERATOR_DESC structure (directml.h)

Article01/09/2024

Resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size. You can use a linear or nearest-neighbor interpolation mode. The operator supports interpolation across multiple dimensions, not just 2D. So you can keep the same spatial size, but interpolate across channels or across batches. The relationship between the input and output coordinates is the following.

```
OutputTensorX = (InputTensorX + InputPixelOffset) * Scale + OutputPixelOffset
```

Syntax

C++

```
struct DML_RESAMPLE1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_INTERPOLATION_MODE InterpolationMode;
    UINT DimensionCount;
    const FLOAT *Scales;
    const FLOAT *InputPixelOffsets;
    const FLOAT *OutputPixelOffsets;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor containing the input data.

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the output data to.

InterpolationMode

Type: **DML_INTERPOLATION_MODE**

This field determines the kind of interpolation used to choose output pixels.

- **DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR**. Uses the *Nearest Neighbor* algorithm, which chooses the input element nearest to the corresponding pixel center for each output element.
- **DML_INTERPOLATION_MODE_LINEAR**. Uses the *Linear Interpolation* algorithm, which computes the output element by computing the weighted average of the 2 nearest neighboring input elements per dimension. Resampling is supported up to 4 dimensions (quadrilinear), where the weighted average is computed on a total of 16 input elements for each output element.

DimensionCount

Type: **UINT**

The number of values in the arrays that *Scales*, *InputPixelOffsets*, and *OutputPixelOffsets* point to. This value must match the dimension count of *InputTensor* and *OutputTensor*.

Scales

Type: `_Field_size_(DimensionCount) const FLOAT*`

The scales to apply when resampling the input, where $\text{scales} > 1$ scale up the image and $\text{scales} < 1$ scale down the image for that dimension. Note that the scales don't need to be exactly `OutputSize / InputSize`. If the input after scaling is larger than the output bound, then we crop it to the output size. On the other hand, if the input after scaling is smaller than the output bound, the output edges are clamped.

InputPixelOffsets

Type: `_Field_size_(DimensionCount) const FLOAT*`

The offsets to apply to the input pixels before resampling. When this value is `0`, the top left corner of the pixel is used instead of its center, which usually won't give the expected result. To resample the image by using the center of the pixels and to get the same behavior as [DML_RESAMPLE_OPERATOR_DESC](#), this value must be `0.5`.

OutputPixelOffsets

Type: `_Field_size_(DimensionCount) const FLOAT*`

The offsets to apply to the output pixels after resampling. When this value is `0`, the top left corner of the pixel is used instead of its center, which usually won't give the

expected result. To resample the image by using the center of the pixels and to get the same behavior as [DML_RESAMPLE_OPERATOR_DESC](#), this value must be `-0.5`.

Remarks

When the *InputPixelOffsets* are set to 0.5, and the *OutputPixelOffsets* are set to -0.5, this operator is equivalent to [DML_RESAMPLE_OPERATOR_DESC](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_6_2 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 4	FLOAT32, FLOAT16, INT8, UINT8
OutputTensor	Output	1 to 4	FLOAT32, FLOAT16, INT8, UINT8

DML_FEATURE_LEVEL_5_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 4	FLOAT32, FLOAT16
OutputTensor	Output	1 to 4	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_2_1 and above

[\[\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[!\[\]\(89e1355b50df05460a77214680aeb413_img.jpg\) Yes](#)[!\[\]\(f81bdac8f9e0b8c851f17165c2be555c_img.jpg\) No](#)[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_RESAMPLE2_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size. You can use a linear or nearest neighbor interpolation mode. The operator supports interpolation across multiple dimensions, not just 2D. So you can keep the same spatial size, but interpolate across channels or across batches. The relationship between the input and output coordinates is the following:

```
OutputTensorX = (InputTensorX + InputPixelOffset) * Scale + OutputPixelOffset
```

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) ↗ version 1.9 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_RESAMPLE2_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* OutputTensor;
    DML_INTERPOLATION_MODE InterpolationMode;
    DML_AXIS_DIRECTION RoundingDirection;
    UINT DimensionCount;
    _Field_size_(DimensionCount) const FLOAT* Scales;
    _Field_size_(DimensionCount) const FLOAT* InputPixelOffsets;
    _Field_size_(DimensionCount) const FLOAT* OutputPixelOffsets;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor containing the input data.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the output data to.

`InterpolationMode`

Type: `DML_INTERPOLATION_MODE`

This field determines the kind of interpolation used to choose output pixels.

- `DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`. Uses the *nearest neighbor* algorithm, which chooses the input element nearest to the corresponding pixel offset for each output element.
- `DML_INTERPOLATION_MODE_LINEAR`. Uses the *linear interpolation* algorithm, which computes the output element by computing the weighted average of the two nearest neighboring input elements per dimension. Resampling is supported up to four dimensions (quadrilinear), where the weighted average is computed on a total of sixteen input elements for each output element.

`RoundingDirection`

Type: `DML_AXIS_DIRECTION`

The direction to round along each axis when mapping fractional coordinates back to input pixels. Since bilinear interpolation interpolates fractional coordinates anyway, this affects only `DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`.

- `DML_AXIS_DIRECTION_INCREASING`. Round fractional coordinates toward increasing axis values (ceil).
- `DML_AXIS_DIRECTION_DECREASING`. Round fractional coordinates toward decreasing axis values (floor).

`DimensionCount`

Type: `UINT`

The number of elements in the *Scales*, *InputPixelOffsets*, and *OutputPixelOffsets* arrays. This value must match the dimension count of *InputTensor* and *OutputTensor*.

`Scales`

Type: `_Field_size_(DimensionCount) const FLOAT*`

The scales to apply when resampling the input, where scales > 1 scale up the image, and scales < 1 scale down the image for that dimension. Note that the scales don't need to be exactly `OutputSize / InputSize`. If the input after scaling is larger than the output bound, then we crop it to the output size. On the other hand, if the input after scaling is smaller than the output bound, then the output edges are clamped.

`InputPixelOffsets`

Type: `_Field_size_(DimensionCount) const FLOAT*`

The offsets to apply to the input pixels before resampling. When this value is 0, the top left corner of the pixel is used instead of its center, which usually won't give the expected result. To resample the image by using the center of the pixels and to get the same behavior as `DML_RESAMPLE_OPERATOR_DESC`, this value must be 0.5.

`OutputPixelOffsets`

Type: `_Field_size_(DimensionCount) const FLOAT*`

The offsets to apply to the output pixels after resampling. When this value is 0, the top left corner of the pixel is used instead of its center, which usually won't give the expected result. To resample the image by using the center of the pixels and to get the same behavior as `DML_RESAMPLE_OPERATOR_DESC`, this value must be -0.5.

Remarks

This operator is equivalent to `DML_RESAMPLE1_OPERATOR_DESC` when *InterpolationMode* is set to `DML_INTERPOLATION_MODE_LINEAR`; or when *InterpolationMode* is set to `DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`, and *RoundingDirection* to `DML_AXIS_DIRECTION_DECREASING`, and *OutputPixelOffsets* are adjusted an additional -0.5.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_5_1`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_6_2 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 4	FLOAT32, FLOAT16, INT8, UINT8
OutputTensor	Output	1 to 4	FLOAT32, FLOAT16, INT8, UINT8

DML_FEATURE_LEVEL_5_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 4	FLOAT32, FLOAT16
OutputTensor	Output	1 to 4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Header	directml.h
--------	------------

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_RESAMPLE3_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Resamples elements from the source to the destination tensor, with antialiasing where appropriate, using the scale factors to compute the destination tensor size. You can use a linear or nearest neighbor interpolation mode. The operator supports interpolation across multiple dimensions, not just 2D. So you can keep the same spatial size, but interpolate across channels or across batches. The relationship between the input and output coordinates is the following:

```
OutputTensorX = (InputTensorX + InputPixelOffset) * Scale + OutputPixelOffset
```

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) version 1.15.1 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_RESAMPLE3_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* OutputTensor;
    DML_INTERPOLATION_MODE InterpolationMode;
    DML_AXIS_DIRECTION RoundingDirection;
    UINT DimensionCount;
    _Field_size_(DimensionCount) const FLOAT* Scales;
    _Field_size_(DimensionCount) const FLOAT* InputPixelOffsets;
    _Field_size_(DimensionCount) const FLOAT* OutputPixelOffsets;
    BOOL Antialiased;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor containing the input data.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the output data to.

`InterpolationMode`

Type: `DML_INTERPOLATION_MODE`

This field determines the kind of interpolation used to choose output pixels.

- `DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`. Uses the *nearest neighbor* algorithm, which chooses the input element nearest to the corresponding pixel offset for each output element.
- `DML_INTERPOLATION_MODE_LINEAR`. Uses the *linear interpolation* algorithm, which computes the output element by computing the weighted average of the two nearest neighboring input elements per dimension. Resampling is supported up to four dimensions (quadrilinear), where the weighted average is computed on a total of sixteen input elements for each output element.

`RoundingDirection`

Type: `DML_AXIS_DIRECTION`

The direction to round along each axis when mapping fractional coordinates back to input pixels. Since bilinear interpolation interpolates fractional coordinates anyway, this affects only `DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`.

- `DML_AXIS_DIRECTION_INCREASING`. Round fractional coordinates toward increasing axis values (ceil).
- `DML_AXIS_DIRECTION_DECREASING`. Round fractional coordinates toward decreasing axis values (floor).

`DimensionCount`

Type: `UINT`

The number of elements in the `Scales`, `InputPixelOffsets`, and `OutputPixelOffsets` arrays. This value must match the dimension count of `InputTensor` and `OutputTensor`.

`Scales`

Type: `_Field_size_(DimensionCount) const FLOAT*`

The scales to apply when resampling the input, where scales > 1 scale up the image, and scales < 1 scale down the image for that dimension. Note that the scales don't need to be exactly `OutputSize / InputSize`. If the input after scaling is larger than the output bound, then we crop it to the output size. On the other hand, if the input after scaling is smaller than the output bound, then the output edges are clamped.

`InputPixelOffsets`

Type: `_Field_size_(DimensionCount) const FLOAT*`

The offsets to apply to the input pixels before resampling. When this value is 0, the top left corner of the pixel is used instead of its center, which usually won't give the expected result. To resample the image by using the center of the pixels and to get the same behavior as `DML_RESAMPLE_OPERATOR_DESC`, this value must be 0.5.

`OutputPixelOffsets`

Type: `_Field_size_(DimensionCount) const FLOAT*`

The offsets to apply to the output pixels after resampling. When this value is 0, the top left corner of the pixel is used instead of its center, which usually won't give the expected result. To resample the image by using the center of the pixels and to get the same behavior as `DML_RESAMPLE_OPERATOR_DESC`, this value must be -0.5.

`Antialiased`

Type: `BOOL`

Antialiasing is achieved by stretching the resampling filter by a factor `max(1, 1 / scale)`, which means that when downsampling, more input pixels contribute to an output pixel.

This occurs when this flag is set to `TRUE`, and assuming the following conditions:

- InterpolationMode is `DML_INTERPOLATION_MODE_LINEAR`
- $1/\text{Scales} > 2$

Then this operator will use an antialiasing filter when downscaling.

Remarks

When *Antialiased* is `false`, this operator is equivalent to [DML_RESAMPLE2_OPERATOR_DESC](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_6_4](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 4	FLOAT32, FLOAT16, INT8, UINT8
OutputTensor	Output	1 to 4	FLOAT32, FLOAT16, INT8, UINT8

Requirements

[+] [Expand table](#)

Header	directml.h
--------	------------

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_REVERSE_SUBSEQUENCES_OPERATOR_DESC structure (directml.h)

Article 01/21/2022

Reverses the elements of one or more *subsequences* of a tensor. The set of subsequences to be reversed is chosen based on the provided axis and sequence lengths.

Syntax

C++

```
struct DML_REVERSE_SUBSEQUENCES_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *SequenceLengthsTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The input tensor containing elements to be reversed.

`SequenceLengthsTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing a value for each subsequence to be reversed, denoting the length in elements of that subsequence. Only the elements within the length of the subsequence are reversed; the remaining elements along that axis are copied to the output unchanged.

This tensor must have dimension count and sizes equal to the `InputTensor`, except for the dimension specified by the `Axis` parameter. The size of the `Axis` dimension must be 1. For example if the `InputTensor` has sizes of `{2,3,4,5}`, and `Axis` is 1, then the sizes of the `SequenceLengthsTensor` must be `{2,1,4,5}`.

If the length of a subsequence exceeds the maximum number of elements along that axis, then this operator behaves as if the value were clamped to the maximum.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The output tensor to write the results to. This tensor must have the same sizes and data type as the *InputTensor*.

`Axis`

Type: `UINT`

The index of the dimension to reverse elements over. This value must be less than the DimensionCount of the *InputTensor*.

Examples

Example 1

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12]]]

SequenceTensor: (Sizes:{1,1,3,1}, DataType:UINT32)
[[[2],
 [4],
 [3]]]

Axis: 3

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[2, 1, 3, 4],
 [8, 7, 6, 5],
 [11, 10, 9, 12]]]
```

Example 2. Reversing along a different axis

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[1, 2, 3, 4],
```

```

[5, 6, 7, 8],
[9, 10, 11, 12]]]

SequenceTensor: (Sizes:{1,1,1,4}, DataType:UINT32)
[[[[2, 3, 1, 0]]]]

Axis: 2

OutputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[5, 10, 3, 4], // Notice that sequence lengths of 1 and 0 are effective
nops
[1, 6, 7, 8],
[9, 2, 11, 12]]]

```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- *InputTensor*, *OutputTensor*, and *SequenceLengthsTensor* must have the same *DimensionCount*.
- *InputTensor* and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
SequenceLengthsTensor	Input	1 to 8	UINT64, UINT32
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_4_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
SequenceLengthsTensor	Input	1 to 8	UINT32
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
SequenceLengthsTensor	Input	4 to 5	UINT32
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
SequenceLengthsTensor	Input	4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

 Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_RNN_OPERATOR_DESC structure (directml.h)

Article05/09/2023

Performs a one-layer simple recurrent neural network (RNN) function on the input. This function is often referred to as the Input Gate. This operator performs this function multiple times in a loop, dictated by the sequence length dimension and the *SequenceLengthsTensor*.

Equation for the forward direction

```
for (t = 0; t < seq_length; t++)  
{  
     $H_t = f(X_t * W_i^T + H_{t-1} * R_i^T + W_{bi} + R_{bi})$   
}
```

Equation for the backward direction

```
for (t = seq_length - 1; t >= 0; t--)  
{  
     $H_t = f(X_t * W_{Bi}^T + H_{t-1} * R_{Bi}^T + W_{Bbi} + R_{Bbi})$   
}
```

Equation legend

- t current RNN layer index.
- $t - 1$ previous RNN layer index. If backwards direction, then it means $t + 1$ index but still the previously calculated t .
- T signifies that a matrix will be transposed before use.
- $*$ signifies a matrix multiplication.
- $+$ signifies a element-wise addition of two matrices.
- H_t output of current RNN index t .
- $f()$ activation function dictated by ActivationDescs.
- X_t Sub-matrix of the InputTensor, which is defined by matrix index $[t, \text{batch_size}, \text{input_size}]$.
- W_i^T Forward sub-matrix defined in WeightTensor, which is defined to be size $[1, \text{batch_size}, \text{input_size}]$. This matrix will be transposed before use.
- W_{Bi}^T Backwards sub-matrix defined in WeightTensor, which is defined to be size $[1, \text{batch_size}, \text{input_size}]$. This matrix will be transposed before use.
- H_{t-1} intermediate tensor that's defined to be the output of the previous layer. For $t = 0$, $H_{\{t-1\}}$ is replaced with HiddenInitTensor. This is defined to be size $[1, \text{batch_size}, \text{hidden_size}]$. A different HiddenInitTensor sub-matrix is used for each direction.
- R_i^T Forward sub-matrix of RecurrenceTensor, which is defined to be size $[1, \text{hidden_size}, \text{hidden_size}]$. This matrix will be transposed before use.
- R_{Bi}^T Backwards sub-matrix of RecurrenceTensor, which is defined to be size $[1, \text{hidden_size}, \text{hidden_size}]$. This matrix will be transposed before use.
- W_{bi}, R_{bi} are the bias tensors of the weight tensor, and recurrence tensor, which are one of the sub-matrices of BiasTensor. This matrix is size $[1, \text{hidden_size}]$ and is broadcasted up to the required size which is $[1, \text{batch_size}, \text{hidden_size}]$.
- W_{Bbi}, R_{Bbi} are the bias tensors for the backwards direction
- i stands for input gate.

Syntax

C++

```
struct DML_RNN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *WeightTensor;
    const DML_TENSOR_DESC *RecurrenceTensor;
    const DML_TENSOR_DESC *BiasTensor;
    const DML_TENSOR_DESC *HiddenInitTensor;
    const DML_TENSOR_DESC *SequenceLengthsTensor;
    const DML_TENSOR_DESC *OutputSequenceTensor;
    const DML_TENSOR_DESC *OutputSingleTensor;
    UINT ActivationDescCount;
    const DML_OPERATOR_DESC *ActivationDescs;
    DML_RECURRENT_NETWORK_DIRECTION Direction;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input data, X. Packed (and potentially padded) into one 4-D tensor with the sizes of { 1, seq_length, batch_size, input_size }. seq_length is the dimension that is mapped to the index, t. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

WeightTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the weight data, W. Concatenation of W_i and W_Bi (if bidirectional). The tensor has sizes { 1, num_directions, hidden_size, input_size }. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

RecurrenceTensor

Type: **const DML_TENSOR_DESC***

An optional tensor containing the recurrence weight data, R. Concatenation of R_i and R_Bi (if bidirectional). This tensor has sizes { 1, num_directions, hidden_size, hidden_size }. The tensor doesn't support the **DML_TENSOR_FLAG OWNED_BY_DML** flag.

BiasTensor

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the bias data for the input gate, B. Concatenation of `{ W_bi, R_bi }`, and `{ W_Bbi, R_Bbi }` (if bidirectional). This tensor has sizes `{ 1, 1, num_directions, 2 * hidden_size }`. If not specified, then defaults to 0. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

`HiddenInitTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing the hidden node initializer tensor, $H_{[t-1]}$ for the first loop index t. If not specified, then defaults to 0. This tensor has sizes `{ 1, num_directions, batch_size, hidden_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

`SequenceLengthsTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor containing an independent seq_length for each element in the batch. If not specified, then all sequences in the batch have length seq_length. This tensor has sizes `{ 1, 1, 1, batch_size }`. The tensor doesn't support the `DML_TENSOR_FLAG OWNED_BY_DML` flag.

`OutputSequenceTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor with which to write the concatenation of all the intermediate layer output values of the hidden nodes, H_t . This tensor has sizes `{ seq_length, num_directions, batch_size, hidden_size }`. seq_length is mapped to the loop index t.

`OutputSingleTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An optional tensor with which to write the final output value of the hidden nodes, H_t . This tensor has sizes `{ 1, num_directions, batch_size, hidden_size }`.

`ActivationDescCount`

Type: `UINT`

This field determines the size of the `ActivationDescs` array.

ActivationDescs

Type: `_Field_size_(ActivationDescCount) const DML_OPERATOR_DESC*`

An array of `DML_OPERATOR_DESC` containing the descriptions of the activation operators, `f()`. The number of activation functions is equal to the number of directions. For forwards and backwards directions there is expected to be 1 activation function. For Bidirectional there are expected to be 2.

Direction

Type: `DML_RECURRENT_NETWORK_DIRECTION`

The direction of the operator: forward, backward, or bidirectional.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

`BiasTensor`, `HiddenInitTensor`, `InputTensor`, `OutputSequenceTensor`, `OutputSingleTensor`, `RecurrenceTensor`, and `WeightTensor` must have the same *DataType*.

Tensor support

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
WeightTensor	Input	4	FLOAT32, FLOAT16
RecurrenceTensor	Input	4	FLOAT32, FLOAT16
BiasTensor	Optional input	4	FLOAT32, FLOAT16
HiddenInitTensor	Optional input	4	FLOAT32, FLOAT16
SequenceLengthsTensor	Optional input	4	UINT32
OutputSequenceTensor	Optional output	4	FLOAT32, FLOAT16

Tensor	Kind	Supported dimension counts	Supported data types
OutputSingleTensor	Optional output	4	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_ROI_ALIGN_GRAD_OPERATOR_DESC structure (directml.h)

Article 07/22/2022

Computes backpropagation gradients for [ROI_ALIGN](#) and [ROI_ALIGN1](#).

Recall that [DML_ROI_ALIGN1_OPERATOR_DESC](#) crops and rescales subregions of an input tensor using either nearest-neighbor sampling or bilinear interpolation. Given an `InputGradientTensor` with the same sizes as the *output* of an equivalent [DML_OPERATOR_ROI_ALIGN1](#), this operator produces an `OutputGradientTensor` with the same sizes as the *input* of [DML_OPERATOR_ROI_ALIGN1](#).

As an example, consider a [DML_OPERATOR_ROI_ALIGN1](#) that performs a *nearest-neighbor* scaling of 1.5x in the width, and 0.5x in the height, for 4 non-overlapping crops of an input with dimensions `[1, 1, 4, 4]`:

```
ROITensor
[[[0, 0, 2, 2],
 [2, 0, 4, 2],
 [0, 2, 2, 4],
 [2, 2, 4, 4]]]

BatchIndicesTensor
[0, 0, 0, 0]

InputTensor
[[[[1, 2, | 3, 4], RoiAlign1   [[[ 1, 1, 2]]],
 [5, 6, | 7, 8],      -->     [[[ 3, 3, 4]]],
 -----           [[[ 9, 9, 10]]],
 [9, 10, | 11, 12],    [[[11, 11, 12]]]
 [13, 14, | 15, 16]]]]
```

Notice how the 0th element of each region contributes to two elements in the output—the 1st element contributes to one element in the output, and the 2nd and 3rd elements contribute to no elements of the output.

The corresponding [DML_OPERATOR_ROI_ALIGN_GRAD](#) would perform the following:

```
InputGradientTensor          OutputGradientTensor
[[[[ 1, 2, 3]]],    ROIAlignGrad [[[ 3, 3, | 9, 6],
 [[ 4, 5, 6]]],      -->     [ 0, 0, | 0, 0],
```

```

[[[ 7,  8,  9]]], -----
[[[10, 11, 12]]]] [15,  9, | 21, 12],
[ 0,  0, |  0,  0]]]

```

In summary, `DML_OPERATOR_ROI_ALIGN_GRAD` behaves similarly to a `DML_OPERATOR_RESAMPLE_GRAD` performed on each batch of the `InputGradientTensor` when regions don't overlap.

For `OutputROIGradientTensor`, the math is a little different, and can be summarized by the following pseudo code (assuming that `MinimumSamplesPerOutput == 1` and `MaximumSamplesPerOutput == 1`):

```

for each region of interest (ROI):
    for each inputGradientCoordinate:
        for each inputCoordinate that contributed to this inputGradient
element:
        topYIndex = floor(inputCoordinate.y)
        bottomYIndex = ceil(inputCoordinate.y)
        leftXIndex = floor(inputCoordinate.x)
        rightXIndex = ceil(inputCoordinate.x)

        yLerp = inputCoordinate.y - topYIndex
        xLerp = inputCoordinate.x - leftXIndex

        topLeft = InputTensor[topYIndex][leftXIndex]
        topRight = InputTensor[topYIndex][rightXIndex]
        bottomLeft = InputTensor[bottomYIndex][leftXIndex]
        bottomRight = InputTensor[bottomYIndex][rightXIndex]

        inputGradientWeight =
InputGradientTensor[inputGradientCoordinate.y][inputGradientCoordinate.x]
        imageGradY = (1 - xLerp) * (bottomLeft - topLeft) + xLerp *
(bottomRight - topRight)
        imageGradX = (1 - yLerp) * (topRight - topLeft) + yLerp *
(bottomRight - bottomLeft)

        imageGradY *= inputGradientWeight
        imageGradX *= inputGradientWeight

        OutputROIGradientTensor[roiIndex][0] += imageGradX * (inputWidth
- inputGradientCoordinate.x)
        OutputROIGradientTensor[roiIndex][1] += imageGradY *
(inputHeight - inputGradientCoordinate.y)
        OutputROIGradientTensor[roiIndex][2] += imageGradX *
inputGradientCoordinate.x
        OutputROIGradientTensor[roiIndex][3] += imageGradY *
inputGradientCoordinate.y

```

`OutputGradientTensor` or `OutputROIGradientTensor` can be omitted if only one is needed; but at least one must be supplied.

Syntax

C++

```
struct DML_ROI_ALIGN_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *ROITensor;
    const DML_TENSOR_DESC *BatchIndicesTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    const DML_TENSOR_DESC *OutputROIGradientTensor;
    DML_REDUCE_FUNCTION ReductionFunction;
    DML_INTERPOLATION_MODE InterpolationMode;
    FLOAT SpatialScaleX;
    FLOAT SpatialScaleY;
    FLOAT InputPixelOffset;
    FLOAT OutputPixelOffset;
    UINT MinimumSamplesPerOutput;
    UINT MaximumSamplesPerOutput;
    BOOL AlignRegionsToCorners;
};
```

Members

`InputTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

A tensor containing the input data from the forward pass with dimensions `{
 BatchCount, ChannelCount, InputHeight, InputWidth }`. This tensor *must* be supplied when `OutputROIGradientTensor` is supplied, or when `ReductionFunction == DML_REDUCE_FUNCTION_MAX`. This is the same tensor that would be supplied to `InputTensor` for `DML_OPERATOR_ROI_ALIGN` or `DML_OPERATOR_ROI_ALIGN1`.

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

`ROITensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the regions of interest (ROI) data—a series of bounding boxes in floating point coordinates that point into the X and Y dimensions of the input tensor. The allowed dimensions of `ROITensor` are `{ NumROIs, 4 }`, `{ 1, NumROIs, 4 }`, or `{ 1, 1, NumROIs, 4 }`. For each ROI, the values will be the coordinates of its top-left and bottom-right corners in the order `[x1, y1, x2, y2]`. Regions can be empty, meaning that all output pixels come from the single input coordinate, and regions can be inverted (for example, `x2` less than `x1`), meaning that the output receives a mirrored/flipped version of the input. These coordinates are first scaled by `SpatialScaleX` and `SpatialScaleY`, but if they are both 1.0 then the region rectangles simply correspond directly to the input tensor coordinates. This is the same tensor that would be supplied to `ROITensor` for `DML_OPERATOR_ROI_ALIGN` or `DML_OPERATOR_ROI_ALIGN1`.

`BatchIndicesTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the batch indices to extract the ROIs from. The allowed dimensions of `BatchIndicesTensor` are `{ NumROIs }`, `{ 1, NumROIs }`, `{ 1, 1, NumROIs }`, or `{ 1, 1, 1, NumROIs }`. Each value is the index of a batch from `InputTensor`. The behavior is undefined if the values are not in the range `[0, BatchCount)`. This is the same tensor that would be supplied to `BatchIndicesTensor` for `DML_OPERATOR_ROI_ALIGN` or `DML_OPERATOR_ROI_ALIGN1`.

`OutputGradientTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An output tensor containing the backpropagated gradients with respect to `InputTensor`. Typically this tensor would have the same sizes as the *input* of the corresponding `DML_OPERATOR_ROI_ALIGN1` in the forward pass. If `OutputROIGradientTensor` is not supplied, then `OutputGradientTensor` *must* be supplied.

`OutputROIGradientTensor`

Type: `_Maybenull_ const DML_TENSOR_DESC*`

An output tensor containing the backpropagated gradients with respect to `ROITensor`. This tensor needs to have the same sizes as `ROITensor`. If `OutputGradientTensor` is not supplied, then `OutputROIGradientTensor` *must* be supplied.

`ReductionFunction`

Type: [DML_REDUCE_FUNCTION](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::ReductionFunction](#).

`InterpolationMode`

Type: [DML_INTERPOLATION_MODE](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::InterpolationMode](#).

`SpatialScaleX`

Type: [FLOAT](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::SpatialScaleX](#).

`SpatialScaleY`

Type: [FLOAT](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::SpatialScaleY](#).

`InputPixelOffset`

Type: [FLOAT](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::InputPixelOffset](#).

`OutputPixelOffset`

Type: [FLOAT](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::OutputPixelOffset](#).

`MinimumSamplesPerOutput`

Type: [UINT](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::MinimumSamplesPerOutput](#).

`MaximumSamplesPerOutput`

Type: [UINT](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::MaximumSamplesPerOutput](#).

`AlignRegionsToCorners`

Type: [BOOL](#)

See [DML_ROI_ALIGN1_OPERATOR_DESC::AlignRegionsToCorners](#).

Remarks

Availability

This operator was introduced in [DML_FEATURE_LEVEL_4_1](#).

Tensor constraints

InputGradientTensor, *InputTensor*, *OutputGradientTensor*, *OutputROIGradientTensor*, and *ROITensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Optional input	4	FLOAT32, FLOAT16
InputGradientTensor	Input	4	FLOAT32, FLOAT16
ROITensor	Input	2 to 4	FLOAT32, FLOAT16
BatchIndicesTensor	Input	1 to 4	UINT32
OutputGradientTensor	Optional output	4	FLOAT32, FLOAT16
OutputROIGradientTensor	Optional output	2 to 4	FLOAT32, FLOAT16

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ROI_ALIGN_OPERATOR_DESC structure (directml.h)

Article01/21/2022

Performs an ROI align operation, as described in the [Mask R-CNN](#) paper. In summary, the operation extracts crops from the input image tensor and resizes them to a common output size specified by the last 2 dimensions of *OutputTensor* using the specified *InterpolationMode*.

Syntax

C++

```
struct DML_ROI_ALIGN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ROITensor;
    const DML_TENSOR_DESC *BatchIndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_REDUCE_FUNCTION ReductionFunction;
    DML_INTERPOLATION_MODE InterpolationMode;
    FLOAT SpatialScaleX;
    FLOAT SpatialScaleY;
    FLOAT OutOfBoundsInputValue;
    UINT MinimumSamplesPerOutput;
    UINT MaximumSamplesPerOutput;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input data with dimensions `{ BatchCount, ChannelCount, InputHeight, InputWidth }`.

ROITensor

Type: **const DML_TENSOR_DESC***

A tensor containing the regions of interest (ROI) data. The allowed dimensions of **ROITensor** are `{ NumROIs, 4 }`, `{ 1, NumROIs, 4 }`, or `{ 1, 1, NumROIs, 4 }`. For each

ROI, the values will be the coordinates of its top-left and bottom-right corners in the order `[x1, y1, x2, y2]`.

BatchIndicesTensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the batch indices to extract the ROIs from. The allowed dimensions of `BatchIndicesTensor` are `{ NumROIs }`, `{ 1, NumROIs }`, `{ 1, 1, NumROIs }`, or `{ 1, 1, 1, NumROIs }`. Each value is the index of a batch from `InputTensor`. The behavior is undefined if the values are not in the range $[0, \text{BatchCount}]$.

OutputTensor

Type: `const DML_TENSOR_DESC*`

A tensor containing the output data. The expected dimensions of `OutputTensor` are `{ NumROIs, ChannelCount, OutputHeight, OutputWidth }`.

ReductionFunction

Type: `DML_REDUCE_FUNCTION`

The reduction function to use when reducing across all input samples that contribute to an output element (`DML_REDUCE_FUNCTION_AVERAGE` or `DML_REDUCE_FUNCTION_MAX`). The number of input samples to reduce across is bounded by `MinimumSamplesPerOutput` and `MaximumSamplesPerOutput`.

InterpolationMode

Type: `DML_INTERPOLATION_MODE`

The interpolation mode to use when resizing the regions.

- `DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`. Uses the *Nearest Neighbor* algorithm, which chooses the input element nearest to the corresponding pixel center for each output element.
- `DML_INTERPOLATION_MODE_LINEAR`. Uses the *Bilinear* algorithm, which computes the output element by doing the weighted average of the 2 nearest neighboring input elements per dimension. Since only 2 dimensions are resized, the weighted average is computed on a total of 4 input elements for each output element.

SpatialScaleX

Type: **FLOAT**

The X (or width) component of the scaling factor to multiply the *ROITensor* coordinates by in order to make them proportionate to *InputHeight* and *InputWidth*. For example, if *ROITensor* contains normalized coordinates (values in the range [0..1]), then *SpatialScaleX* would usually have the same value as *InputWidth*.

SpatialScaleY

Type: **FLOAT**

The Y (or height) component of the scaling factor to multiply the *ROITensor* coordinates by in order to make them proportionate to *InputHeight* and *InputWidth*. For example, if *ROITensor* contains normalized coordinates (values in the range [0..1]), *SpatialScaleY* would usually have the same value as *InputHeight*.

OutOfBounds inputValue

Type: **FLOAT**

The value to read from *InputTensor* when the ROIs are outside the bounds of *InputTensor*. This can happen when the values obtained after scaling *ROITensor* by *SpatialScaleX* and *SpatialScaleY* are bigger than *InputWidth* and *InputHeight*.

MinimumSamplesPerOutput

Type: **UINT**

The minimum number of input samples to use for every output element. The operator will calculate the number of input samples by doing *ScaledCropSize / OutputSize*, and then clamp it to *MinimumSamplesPerOutput* and *MaximumSamplesPerOutput*.

MaximumSamplesPerOutput

Type: **UINT**

The maximum number of input samples to use for every output element. The operator will calculate the number of input samples by doing *ScaledCropSize / OutputSize*, and then clamp it to *MinimumSamplesPerOutput* and *MaximumSamplesPerOutput*.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_3_0`.

Tensor constraints

InputTensor, *OutputTensor*, and *ROITensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
ROITensor	Input	2 to 4	FLOAT32, FLOAT16
BatchIndicesTensor	Input	1 to 4	UINT64, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
ROITensor	Input	2 to 4	FLOAT32, FLOAT16
BatchIndicesTensor	Input	1 to 4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ROI_ALIGN1_OPERATOR_DESC structure (directml.h)

Article03/15/2023

Performs an ROI align operation, as described in the [Mask R-CNN](#) paper. In summary, the operation extracts cropped windows from the input image tensor, and resizes them to a common output size specified by the last 2 dimensions of *OutputTensor* using the specified *InterpolationMode*.

The general logic is as follows.

```
for every region roiIndex
    outputSizeX = OutputTensor.Sizes[3]
    outputSizeY = OutputTensor.Sizes[2]
    scaledRegionX1 = ROITensor[roiIndex, 0] * SpatialScaleX
    scaledRegionY1 = ROITensor[roiIndex, 1] * SpatialScaleY
    scaledRegionX2 = ROITensor[roiIndex, 2] * SpatialScaleX
    scaledRegionY2 = ROITensor[roiIndex, 3] * SpatialScaleY
    scaledRegionSizeX = scaledRegionX2 - scaledRegionX1
    scaledRegionSizeY = scaledRegionY2 - scaledRegionY1
    inputSamplesPerOutputSampleX = clamp(scaledRegionSizeX / outputSizeX,
    MinimumSamplesPerOutput, MaximumSamplesPerOutput)
    inputSamplesPerOutputSampleY = clamp(scaledRegionSizeY / outputSizeY,
    MinimumSamplesPerOutput, MaximumSamplesPerOutput)
    outputSampleSizeX = outputSizeX * inputSamplesPerOutputSampleX
    outputSampleSizeY = outputSizeY * inputSamplesPerOutputSampleY
    outputSampleToInputScaleX = scaledRegionSizeX / outputSampleSizeX
    outputSampleToInputScaleY = scaledRegionSizeY / outputSampleSizeX

    compute all output values
endfor
```

Compute all the output values for the current region as follows.

```
for every output tensor element x y and channel in the region
    outputValue = getOutputValue(channel, outputTensorX, outputTensorY)
    OutputTensor[roiIndex, channel, outputTensorY, outputTensorX] =
    outputValue
endfor
```

Compute each input sample for the output element as follows.

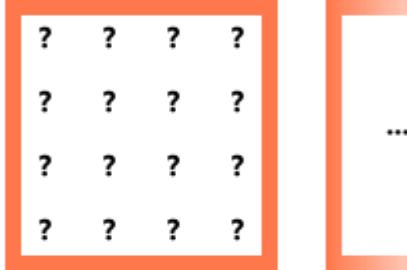
```
outputTensorSampleX = outputTensorX * inputSamplesPerOutputSampleX
outputTensorSampleY = outputTensorY * inputSamplesPerOutputSampleY
outputValue = 0
for sampleX from outputTensorSampleX to <= outputTensorSampleX +
inputSamplesPerOutputSampleX
    for sampleY from outputTensorSampleY to <= outputTensorSampleY +
inputSamplesPerOutputSampleY
        inputTensorX = (sampleX - OutputPixelOffset) *
outputSampleToInputScaleX + scaledRegionX1 - InputPixelOffset
        inputTensorY = (sampleY - OutputPixelOffset) *
outputSampleToInputScaleY + scaledRegionY1 - InputPixelOffset
        inputValue = interpolate2D(InputTensor,
BatchIndicesTensor[roiIndex], channel, inputTensorX, inputTensorY)
        outputValue = either average or maximum with inputValue
    endfor
endfor
return outputValue
```

Examples

Input image tensor values, 6x6:

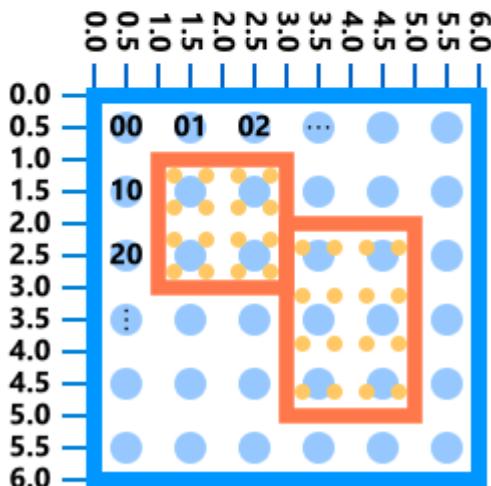
00	01	02	03	04	05
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Output image tensor values, 4x4:



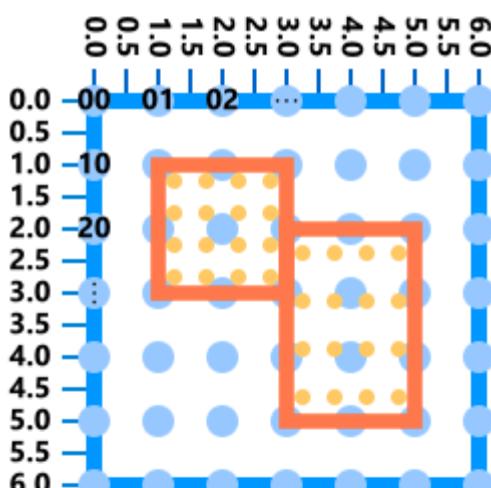
InputPixelOffset = 0.5
OutputPixelOffset = -0.5
AlignRegionsToCorners = false

region 0's output:
[08.25, 08.75, 09.25, 09.75],
[13.25, 13.75, 14.25, 14.75],
[18.25, 18.75, 19.25, 19.75],
[23.25, 23.75, 24.25, 24.75]



InputPixelOffset = 0.0
OutputPixelOffset = -0.5
AlignRegionsToCorners = false

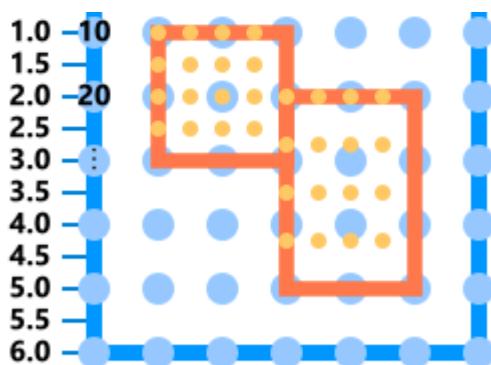
region 0's output:
[13.75, 14.25, 14.75, 15.25],
[18.75, 19.25, 19.75, 20.25],
[23.75, 24.25, 24.75, 25.25],
[28.75, 29.25, 29.75, 30.25]



InputPixelOffset = 0.0

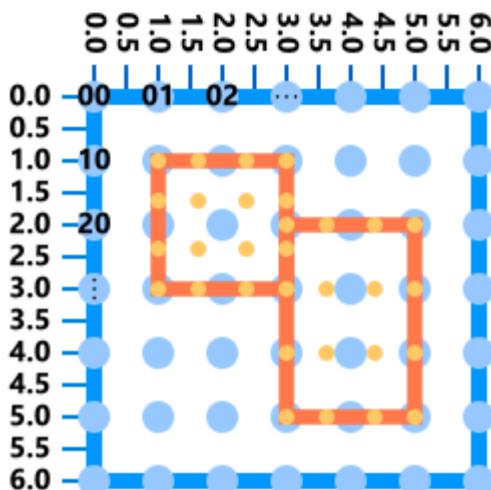
OutputPixelOffset = 0.0
AlignRegionsToCorners = false

region 0's output:
[11.00, 11.50, 12.00, 12.50],
[16.00, 16.50, 17.00, 17.50],
[21.00, 21.50, 22.00, 22.50],
[26.00, 26.50, 27.00, 27.50]



InputPixelOffset = 0.0
OutputPixelOffset = 0.0
AlignRegionsToCorners = true

region 0's output:
[11.00, 11.66, 12.33, 13.00],
[17.66, 18.33, 19.00, 19.66],
[24.33, 25.00, 25.66, 26.33],
[31.00, 31.66, 32.33, 33.00]



Syntax

C++

```
struct DML_ROI_ALIGN1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ROITensor;
    const DML_TENSOR_DESC *BatchIndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_REDUCE_FUNCTION ReductionFunction;
    DML_INTERPOLATION_MODE InterpolationMode;
    FLOAT SpatialScaleX;
    FLOAT SpatialScaleY;
    FLOAT InputPixelOffset;
    FLOAT OutputPixelOffset;
    FLOAT OutOfBoundsInputValue;
    UINT MinimumSamplesPerOutput;
    UINT MaximumSamplesPerOutput;
    BOOL AlignRegionsToCorners;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the input data with dimensions { BatchCount, ChannelCount, InputHeight, InputWidth }.

ROITensor

Type: **const DML_TENSOR_DESC***

A tensor containing the regions of interest (ROI) data, a series of bounding boxes in floating-point coordinates that point into the X and Y dimensions of the input tensor. The allowed dimensions of *ROITensor* are { NumROIs, 4 }, { 1, NumROIs, 4 }, or { 1, 1, NumROIs, 4 }. For each ROI, the values will be the coordinates of its top-left and bottom-right corners in the order [x1, y1, x2, y2]. Regions can be empty, meaning that all output pixels come from the single input coordinate, and regions can be inverted (for example, x2 less than x1), meaning that the output receives a mirrored/flipped version of the input. These coordinates are first scaled by *SpatialScaleX* and *SpatialScaleY*, but if they are both 1.0, then the region rectangles simply correspond directly to the input tensor coordinates.

BatchIndicesTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the batch indices to extract the ROIs from. The allowed dimensions of *BatchIndicesTensor* are { NumROIs }, { 1, NumROIs }, { 1, 1, NumROIs }, or { 1, 1, 1, NumROIs }. Each value is the index of a batch from *InputTensor*. The behavior is undefined if the values are not in the range [0, BatchCount).

OutputTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the output data. The expected dimensions of *OutputTensor* are { NumROIs, ChannelCount, OutputHeight, OutputWidth }.

ReductionFunction

Type: **DML_REDUCE_FUNCTION**

The reduction function to use when reducing across all input samples that contribute to an output element (**DML_REDUCE_FUNCTION_AVERAGE** or **DML_REDUCE_FUNCTION_MAX**). The number of input samples to reduce across is bounded by *MinimumSamplesPerOutput* and *MaximumSamplesPerOutput*.

InterpolationMode

Type: [DML_INTERPOLATION_MODE](#)

The interpolation mode to use when resizing the regions.

- **DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR**. Uses the *nearest neighbor* algorithm, which chooses the input element nearest to the corresponding pixel center for each output element.
- **DML_INTERPOLATION_MODE_LINEAR**. Uses the *bilinear* algorithm, which computes the output element by doing the weighted average of the 2 nearest neighboring input elements per dimension. Since only 2 dimensions are resized, the weighted average is computed on a total of 4 input elements for each output element.

SpatialScaleX

Type: [FLOAT](#)

The X (or width) component of the scaling factor to multiply the *ROI*/*Tensor* coordinates by in order to make them proportionate to *InputHeight* and *InputWidth*. For example, if *ROI*/*Tensor* contains normalized coordinates (values in the range $[0..1]$), then *SpatialScaleX* would usually have the same value as *InputWidth*.

SpatialScaleY

Type: [FLOAT](#)

The Y (or height) component of the scaling factor to multiply the *ROI*/*Tensor* coordinates by in order to make them proportionate to *InputHeight* and *InputWidth*. For example, if *ROI*/*Tensor* contains normalized coordinates (values in the range $[0..1]$), then *SpatialScaleY* would usually have the same value as *InputHeight*.

InputPixelOffset

Type: [FLOAT](#)

The offset from $(0, 0)$ of the input coordinates to the top-left pixel center, typically either 0 or 0.5. When this value is 0, the top-left corner of the pixel is used instead of its center, which usually won't give the expected result, but is useful for compatibility with some frameworks. When this value is 0.5, pixels are treated as being at the center, which is the same behavior as [DML_ROI_ALIGN_OPERATOR_DESC](#).

OutputPixelOffset

Type: **FLOAT**

The offset from the top-left pixel center to $(0,0)$ of the output coordinates, typically either 0 or -0.5. When this value is 0, the top-left corner of the pixel is used instead of its center, which usually won't give the expected result, but is useful for compatibility with some frameworks. When this value is -0.5, pixels are treated as being at the center, which is the same behavior as [DML_ROI_ALIGN_OPERATOR_DESC](#).

`OutOfBoundsInputValue`

Type: **FLOAT**

The value to read from *InputTensor* when the ROIs are outside the bounds of *InputTensor*. This can happen when the values obtained after scaling *ROITensor* by *SpatialScaleX* and *SpatialScaleY* are bigger than *InputWidth* and *InputHeight*.

`MinimumSamplesPerOutput`

Type: **UINT**

The minimum number of input samples to use for every output element. The operator will calculate the number of input samples by doing *ScaledCropSize* / *OutputSize*, and then clamping it to *MinimumSamplesPerOutput* and *MaximumSamplesPerOutput*.

`MaximumSamplesPerOutput`

Type: **UINT**

The maximum number of input samples to use for every output element. The operator will calculate the number of input samples by doing *ScaledCropSize* / *OutputSize*, and then clamping it to *MinimumSamplesPerOutput* and *MaximumSamplesPerOutput*.

`AlignRegionsToCorners`

Type: **BOOL**

The output sample points in each region should be stretched out to the very corners of the region rather than evenly spread within the region. The default value is **FALSE**, which is the same behavior as [DML_ROI_ALIGN_OPERATOR_DESC](#).

Remarks

Availability

This operator was introduced in **DML_FEATURE_LEVEL_4_0**.

Tensor constraints

InputTensor, *OutputTensor*, and *ROITensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
ROITensor	Input	2 to 4	FLOAT32, FLOAT16
BatchIndicesTensor	Input	1 to 4	UINT64, UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16

DML_FEATURE_LEVEL_4_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
ROITensor	Input	2 to 4	FLOAT32, FLOAT16
BatchIndicesTensor	Input	1 to 4	UINT32
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Build 22000

Requirement	Value
Minimum supported server	Windows Build 22000
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_ROI_POOLING_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Performs a MaxPool function across the input tensor (according to regions of interest, or ROIs). For each output element, the coordinates of its corresponding ROI in the input are computed by the following equations.

Let `Y` be an index into the third dimension of `InputTensor` (`{ BatchSize, ChannelCount, **height**, width }`).

Let `X` be an index into the fourth dimension of `InputTensor` (`{ BatchSize, ChannelCount, height, **width** }`).

```
x1 = round(RoiX1 * SpatialScale)
x2 = round(RoiX2 * SpatialScale)
y1 = round(RoiY1 * SpatialScale)
y2 = round(RoiY2 * SpatialScale)

RegionHeight = y2 - y1 + 1
RegionWidth = x2 - x1 + 1

StartY = (OutputIndices.Y * RegionHeight) / PooledSize.Height + y1
StartX = (OutputIndices.X * RegionWidth) / PooledSize.Width + x1

EndY = ((OutputIndices.Y + 1) * RegionHeight + PooledSize.Height - 1) /
PooledSize.Height + y1
EndX = ((OutputIndices.X + 1) * RegionWidth + PooledSize.Width - 1) /
PooledSize.Width + x1
```

If the computed coordinates are out of bounds, then they are clamped to the input boundaries.

Syntax

```
C++

struct DML_ROI_POOLING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ROITensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT          SpatialScale;
```

```
    DML_SIZE_2D          PooledSize;  
};
```

Members

`InputTensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the input data with dimensions `{ BatchCount, ChannelCount, InputHeight, InputWidth }`.

`ROITensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the regions of interest (ROI) data. The expected dimensions of `ROITensor` are `{ 1, 1, NumROIs, 5 }` and the data for each ROI is `[BatchID, x1, y1, x2, y2]`. $x1, y1, x2, y2$ are the inclusive coordinates of the corners of each ROI and $x2 \geq x1, y2 \geq y1$.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

A tensor containing the output data. The expected dimensions of `OutputTensor` are `{ NumROIs, InputChannelCount, PooledSize.Height, PooledSize.Width }`.

`SpatialScale`

Type: **FLOAT**

Multiplicative spatial scale factor used to translate the ROI coordinates from their input scale to the scale used when pooling.

`PooledSize`

Type: **DML_SIZE_2D**

The ROI pool output size (height, width), which must match the last 2 dimensions of `OutputTensor`.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor, *OutputTensor*, and *ROITensor* must have the same *DataType*.

Tensor support

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
ROITensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[\[+\] Yes](#)

[\[+\] No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_SCALAR_UNION union (directml.h)

Article02/22/2024

A union of scalar types.

Syntax

C++

```
union DML_SCALAR_UNION {
    BYTE Bytes[8];
    INT8 Int8;
    UINT8 UInt8;
    INT16 Int16;
    UINT16 UInt16;
    INT32 Int32;
    UINT32 UInt32;
    INT64 Int64;
    UINT64 UInt64;
    FLOAT Float32;
    DOUBLE Float64;
};
```

Members

Bytes[8]

An 8-byte array.

Int8

An 8-bit signed integer.

UInt8

An 8-bit unsigned integer.

Int16

A 16-bit signed integer.

UInt16

A 16-bit unsigned integer.

`Int32`

A 32-bit signed integer.

`UInt32`

A 32-bit unsigned integer.

`Int64`

A 64-bit signed integer.

`UInt64`

A 64-bit unsigned integer.

`Float32`

A single precision floating-point number.

`Float64`

A double precision floating-point number.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_SCALE_BIAS structure (directml.h)

Article 02/22/2024

Contains the values of scale and bias terms supplied to a DirectML operator. Scale and bias have the effect of applying the function $g(x) = x * Scale + Bias$.

Syntax

C++

```
struct DML_SCALE_BIAS {
    FLOAT Scale;
    FLOAT Bias;
};
```

Members

Scale

Type: [FLOAT](#)

The scale term in $g(x) = x * Scale + Bias$.

Bias

Type: [FLOAT](#)

The bias term in $g(x) = x * Scale + Bias$.

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_SCATTER_ND_OPERATOR_DESC structure (directml.h)

Article07/27/2022

Copies the whole input tensor to the output, then overwrites selected indices with corresponding values from the updates tensor. This operator performs the following pseudocode, where "..." represents a series of coordinates, with the exact behavior determined by the axis and indices size.

```
output = input
output[indices[...]] = updates[...]
```

If two output element indices overlap (which is invalid), there is no guarantee of which last write wins.

Syntax

C++

```
struct DML_SCATTER_ND_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *UpdatesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT InputDimensionCount;
    UINT IndicesDimensionCount;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor to read from.

IndicesTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the indices. The *DimensionCount* of this tensor must match *InputTensor.DimensionCount*. The last dimension of the *IndicesTensor* is actually the number of coordinates per index tuple, and it mustn't exceed *InputTensor.DimensionCount*. For example, an indices tensor of size `{1,4,5,2}` with *IndicesDimensionCount* = 3 means a 4x5 array of 2-value coordinate tuples that index into *InputTensor*.

Starting with `DML_FEATURE_LEVEL_3_0`, this operator supports negative index values when using a signed integral type with this tensor. Negative indices are interpreted as being relative to the end of the respective dimension. For example, an index of -1 refers to the last element along that dimension.

`UpdatesTensor`

Type: `const DML_TENSOR_DESC*`

A tensor containing the new values to replace the existing input values at the corresponding indices. The *DimensionCount* of this tensor must match *InputTensor.DimensionCount*. The expected *UpdatesTensor.Sizes* are the concatenation of the *IndicesTensor.Sizes* leading segments and *InputTensor.Sizes* trailing segment to yield the following.

```
indexTupleSize = IndicesTensor.Sizes[IndicesTensor.DimensionCount - 1]
UpdatesTensor.Sizes = [
    1...,
    IndicesTensor.Sizes[(IndicesTensor.DimensionCount -
    IndicesDimensionCount) .. (IndicesTensor.DimensionCount - 1)],
    InputTensor.Sizes[(InputTensor.DimensionCount - indexTupleSize) ..
    InputTensor.DimensionCount]
]
```

The dimensions are right-aligned, with leading 1 values prepended if needed to satisfy *UpdatesTensor.DimensionCount*.

Here's an example.

```
InputTensor.Sizes = [3,4,5,6,7]
InputDimensionCount = 5
IndicesTensor.Sizes = [1,1, 1,2,3]
IndicesDimensionCount = 3 // can be thought of as a [1,2] array of 3-
coordinate tuples

// The [1,2] comes from the indices tensor (ignoring last dimension, which
```

```
is the tuple size),
// and the [6,7] comes from input tensor, ignoring the first 3 dimensions
// since the index tuples are 3 elements (from the indices tensor last
dimension).
UpdatesTensor.Sizes = [1, 1, 2, 6, 7]
```

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the results to. The *Sizes* and *DataType* of this tensor must match *InputTensor.Sizes*.

InputDimensionCount

Type: **UINT**

The number of actual input dimensions within the *InputTensor* after ignoring any irrelevant leading ones, ranging [1, *InputTensor.DimensionCount*). For example, given *InputTensor.Sizes* = {1,1,4,6} and *InputDimensionCount* = 3, the actual meaningful indices are {1,4,6}.

IndicesDimensionCount

Type: **UINT**

The number of actual index dimensions within the *IndicesTensor* after ignoring any irrelevant leading ones, ranging [1, *IndicesTensor.DimensionCount*). For example, given *IndicesTensor.Sizes* = {1,1,4,6} and *IndicesDimensionCount* = 3, the actual meaningful indices are {1,4,6}.

Examples

```
InputTensor: (Sizes:{8}, DataType:FLOAT32)
[1, 2, 3, 4, 5, 6, 7, 8]

IndicesTensor: (Sizes:{4,1}, DataType:FLOAT32)
[[4], [3], [1], [7]]

UpdatesTensor: (Sizes:{4}, DataType:FLOAT32)
[9, 10, 11, 12]

// output = input
// output[indices[x, 0]] = updates[x]
```

```
OutputTensor: (Sizes:{8}, DataType:FLOAT32)
[1, 11, 3, 10, 9, 6, 7, 12]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_2_1`.

Tensor constraints

- *IndicesTensor*, *InputTensor*, *OutputTensor*, and *UpdatesTensor* must have the same *DimensionCount*.
- *InputTensor*, *OutputTensor*, and *UpdatesTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
UpdatesTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
UpdatesTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	4	UINT32
UpdatesTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_SCATTER_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Copies the whole input tensor to the output, then overwrites selected indices with corresponding values from the updates tensor. This operator performs the following pseudocode.

```
output = input
output[index[i, j, k, ...], j, k, ...] = updates[i, j, k, ...] // if axis == 0
output[i, index[i, j, k, ...], k, ...] = updates[i, j, k, ...] // if axis == 1
output[i, j, index[i, j, k, ...], ...] = updates[i, j, k, ...] // if axis == 2
...
...
```

If two output element indices overlap (which is invalid), then there's no guarantee of which last write wins.

DML_SCATTER_OPERATOR_DESC is the inverse of [DML_GATHER_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_SCATTER_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *UpdatesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
};
```

Members

InputTensor

Type: [const DML_TENSOR_DESC*](#)

The tensor to read from.

IndicesTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the indices into the output tensor. The *Sizes* must match *InputTensor.Sizes* for every dimension except *Axis*.

Starting with `DML_FEATURE_LEVEL_3_0`, this operator supports negative index values when using a signed integral type with this tensor. Negative indices are interpreted as being relative to the end of the axis dimension. For example, an index of -1 refers to the last element along that dimension.

UpdatesTensor

Type: **const DML_TENSOR_DESC***

A tensor containing the new values to replace the existing input values at the corresponding indices. The *Sizes* of this tensor must match *IndicesTensor.Sizes*. The *DataType* must match *InputTensor.DataType*.

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write the results to. The *Sizes* and *DataType* of this tensor must match *InputTensor*.

Axis

Type: **UINT**

The axis dimension to use for indexing in *OutputTensor*, ranging `[0, OutputTensor.DimensionCount)`.

Examples

Example 1. 1D scatter

```
Axis: 0
```

```
InputTensor: (Sizes:{5}, DataType:FLOAT32)
[0,1,2,3,4]
```

```
IndicesTensor: (Sizes:{4}, DataType:UINT32)
[3,1,3,0]

UpdatesTensor: (Sizes:{4}, DataType:FLOAT32)
[5,6,7,8]

// output = input
// output[indices[x]] = updates[x]
OutputTensor: (Sizes:{5}, DataType:FLOAT32)
[8,6,2,7,4]
```

Example 2. 2D scatter

```
Axis: 0

InputTensor: (Sizes:{2,3}, DataType:FLOAT32)
[[0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0]]

IndicesTensor: (Sizes:{2,3}, DataType:UINT32)
[[1, 0, 2],
 [0, 2, 1]]

UpdatesTensor: (Sizes:{2,3}, DataType:FLOAT32)
[[10, 11, 12],
 [20, 21, 22]]

// output = input
// output[indices[y, x], x] = updates[y, x]
OutputTensor: (Sizes:{3,3}, DataType:FLOAT32)
[[20, 11, 0],
 [10, 0, 22],
 [0, 21, 12]]
```

Remarks

DML_SCATTER_OPERATOR_DESC has been more properly aliased to the name DML_SCATTER_ELEMENTS_OPERATOR_DESC as the proper counterpart to DML_GATHER_ELEMENTS_OPERATOR_DESC. This is because DML_SCATTER_OPERATOR_DESC was not really symmetric to DML_GATHER_OPERATOR_DESC.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

- *IndicesTensor*, *InputTensor*, *OutputTensor*, and *UpdatesTensor* must have the same *DimensionCount*.
- *InputTensor*, *OutputTensor*, and *UpdatesTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
UpdatesTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	1 to 8	INT64, INT32, UINT64, UINT32
UpdatesTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8,

Tensor	Kind	Supported dimension counts	Supported data types
			UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
IndicesTensor	Input	4	UINT32
UpdatesTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
IndicesTensor	Input	4	UINT32
UpdatesTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)

Requirement	Value
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_SIZE_2D structure (directml.h)

Article 02/22/2024

Contains values that can represent the size (as supplied to a DirectML operator) of a 2-D plane of elements within a tensor, or a 2-D scale, or any 2-D width/height value.

Syntax

C++

```
struct DML_SIZE_2D {
    UINT Width;
    UINT Height;
};
```

Members

Width

Type: [UINT](#)

The width.

Height

Type: [UINT](#)

The height.

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_SLICE_GRAD_OPERATOR_DESC structure (directml.h)

Article 10/20/2021

Computes backpropagation gradients for Slice (see [DML_SLICE1_OPERATOR_DESC](#)).

Recall that [DML_SLICE1_OPERATOR_DESC](#) extracts a subregion of an input tensor. Given an *InputGradientTensor* with the same sizes as the *output* of an equivalent [DML_SLICE1_OPERATOR_DESC](#), this operator produces an *OutputGradientTensor* with the same sizes as the *input* of [DML_SLICE1_OPERATOR_DESC](#). The sliced elements are propagated to the output, and all other elements are set to 0.

As an example, consider a [DML_SLICE1_OPERATOR_DESC](#) that extracts the following elements from a tensor:

```
InputTensor          OutputTensor
[[a, b, c, d],
 [e, f, g, h],    Slice   [[a, c],
 [i, j, k, l],    -->     [i, k]]
 [m, n, o, p]]
```

If provided the same *InputWindowOffsets/Sizes/Strides* as in the above example, this operator would then perform the following transform.

```
InputGradientTensor      OutputGradientTensor
[[a, c],    SliceGrad   [[a, 0, c, 0],
 [i, k]]       -->     [0, 0, 0, 0],
                  [i, 0, k, 0],
                  [0, 0, 0, 0]]
```

Syntax

C++

```
struct DML_SLICE_GRAD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputGradientTensor;
    const DML_TENSOR_DESC *OutputGradientTensor;
    UINT                 DimensionCount;
    const UINT           *InputWindowOffsets;
```

```
    const UINT           *InputWindowSizes;
    const INT            *InputWindowStrides;
};
```

Members

`InputGradientTensor`

Type: `const DML_TENSOR_DESC*`

The incoming gradient tensor. This is typically obtained from the output of backpropagation of a preceding layer. Typically, this tensor would have the same sizes as the *output* of the corresponding `DML_SLICE1_OPERATOR_DESC` in the forward pass.

`OutputGradientTensor`

Type: `const DML_TENSOR_DESC*`

An output tensor containing the backpropagated gradients. Typically, this tensor would have the same sizes as the *input* of the corresponding `DML_SLICE1_OPERATOR_DESC` in the forward pass.

`DimensionCount`

Type: `UINT`

The number of elements in the `InputWindowOffsets`, `InputWindowSizes`, and `InputWindowStrides` arrays. This value must equal the `DimensionCount` provided in the `InputGradientTensor` and `OutputGradientTensor`.

`InputWindowOffsets`

Type: `_Field_size_(DimensionCount) const UINT*`

See `InputWindowOffsets` in `DML_SLICE1_OPERATOR_DESC`.

`InputWindowSizes`

Type: `_Field_size_(DimensionCount) const UINT*`

See `InputWindowSizes` in `DML_SLICE1_OPERATOR_DESC`.

`InputWindowStrides`

Type: `_Field_size_(DimensionCount) const UINT*`

See *InputWindowStrides* in [DML_SLICE1_OPERATOR_DESC](#).

Note that unlike [DML_SLICE1_OPERATOR_DESC](#), this operator requires non-zero strides. That's because with a zero stride, it's ambiguous as to which input element should map to each output element, and therefore backpropagation can't be performed. Like [DML_SLICE1_OPERATOR_DESC](#), negative strides flip the input window direction along that axis.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_3_0](#).

Tensor constraints

InputGradientTensor and *OutputGradientTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputGradientTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputGradientTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_1 and above

[] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputGradientTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputGradientTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputGradientTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputGradientTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DML_SLICE_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Extracts a single subregion (a "slice") of an input tensor.

The elements copied in the slice are determined using three values for each dimension.

- The *offset* marks the first element to copy in a dimension.
- The *size* marks the number of elements to copy in a dimension.
- The *stride* indicates the element increment or step in a dimension.

The provided *Offsets*, *Sizes*, and *Strides* must only copy elements that are within the bounds of the input tensor (out-of-bounds reads are not permitted). The *Sizes* of the slice must exactly match the output tensor sizes. In general, the elements copied are calculated as follows.

```
OutputTensor[OutputCoordinates] = InputTensor[Offsets + Strides *  
OutputCoordinates]
```

Syntax

C++

```
struct DML_SLICE_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT DimensionCount;  
    const UINT *Offsets;  
    const UINT *Sizes;  
    const UINT *Strides;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor to extract slices from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the sliced data results to.

`DimensionCount`

Type: `UINT`

The number of dimensions. This field determines the size of the *Offsets*, *Sizes*, and *Strides* arrays. This value must match the *DimensionCount* of the input and output tensors. This value must be between 1 and 8, inclusively, starting from `DML_FEATURE_LEVEL_3_0`; earlier feature levels require a value of either 4 or 5.

`Offsets`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the slice's start along each dimension of the input tensor, in elements.

`Sizes`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the slice's size along each dimension, in elements. The values in this array must match the sizes specified in the output tensor.

`Strides`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the slice's stride along each dimension of the input tensor, in elements. A stride larger than 1 indicates that elements of the input tensor may be skipped (for example, a stride of 2 will select every second element along the dimension).

Examples

The following examples use the same input tensor:

```
InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[1, 2, 3, 4],
```

```
[ 5,  6,  7,  8],  
[ 9, 10, 11, 12],  
[13, 14, 15, 16]]]
```

Example 1. Contiguous slice

```
Offsets = {0, 0, 1, 2}  
Sizes   = {1, 1, 3, 2}  
Strides = {1, 1, 1, 1}  
  
OutputTensor: (Sizes:{1, 1, 3, 2}, DataType:FLOAT32)  
[[[[ 7,  8],  
   [11, 12],  
   [15, 16]]]]
```

Example 2. Strided slice

```
Offsets = {0, 0, 1, 0}  
Sizes   = {1, 1, 2, 2}  
Strides = {1, 1, 2, 3}  
  
OutputTensor: (Sizes:{1, 1, 2, 2}, DataType:FLOAT32)  
[[[[ 5,  8],  
   [13, 16]]]]
```

Remarks

A newer version of this operator, [DML_SLICE1_OPERATOR_DESC](#), was introduced in [DML_FEATURE_LEVEL_2_1](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, INT16, UINT32, UINT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, UINT32, UINT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[!\[\]\(1b67493d1b9291268fc25eee441ddc5b_img.jpg\) Yes](#)[!\[\]\(d7d1221f46e1fd94ac019391ff84c5a9_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_SLICE1_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Extracts a single subregion (a "slice") of an input tensor.

The *input window* describes the bounds of the input tensor to consider in the slice. The window is defined using three values for each dimension.

- The *offset* marks the *beginning of the window* in a dimension.
- The *size* marks the extent of the window in a dimension. The *end of the window* in a dimension is `offset + size - 1`.
- The *stride* indicates how to traverse the elements in a dimension.
 - The magnitude of the stride indicates how many elements to advance when copying within the window.
 - If a stride is positive, elements are copied starting at the *beginning* of the window in the dimension.
 - If a stride is negative, elements are copied starting at the *end* of the window in the dimension.

The following pseudo-code illustrates how elements are copied using the input window. Note how elements in a dimension are copied from start to end with a positive stride, and copied from end to start with a negative stride.

```
CopyStart = InputWindowOffsets
for dimension i in [0, DimensionCount - 1]:
    if InputWindowStrides[i] < 0:
        CopyStart[i] += InputWindowSizes[i] - 1 // start at the end of the
                                                // window in this dimension

OutputTensor[OutputCoordinates] = InputTensor[CopyStart + InputWindowStrides
                                             * OutputCoordinates]
```

The input window mustn't be empty in any dimension, and the window mustn't extend beyond the dimensions of the input tensor (out-of-bounds reads aren't permitted). The window's size and strides effectively limit the number of elements that may be copied from each dimension `i` of the input tensor.

```
MaxCopiedElements[i] = 1 + (InputWindowSize[i] - 1) / InputWindowStrides[i]
```

The output tensor isn't required to copy all reachable elements within the window. The slice is valid so long as `1 <= OutputSizes[i] <= MaxCopiedElements[i]`.

Syntax

C++

```
struct DML_SLICE1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT DimensionCount;
    const UINT *InputWindowOffsets;
    const UINT *InputWindowSizes;
    const INT *InputWindowStrides;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to extract slices from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

The tensor to write the sliced data results to.

`DimensionCount`

Type: `UINT`

The number of dimensions. This field determines the size of the `InputWindowOffsets`, `InputWindowSizes`, and `InputWindowStrides` arrays. This value must match the `DimensionCount` of the input and output tensors. This value must be between 1 and 8, inclusively, starting from `DML_FEATURE_LEVEL_3_0`; earlier feature levels require a value of either 4 or 5.

`InputWindowOffsets`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the beginning (in elements) of the input window in each dimension.
Values in the array must satisfy the constraint `InputWindowOffsets[i] + InputWindowSizes[i] <= InputTensor.Sizes[i]`

`InputWindowSizes`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the extent (in elements) of the input window in each dimension.
Values in the array must satisfy the constraint `InputWindowOffsets[i] + InputWindowSizes[i] <= InputTensor.Sizes[i]`

`InputWindowStrides`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the slice's stride along each dimension of the input tensor, in elements. The magnitude of the stride indicates how many elements to advance when copying within the input window. The sign of the stride determines if elements are copied starting at the beginning of the window (positive stride) or end of the window (negative stride). Strides may not be 0.

Examples

The following examples use this same input tensor.

```
InputTensor: (Sizes:{1, 1, 4, 4}, DataType:FLOAT32)
[[[[ 1,  2,  3,  4],
   [ 5,  6,  7,  8],
   [ 9, 10, 11, 12],
   [13, 14, 15, 16]]]]
```

Example 1. Strided slice with positive strides

```
InputWindowOffsets = {0, 0, 0, 1}
InputWindowSizes   = {1, 1, 4, 3}
InputWindowStrides = {1, 1, 2, 2}

OutputTensor: (Sizes:{1, 1, 2, 2}, DataType:FLOAT32)
```

```
[[[[ 2,  4],  
   [10, 12]]]]
```

The copied elements are calculated as follows.

```
Output[0,0,0,0] = {0,0,0,1} + {1,1,2,2} * {0,0,0,0} = Input[{0,0,0,1}] = 2  
Output[0,0,0,1] = {0,0,0,1} + {1,1,2,2} * {0,0,0,1} = Input[{0,0,0,3}] = 4  
Output[0,0,1,0] = {0,0,0,1} + {1,1,2,2} * {0,0,1,0} = Input[{0,0,2,1}] = 10  
Output[0,0,1,1] = {0,0,0,1} + {1,1,2,2} * {0,0,1,1} = Input[{0,0,2,3}] = 12
```

Example 2. Strided slice with negative strides

```
InputWindowOffsets = {0, 0, 0, 1}  
InputWindowSizes   = {1, 1, 4, 3}  
InputWindowStrides = {1, 1, -2, 2}  
  
OutputTensor: (Sizes:{1, 1, 2, 2}, DataType:FLOAT32)  
[[[[14, 16],  
   [ 6,  8]]]]
```

Recall that dimensions with negative window strides start copying at the end of the input window for that dimension; this is done by adding `InputWindowSize[i] - 1` to the window offset. Dimensions with a positive stride simply start at `InputWindowOffset[i]`.

- Axis 0 (+1 window stride) starts copying at `InputWindowOffsets[0] = 0`.
- Axis 1 (+1 window stride) starts copying at `InputWindowOffsets[1] = 0`.
- Axis 2 (-2 window stride) starts copying at `InputWindowOffsets[2] + InputWindowSize[0] - 1 = 0 + 4 - 1 = 3`.
- Axis 3 (+2 window stride) starts copying at `InputWindowOffsets[3] = 1`.

The copied elements are calculated as follows.

```
Output[0,0,0,0] = {0,0,3,1} + {1,1,-2,2} * {0,0,0,0} = Input[{0,0,3,1}] = 14  
Output[0,0,0,1] = {0,0,3,1} + {1,1,-2,2} * {0,0,0,1} = Input[{0,0,3,3}] = 16  
Output[0,0,1,0] = {0,0,3,1} + {1,1,-2,2} * {0,0,1,0} = Input[{0,0,1,1}] = 6  
Output[0,0,1,1] = {0,0,3,1} + {1,1,-2,2} * {0,0,1,1} = Input[{0,0,1,3}] = 8
```

Remarks

This operator is similar to [DML_SLICE_OPERATOR_DESC](#), but it differs in two important ways.

- Slice strides may be negative, which allows reversing values along dimensions.
- The input window sizes are not necessarily the same as the output tensor sizes.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_SPACE_TO_DEPTH_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Rearranges blocks of spatial data into depth. The operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension.

This is the inverse transformation of [DML_DEPTH_TO_SPACE_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_SPACE_TO_DEPTH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT BlockSize;
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

The tensor to read from. The input tensor's dimensions are `{ Batch, Channels, Height, Width }`.

`OutputTensor`

Type: [const DML_TENSOR_DESC*](#)

The tensor to write the results to. The output tensor's dimensions are `{ Batch, Channels / (BlockSize * BlockSize), Height * BlockSize, Width * BlockSize }`.

`BlockSize`

Type: [UINT](#)

The width and height of the Blocks that are moved.

Example

```
BlockSize: 2

InputTensor: (Sizes:{1, 2, 4, 6}, DataType:UINT32)
[[[[ 0, 18, 1, 19, 2, 20],
   [36, 54, 37, 55, 38, 56],
   [ 3, 21, 4, 22, 5, 23],
   [39, 57, 40, 58, 41, 59]],
  [[ 9, 27, 10, 28, 11, 29],
   [45, 63, 46, 64, 47, 65],
   [12, 30, 13, 31, 14, 32],
   [48, 66, 49, 67, 50, 68]]]

OutputTensor: (Sizes:{1, 8, 2, 3}, DataType:UINT32)
[[[[0, 1, 2],
   [3, 4, 5]],
  [[9, 10, 11],
   [12, 13, 14]],
  [[18, 19, 20],
   [21, 22, 23]],
  [[27, 28, 29],
   [30, 31, 32]],
  [[36, 37, 38],
   [39, 40, 41]],
  [[45, 46, 47],
   [48, 49, 50]],
  [[54, 55, 56],
   [57, 58, 59]],
  [[63, 64, 65],
   [66, 67, 68]]]]
```

Remarks

A newer version of this operator, [DML_SPACE_TO_DEPTH1_OPERATOR_DESC](#), was introduced in [DML_FEATURE_LEVEL_2_1](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT32, FLOAT16

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_SPACE_TO_DEPTH1_OPERATOR_DESC structure (directml.h)

Article 07/27/2022

Rearranges blocks of spatial data into depth. The operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension.

This is the inverse transformation of [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_SPACE_TO_DEPTH1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT BlockSize;
    DML_DEPTH_SPACE_ORDER Order;
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

The tensor to read from. The input tensor's dimensions are `{ Batch, Channels, Height, Width }`.

`OutputTensor`

Type: [const DML_TENSOR_DESC*](#)

The tensor to write the results to. The output tensor's dimensions are `{ Batch, Channels / (BlockSize * BlockSize), Height * BlockSize, Width * BlockSize }`.

`BlockSize`

Type: [UINT](#)

The width and height of the Blocks that are moved.

Order

Type: [DML_DEPTH_SPACE_ORDER](#)

See [DML_DEPTH_SPACE_ORDER](#).

Examples

Example 1. Depth-column-row order

```
BlockSize: 2
Order: DML_DEPTH_SPACE_ORDER_DEPTH_COLUMN_ROW
InputTensor: (Sizes:{1, 2, 4, 6}, DataType:UINT32)
[[[[ 0, 18, 1, 19, 2, 20],
   [36, 54, 37, 55, 38, 56],
   [ 3, 21, 4, 22, 5, 23],
   [39, 57, 40, 58, 41, 59]],
  [[ 9, 27, 10, 28, 11, 29],
   [45, 63, 46, 64, 47, 65],
   [12, 30, 13, 31, 14, 32],
   [48, 66, 49, 67, 50, 68]]]

OutputTensor: (Sizes:{1, 8, 2, 3}, DataType:UINT32)
[[[[0, 1, 2],
   [3, 4, 5]],
  [[9, 10, 11],
   [12, 13, 14]],
  [[18, 19, 20],
   [21, 22, 23]],
  [[27, 28, 29],
   [30, 31, 32]],
  [[36, 37, 38],
   [39, 40, 41]],
  [[45, 46, 47],
   [48, 49, 50]],
  [[54, 55, 56],
   [57, 58, 59]],
  [[63, 64, 65],
   [66, 67, 68]]]]
```

Example 2. Column-row-depth order

```
BlockSize: 2
Order: DML_DEPTH_SPACE_ORDER_COLUMN_ROW_DEPTH
```

```
InputTensor: (Sizes:{1, 2, 4, 6}, DataType:UINT32)
[[[ [ 0, 9, 1, 10, 2, 11],
    [18, 27, 19, 28, 20, 29],
    [ 3, 12, 4, 13, 5, 14],
    [21, 30, 22, 31, 23, 32]],

   [[36, 45, 37, 46, 38, 47],
    [54, 63, 55, 64, 56, 65],
    [39, 48, 40, 49, 41, 50],
    [57, 66, 58, 67, 59, 68]]]]]

OutputTensor: (Sizes:{1, 8, 2, 3}, DataType:UINT32)
[[[ [0, 1, 2],
    [3, 4, 5]],
   [[9, 10, 11],
    [12, 13, 14]],
   [[18, 19, 20],
    [21, 22, 23]],
   [[27, 28, 29],
    [30, 31, 32]],
   [[36, 37, 38],
    [39, 40, 41]],
   [[45, 46, 47],
    [48, 49, 50]],
   [[54, 55, 56],
    [57, 58, 59]],
   [[63, 64, 65],
    [66, 67, 68]]]]]
```

Remarks

When the *Order* parameter is set to [DML_DEPTH_SPACE_ORDER_DEPTH_COLUMN_ROW](#), this operator is equivalent to [DML_SPACE_TO_DEPTH_OPERATOR_DESC](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_1](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*.

Tensor support

[DML_FEATURE_LEVEL_5_0 and above](#)

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Dimensions	Supported dimension counts	Supported data types
InputTensor	Input	{ BatchCount, InputChannelCount, InputHeight, InputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	{ BatchCount, OutputChannelCount, OutputHeight, OutputWidth }	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_SPLIT_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Splits an input tensor along an axis into multiple output tensors.

All input and output tensors must have the same sizes, except for the split axis. The size of input tensor in the split axis determines the possible splits. For example, if the input tensor's split axis has size 3, then there are these potential splits: 1+1+1 (3 outputs), 1+2 (2 outputs), 2+1 (2 outputs), or 3 (1 output, which is simply a copy of the input tensor). The output tensors' split axis sizes must sum up to exactly the input tensor's split axis size. These constraints are illustrated in the pseudocode below.

```
splitSize = 0;

for (i = 0; i < OutputCount; i++) {
    assert(outputTensors[i]->DimensionCount == inputTensor->DimensionCount);
    for (dim = 0; dim < inputTensor->DimensionCount; dim++) {
        if (dim == Axis) { splitSize += outputTensors[i]->Sizes[dim]; }
        else { assert(outputTensors[i]->Sizes[dim] == inputTensor-
>Sizes[dim]); }
    }
}

assert(splitSize == inputTensor->Sizes[Axis]);
```

Splitting into a single output tensor simply produces a copy of the input tensor.

This operator is the inverse of [DML_JOIN_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_SPLIT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    UINT OutputCount;
    const DML_TENSOR_DESC *OutputTensors;
    UINT Axis;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor to split into multiple output tensors.

OutputCount

Type: **UINT**

This field determines the size of the *OutputTensors* array. This value must be greater than 0.

OutputTensors

Type: **const DML_TENSOR_DESC***

An array containing the descriptions of the tensors split off from the input tensor. The output sizes must have the same sizes as the input tensor except for the split axis.

Axis

Type: **UINT**

The index of the dimension of the input tensor to split. All input and output tensors must have identical sizes in all dimensions except for this axis. This value must be in the range `[0, InputTensor.DimensionCount - 1]`.

Examples

The following examples use this same input tensor.

```
InputTensor: (Sizes:{1, 1, 6, 2}, DataType:FLOAT32)
[[[1, 2],
 [3, 4],
 [5, 6],
 [7, 8],
 [9, 10],
 [11, 12]]]
```

Example 1. Splitting axis 2

```
OutputCount: 3
Axis: 2

OutputTensors[0]: (Sizes:{1, 1, 2, 2}, DataType:FLOAT32)
[[[[1, 2],
   [3, 4]]]]

OutputTensors[1]: (Sizes:{1, 1, 1, 2}, DataType:FLOAT32)
[[[[5, 6]]]]

OutputTensors[2]: (Sizes:{1, 1, 3, 2}, DataType:FLOAT32)
[[[[7, 8],
   [9, 10],
   [11, 12]]]]
```

Example 2. Splitting axis 3

```
OutputCount: 2
Axis: 3

OutputTensors[0]: (Sizes:{1, 1, 6, 1}, DataType:FLOAT32)
[[[[1],
   [3],
   [5],
   [7],
   [9],
   [11]]]]

OutputTensors[1]: (Sizes:{1, 1, 6, 1}, DataType:FLOAT32)
[[[[2],
   [4],
   [6],
   [8],
   [10],
   [12]]]]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensors* must have the same *DataType* and *DimensionCount*.

Tensor support

DML_FEATURE_LEVEL_4_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensors	Array of outputs	1 to 8	FLOAT64, FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensors	Array of outputs	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensors	Array of outputs	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, UINT32, UINT16
OutputTensors	Array of outputs	4	FLOAT32, FLOAT16, INT32, INT16, UINT32, UINT16

Requirements

[\[\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 [Yes](#) [No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_TENSOR_DESC structure (directml.h)

Article 02/22/2024

A generic container for a DirectML tensor description.

Syntax

C++

```
struct DML_TENSOR_DESC {
    DML_TENSOR_TYPE Type;
    const void      *Desc;
};
```

Members

Type

Type: [DML_TENSOR_TYPE](#)

The type of the tensor description. See [DML_TENSOR_TYPE](#) for the available types.

Desc

Type: `const void*`

A pointer to the tensor description. The type of the pointed-to struct must match the value specified in *Type*.

Requirements

 [Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

DML_TILE_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Constructs an output tensor by tiling the input tensor. The elements in each dimension of the input tensor are repeated by a multiple in the *Repeats* array.

Syntax

C++

```
struct DML_TILE_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 RepeatsCount;
    const UINT           *Repeats;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The tensor to read from, which contains the elements to be tiled.

OutputTensor

Type: **const DML_TENSOR_DESC***

The tensor to write to, which will hold the tiled output. For each dimension *i* in [0, *InputTensor.DimensionCount-1*], the output size is calculated as *OutputTensor.Sizes[i] = InputTensor.Sizes[i] * Repeats[i]*. This tensor must have the same *DimensionCount* as the input tensor.

RepeatsCount

Type: **UINT**

This field determines the size of the *Repeats* array. This value must be the same as the *InputTensor.DimensionCount*.

Repeats

Type: **const UINT***

Each value in this array corresponds to one of the input tensor's dimensions (in order). Each value is the number of tiled copies to make of that dimension. Values must be larger than 0.

Examples

```
RepeatsCount: 4
Repeats: {1, 1, 3, 3}

InputTensor: (Sizes:{1, 1, 2, 3}, DataType:FLOAT32)
[[[[1, 2, 3]
 [4, 5, 6]]]

InputTensor: (Sizes:{1, 1, 6, 9}, DataType:FLOAT32)
[[[[1, 2, 3, 1, 2, 3, 1, 2, 3]
 [4, 5, 6, 4, 5, 6, 4, 5, 6]
 [1, 2, 3, 1, 2, 3, 1, 2, 3]
 [4, 5, 6, 4, 5, 6, 4, 5, 6]
 [1, 2, 3, 1, 2, 3, 1, 2, 3]
 [4, 5, 6, 4, 5, 6, 4, 5, 6]]]
```

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType*.

Tensor support

`DML_FEATURE_LEVEL_4_1` and above

 Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

DML_FEATURE_LEVEL_1_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

 Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_TOP_K_OPERATOR_DESC structure (directml.h)

Article01/21/2022

Selects the largest K elements from each sequence along an axis of the *InputTensor*, and returns the values and indices of those elements in the *OutputValueTensor* and *OutputIndexTensor*, respectively. A *sequence* refers to one of the sets of elements that exist along the *Axis* dimension of the *InputTensor*.

Syntax

C++

```
struct DML_TOP_K_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputValueTensor;
    const DML_TENSOR_DESC *OutputIndexTensor;
    UINT                 Axis;
    UINT                 K;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor containing elements to select.

OutputValueTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the values of the top K elements to. This tensor must have sizes equal to the *InputTensor*, except for the dimension specified by the *Axis* parameter, which must have a size equal to K .

The K values selected from each input sequence are guaranteed to be sorted descending (largest to smallest).

OutputIndexTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the indices of the top K elements to. This tensor must have sizes equal to the *InputTensor*, except for the dimension specified by the *Axis* parameter, which must have a size equal to K .

The indices returned in this tensor are measured relative to the beginning of their sequence (as opposed to the beginning of the tensor). For example, an index of 0 always refers to the first element for all sequences in an axis.

In cases where two or more elements in the top- K have the same value (that is, when there is a tie), the indices of both elements are included, and are guaranteed to be ordered by ascending element index.

Axis

Type: **UINT**

The index of the dimension to select elements across. This value must be less than the *DimensionCount* of the *InputTensor*.

K

Type: **UINT**

The number of elements to select. K must be greater than 0, but less than the number of elements in the *InputTensor* along the dimension specified by *Axis*.

Examples

Example 1

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[ 0,  1, 10, 11],
   [ 3,  2,  9,  8],
   [ 4,  5,  6,  7]]]]
```

```
Axis: 3
K:    2
```

```
OutputValueTensor: (Sizes:{1,1,3,2}, DataType:FLOAT32)
[[[11, 10],
  [ 9,  8],
  [ 7,  6]]]]
```

```
OutputIndexTensor: (Sizes:{1,1,3,2}, DataType:UINT32)
[[[[3, 2],
   [2, 3],
   [3, 2]]]]
```

Example 2. Using a different axis

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[ 0,  1, 10, 11],
   [ 3,  2,  9,  8],
   [ 4,  5,  6,  7]]]]
```

Axis: 2
K: 2

```
OutputValueTensor: (Sizes:{1,1,2,4}, DataType:FLOAT32)
[[[[ 4,  5, 10, 11],
   [ 3,  2,  9,  8]]]]
```

```
OutputIndexTensor: (Sizes:{1,1,2,4}, DataType:UINT32)
[[[[2, 2, 0, 0],
   [1, 1, 1, 1]]]]
```

Example 3. Tied values

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[1, 2, 2, 3],
   [3, 4, 5, 5],
   [6, 6, 6, 6]]]]
```

Axis: 3
K: 3

```
OutputValueTensor: (Sizes:{1,1,3,3}, DataType:FLOAT32)
[[[[3, 2, 2],
   [5, 5, 4],
   [6, 6, 6]]]]
```

```
OutputIndexTensor: (Sizes:{1,1,3,3}, DataType:UINT32)
[[[[3, 1, 2],
   [2, 3, 1],
   [0, 1, 2]]]]
```

Remarks

A newer version of this operator, [DML_TOP_K1_OPERATOR_DESC](#), was introduced in [DML_FEATURE_LEVEL_2_1](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_0](#).

Tensor constraints

- *InputTensor*, *OutputIndexTensor*, and *OutputValueTensor* must have the same *DimensionCount*.
- *InputTensor* and *OutputValueTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputValueTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputIndexTensor	Output	1 to 8	UINT64, UINT32

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8

Tensor	Kind	Supported dimension counts	Supported data types
OutputValueTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputIndexTensor	Output	1 to 8	UINT32

DML_FEATURE_LEVEL_2_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputValueTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputIndexTensor	Output	4	UINT32

DML_FEATURE_LEVEL_2_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputValueTensor	Output	4	FLOAT32, FLOAT16
OutputIndexTensor	Output	4	UINT32

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_TOP_K1_OPERATOR_DESC structure (directml.h)

Article01/21/2022

Selects the largest or smallest K elements from each sequence along an axis of the *InputTensor*, and returns the values and indices of those elements in the *OutputValueTensor* and *OutputIndexTensor*, respectively. A *sequence* refers to one of the sets of elements that exist along the *Axis* dimension of the *InputTensor*.

The choice of whether to select the largest K elements or the smallest K elements can be controlled using *AxisDirection*.

Syntax

C++

```
struct DML_TOP_K1_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputValueTensor;
    const DML_TENSOR_DESC *OutputIndexTensor;
    UINT                 Axis;
    UINT                 K;
    DML_AXIS_DIRECTION   AxisDirection;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

The input tensor containing elements to select.

OutputValueTensor

Type: **const DML_TENSOR_DESC***

The output tensor to write the values of the top K elements to. The top K elements are selected based on whether they are the largest or the smallest, depending on the value of *AxisDirection*. This tensor must have sizes equal to the *InputTensor*, except for the dimension specified by the *Axis* parameter, which must have a size equal to K .

The K values selected from each input sequence are guaranteed to be sorted descending (largest to smallest) if *AxisDirection* is [DML_AXIS_DIRECTION_DECREASING](#). Otherwise, the opposite is true and the values selected are guaranteed to be sorted ascending (smallest to largest).

`OutputIndexTensor`

Type: [const DML_TENSOR_DESC*](#)

The output tensor to write the indices of the top K elements to. This tensor must have sizes equal to the *InputTensor*, except for the dimension specified by the *Axis* parameter, which must have a size equal to K .

The indices returned in this tensor are measured relative to the beginning of their sequence (as opposed to the beginning of the tensor). For example, an index of 0 always refers to the first element for all sequences in an axis.

In cases where two or more elements in the top- K have the same value (that is, when there is a tie), the indices of both elements are included, and are guaranteed to be ordered by ascending element index. Note that this is true irrespective of the value of *AxisDirection*.

`Axis`

Type: [UINT](#)

The index of the dimension to select elements across. This value must be less than the *DimensionCount* of the *InputTensor*.

`K`

Type: [UINT](#)

The number of elements to select. K must be greater than 0, but less than the number of elements in the *InputTensor* along the dimension specified by *Axis*.

`AxisDirection`

Type: [DML_AXIS_DIRECTION](#)

A value from the [DML_AXIS_DIRECTION](#) enumeration. If set to [DML_AXIS_DIRECTION_INCREASING](#), then this operator returns the *smallest* K elements in order of increasing value. Otherwise, it returns the *largest* K elements in decreasing order.

Examples

Example 1

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[ 0,  1, 10, 11],
   [ 3,  2,  9,  8],
   [ 4,  5,  6,  7]]]]

Axis: 3
K:    2
AxisDirection: DML_AXIS_DIRECTION_DECREASING

OutputValueTensor: (Sizes:{1,1,3,2}, DataType:FLOAT32)
[[[[11, 10],
   [ 9,  8],
   [ 7,  6]]]]

OutputIndexTensor: (Sizes:{1,1,3,2}, DataType:UINT32)
[[[[3, 2],
   [2, 3],
   [3, 2]]]]
```

Example 2. Using a different axis

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[[ 0,  1, 10, 11],
   [ 3,  2,  9,  8],
   [ 4,  5,  6,  7]]]]

Axis: 2
K:    2
AxisDirection: DML_AXIS_DIRECTION_DECREASING

OutputValueTensor: (Sizes:{1,1,2,4}, DataType:FLOAT32)
[[[[ 4,  5, 10, 11],
   [ 3,  2,  9,  8]]]]

OutputIndexTensor: (Sizes:{1,1,2,4}, DataType:UINT32)
[[[[2, 2, 0, 0],
   [1, 1, 1, 1]]]]
```

Example 3. Tied values

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[1, 2, 2, 3],
 [3, 4, 5, 5],
 [6, 6, 6, 6]]]

Axis: 3
K: 3
AxisDirection: DML_AXIS_DIRECTION_DECREASING

OutputValueTensor: (Sizes:{1,1,3,3}, DataType:FLOAT32)
[[[3, 2, 2],
 [5, 5, 4],
 [6, 6, 6]]]

OutputIndexTensor: (Sizes:{1,1,3,3}, DataType:UINT32)
[[[3, 1, 2],
 [2, 3, 1],
 [0, 1, 2]]]
```

Example 4. Increasing axis direction

```
InputTensor: (Sizes:{1,1,3,4}, DataType:FLOAT32)
[[[1, 2, 2, 3],
 [3, 4, 5, 5],
 [6, 6, 6, 6]]]

Axis: 3
K: 3
AxisDirection: DML_AXIS_DIRECTION_INCREASING

OutputValueTensor: (Sizes:{1,1,3,3}, DataType:FLOAT32)
[[[1, 2, 2],
 [3, 4, 5],
 [6, 6, 6]]]

OutputIndexTensor: (Sizes:{1,1,3,3}, DataType:UINT32)
[[[0, 1, 2],
 [0, 1, 2],
 [0, 1, 2]]]
```

Remarks

When *AxisDirection* is set to [DML_AXIS_DIRECTION_DECREASING](#), this operator is equivalent to [DML_TOP_K_OPERATOR_DESC](#).

Availability

This operator was introduced in [DML_FEATURE_LEVEL_2_1](#).

Tensor constraints

- *InputTensor*, *OutputIndexTensor*, and *OutputValueTensor* must have the same *DimensionCount*.
- *InputTensor* and *OutputValueTensor* must have the same *DataType*.

Tensor support

DML_FEATURE_LEVEL_5_0 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputValueTensor	Output	1 to 8	FLOAT32, FLOAT16, INT64, INT32, INT16, INT8, UINT64, UINT32, UINT16, UINT8
OutputIndexTensor	Output	1 to 8	UINT64, UINT32

DML_FEATURE_LEVEL_3_1 and above

[\[+\] Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputValueTensor	Output	1 to 8	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputIndexTensor	Output	1 to 8	UINT32

DML_FEATURE_LEVEL_2_1 and above

[Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputValueTensor	Output	4	FLOAT32, FLOAT16, INT32, INT16, INT8, UINT32, UINT16, UINT8
OutputIndexTensor	Output	4	UINT32

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_UNFOLD_OPERATOR_DESC structure (directml.h)

Article • 02/10/2025

Extracts sliding local blocks from a batched input tensor.

Consider a InputTensor with shape $(N, C, *)$, where $*$ is an arbitrary number of spatial dimensions (≤ 6). The operation flattens each sliding WindowSize'd block within the InputTensor into a column of an OutputTensor. The OutputTensor is a 3D tensor containing sliding local blocks, e.g., patches of images, of shape $(N, C \times \prod(\text{WindowSizes}), \text{BlockCount})$, where N is batch dimension, $C \times \prod(\text{WindowSizes})$ is the number of values within a window (a window has $\prod(\text{WindowSizes})$ spatial locations each containing a C -channeled vector), and BlockCount is the total number of blocks. The arguments must satisfy:

$$\text{BlocksPerDimension}[d] = ((\text{SpatialSize}[d] + \text{StartPadding}[d] + \text{EndPadding}[d] - \text{Dilations}[d] * (\text{WindowSize}[d] - 1) - 1) / \text{stride}[d]) + 1$$

$$\text{BlockCount} = \prod_d \text{BlocksPerDimension}[d]$$

Where:

- SpatialSize is formed by the spatial dimensions of input (*)
- $0 \leq d < \text{DimensionCount}$

Indexing output at the last dimension (column dimension) gives all values within a certain block.

The StartPadding, EndPadding, Strides, and Dilations arguments specify how the sliding blocks are retrieved.

ⓘ Important

This API is available as part of the DirectML standalone redistributable package (see [Microsoft.AI.DirectML](#) version 1.15.1 and later. Also see [DirectML version history](#).

Syntax

C++

```
struct DML_UNFOLD_OPERATOR_DESC
{
    const DML_TENSOR_DESC* InputTensor;
    const DML_TENSOR_DESC* OutputTensor;
    UINT DimensionCount;
    _Field_size_(DimensionCount) const UINT* WindowSizes;
    _Field_size_(DimensionCount) const UINT* Strides;
    _Field_size_(DimensionCount) const UINT* Dilations;
    _Field_size_(DimensionCount) const UINT* StartPadding;
    _Field_size_(DimensionCount) const UINT* EndPadding;
};
```

Members

`InputTensor`

Type: **const DML_TENSOR_DESC***

The input tensor to read from.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

The output tensor to write the results to.

`DimensionCount`

Type: **UINT**

The spatial dimensions of *InputTensor*. *DimensionCount* must be in the range [1, 6].

`WindowSizes`

Type: *_Field_size_(DimensionCount) const UINT**

The size of the sliding window.

`Strides`

Type: *_Field_size_(DimensionCount) const UINT**

The stride of the sliding window (with dimensions *WindowSizes*) in the input spatial dimensions. They are separate from the tensor strides included in **DML_TENSOR_DESC**. Step size of the extracted patches.

`Dilations`

Type: `_Field_size_(DimensionCount) const UINT*`

The dilations of the sliding window (with dimensions `WindowSizes`) in the input spatial dimensions, by scaling the space between the kernel points. Dilations of the extracted patch.

`StartPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the amount of implicit zero-padding to be applied to the beginning of each spatial dimension of `InputTensor`. Start padding of the source tensor.

`EndPadding`

Type: `_Field_size_(DimensionCount) const UINT*`

An array containing the amount of implicit zero-padding to be applied to the end of each spatial dimension of `InputTensor`. End padding of the source tensor.

Examples

Example 1.

1-channel unfold.

```
InputTensor: (Sizes:{1, 1, 5, 5}, DataType:FLOAT32)
[[[ [ 0., 1., 2., 3., 4.],
      [ 5., 6., 7., 8., 9.],
      [10., 11., 12., 13., 14.],
      [15., 16., 17., 18., 19.],
      [20., 21., 22., 23., 24.] ]]]
DimensionCount: 2
WindowSizes: {3, 3}
Strides: {1, 1}
Dilations: {1, 1}
StartPadding: {0, 0}
EndPadding: {0, 0}
OutputTensor: (Sizes:{1, 9, 9}, DataType:FLOAT32)
[[[ [ 0., 1., 2., 5., 6., 7., 10., 11., 12.],
      [ 1., 2., 3., 6., 7., 8., 11., 12., 13.],
      [ 2., 3., 4., 7., 8., 9., 12., 13., 14.],
      [ 5., 6., 7., 10., 11., 12., 15., 16., 17.],
      [ 6., 7., 8., 11., 12., 13., 16., 17., 18.],
      [ 7., 8., 9., 12., 13., 14., 17., 18., 19.],
      [10., 11., 12., 15., 16., 17., 20., 21., 22.] ]]]
```

```
[11., 12., 13., 16., 17., 18., 21., 22., 23.],  
[12., 13., 14., 17., 18., 19., 22., 23., 24.]]]
```

Example 2.

1-channel, padded fold.

```
InputTensor: (Sizes:{1, 1, 5, 5}, DataType:FLOAT32)  
[[[[ 0., 1., 2., 3., 4.],  
   [ 5., 6., 7., 8., 9.],  
   [10., 11., 12., 13., 14.],  
   [15., 16., 17., 18., 19.],  
   [20., 21., 22., 23., 24.]]]]  
DimensionCount: 2  
WindowSizes: {3, 3}  
Strides: {1, 1}  
Dilations: {1, 1}  
StartPadding: {1, 0}  
EndPadding: {1, 0}  
OutputTensor: (Sizes:{1, 9, 15}, DataType:FLOAT32)  
[[[ 0., 0., 0., 1., 2., 5., 6., 7., 10., 11., 12., 15., 16.,  
 17.],  
  [ 0., 0., 0., 1., 2., 3., 6., 7., 8., 11., 12., 13., 16., 17.,  
 18.],  
  [ 0., 0., 0., 2., 3., 4., 7., 8., 9., 12., 13., 14., 17., 18.,  
 19.],  
  [ 0., 1., 2., 5., 6., 7., 10., 11., 12., 15., 16., 17., 20., 21.,  
 22.],  
  [ 1., 2., 3., 6., 7., 8., 11., 12., 13., 16., 17., 18., 21., 22.,  
 23.],  
  [ 2., 3., 4., 7., 8., 9., 12., 13., 14., 17., 18., 19., 22., 23.,  
 24.],  
  [ 5., 6., 7., 10., 11., 12., 15., 16., 17., 20., 21., 22., 0., 0.,  
 0.],  
  [ 6., 7., 8., 11., 12., 13., 16., 17., 18., 21., 22., 23., 0., 0.,  
 0.],  
  [ 7., 8., 9., 12., 13., 14., 17., 18., 19., 22., 23., 24., 0., 0.,  
 0.]]]
```

Availability

This operator was introduced in [DML_FEATURE_LEVEL_6_4](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DimensionCount*.

Tensor support

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	3 to 8	
OutputTensor	Output	3 to 8	

Requirements

[+] Expand table

Header	directml.h
--------	------------

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_UPSAMPLE_2D_OPERATOR_DESC structure (directml.h)

Article 10/05/2021

Upsamples the input image, writing the result into the output tensor. The order of the dimensions should be NCHW (BatchSize, ChannelCount, Height, Width) or NCDHW (BatchSize, ChannelCount, Depth, Height, Width), but strides can be used if the data is stored in a different format. Unlike [DML_RESAMPLE_OPERATOR_DESC](#), only the last 2 dimensions (height and width) can be upsampled.

If available, you should prefer [DML_RESAMPLE_OPERATOR_DESC](#) since it is a more flexible version of [DML_UPSAMPLE_2D_OPERATOR_DESC](#).

Syntax

C++

```
struct DML_UPSAMPLE_2D_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_SIZE_2D ScaleSize;
    DML_INTERPOLATION_MODE InterpolationMode;
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

A tensor containing the input data. The expected dimensions of the InputTensor are {
`InputBatchCount, InputChannelCount, InputHeight, InputWidth`} for 4D, and {
`InputBatchCount, InputChannelCount, InputDepth, InputHeight, InputWidth`} for 5D.

`OutputTensor`

Type: [const DML_TENSOR_DESC*](#)

A tensor containing the input data. The expected dimensions of the OutputTensor are {
`InputBatchCount, InputChannelCount, InputHeight * HeightScale, InputWidth * WidthScale`}

`WidthScale }` for 4D, and `{ InputBatchCount, InputChannelCount, InputDepth, InputHeight * HeightScale, InputWidth * WidthScale }` for 5D.

ScaleSize

Type: [DML_SIZE_2D](#)

The width and height scales of type `UINT` to apply when upsampling the input. `0 < ScaleSize.Height <= UINT_MAX / InputHeight` and `0 < ScaleSize.Width <= UINT_MAX / InputWidth`.

InterpolationMode

Type: [DML_INTERPOLATION_MODE](#)

This field determines the kind of interpolation used to choose output pixels.

- [DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR](#). Uses the *Nearest Neighbor* algorithm, which chooses the input element nearest to the corresponding pixel center for each output element.
- [DML_INTERPOLATION_MODE_LINEAR](#). Uses the *Bilinear* algorithm, which computes the output element by doing the weighted average of the 2 nearest neighboring input elements in the height dimension, and the 2 nearest neighboring input elements in the width dimension, for a total of 4 elements. This is true even if the input/output `DimensionCount` is 5. That is, samples are only ever averaged along the width and height dimensions, and never along the batch, channel, or depth.

Availability

This operator was introduced in [DML_FEATURE_LEVEL_1_0](#).

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *DimensionCount*.

Tensor support

[+] [Expand table](#)

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4 to 5	FLOAT32, FLOAT16
OutputTensor	Output	4 to 5	FLOAT32, FLOAT16

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_VALUE_SCALE_2D_OPERATOR_DESC structure (directml.h)

Article 02/22/2024

Performs an element-wise scale-and-bias function, `Output = Scale * Input + Bias`. This operator is similar to using an [DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC](#) with a scale and bias, except that `DML_VALUE_SCALE_2D_OPERATOR_DESC` applies a different bias for each channel, rather than a single bias for the entire tensor.

Syntax

C++

```
struct DML_VALUE_SCALE_2D_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Scale;
    UINT ChannelCount;
    const FLOAT *Bias;
};
```

Members

`InputTensor`

Type: [const DML_TENSOR_DESC*](#)

A tensor containing the input data. This tensor's dimensions should be `{ BatchCount, ChannelCount, Height, Width }`.

`OutputTensor`

Type: [const DML_TENSOR_DESC*](#)

A tensor with which to write the results to. This tensor's dimensions should match the `InputTensor`'s dimensions.

`Scale`

Type: [FLOAT](#)

Scale value to be applied to all input values.

ChannelCount

Type: **UINT**

This field determines the size of the Bias array. This field must be set to either 1 or 3, and must also match the size of the Channel dimension of the input tensor.

Bias

Type: **const FLOAT***

An array of *FLOAT* values containing the bias term for each dimension of the input tensor.

Availability

This operator was introduced in `DML_FEATURE_LEVEL_1_0`.

Tensor constraints

InputTensor and *OutputTensor* must have the same *DataType* and *Sizes*.

Tensor support

[+] Expand table

Tensor	Kind	Supported dimension counts	Supported data types
InputTensor	Input	4	FLOAT32, FLOAT16
OutputTensor	Output	4	FLOAT32, FLOAT16

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DirectML enumerations

Article • 02/10/2025

The following enumerations are declared in `DirectML.h`.

In this section

[] [Expand table](#)

Topic and description
DML_AXIS_DIRECTION . Defines constants that specify the direction of an operation along the given axis for the operator (for example, summation, selecting the top-k elements, selecting the minimum element).
DML_BINDING_TYPE . Defines constants that specify the nature of the resource(s) referred to by a binding description DML_BINDING_DESC structure .
DML_CONVOLUTION_DIRECTION . Defines constants that specify a direction for the DirectML convolution operator (as described by the DML_CONVOLUTION_OPERATOR_DESC structure).
DML_CONVOLUTION_MODE . Defines constants that specify a mode for the DirectML convolution operator (as described by the DML_CONVOLUTION_OPERATOR_DESC structure).
DML_CREATE_DEVICE_FLAGS . Supplies additional device creation options to DMLCreateDevice .
DML_DEPTH_SPACE_ORDER . Defines constants controlling the transform applied in the DirectML operators DML_OPERATOR_DEPTH_TO_SPACE1 and DML_OPERATOR_SPACE_TO_DEPTH1 .
DML_EXECUTION_FLAGS . Supplies options to DirectML to control execution of operators.
DML_FEATURE . Defines a set of optional features and capabilities that can be queried from the DirectML device.
DML_GRAPH_EDGE_TYPE . Defines constants that specify a type of graph edge. See DML_GRAPH_EDGE_DESC for the usage of this enumeration.
DML_GRAPH_NODE_TYPE . Defines constants that specify a type of graph node. See DML_GRAPH_NODE_DESC for the usage of this enumeration.
DML_INTERPOLATION_MODE . Defines constants that specify a mode for the DirectML upsample 2-D operator (as described by the DML_UPSAMPLE_2D_OPERATOR_DESC structure).
DML_MATRIX_TRANSFORM . Defines constants that specify a matrix transform to be applied to a DirectML tensor.
DML_OPERATOR_TYPE . Defines the type of an operator description.

Topic and description
DML_PADDING_MODE . Defines constants that specify a mode for the DirectML pad operator (as described by the DML_PADDING_OPERATOR_DESC structure).
DML_RANDOM_GENERATOR_TYPE . Defines constants that specify types of random random-number generator.
DML_RECURRENT_NETWORK_DIRECTION . Defines constants that specify a direction for a recurrent DirectML operator.
DML_REDUCE_FUNCTION . Defines constants that specify the specific reduction algorithm to use for the DirectML reduce operator (as described by the DML_REDUCE_OPERATOR_DESC structure).
DML_TENSOR_DATA_TYPE . Specifies the data type of the values in a tensor.
DML_TENSOR_FLAGS . Specifies additional options in a tensor description.
DML_TENSOR_TYPE . Identifies a type of tensor description.

Related topics

- [DirectML reference](#)
- [Windows AI](#)
- [Core reference](#)
- [Direct3D 12 Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_AXIS_DIRECTION enumeration (directml.h)

Article02/22/2024

Defines constants that specify the direction of an operation along the given axis for the operator (for example, summation, selecting the top-k elements, selecting the minimum element).

Syntax

C++

```
typedef enum DML_AXIS_DIRECTION {
    DML_AXIS_DIRECTION_INCREASING = 0,
    DML_AXIS_DIRECTION_DECREASING = 1
};
```

Constants

[] Expand table

	<code>DML_AXIS_DIRECTION_INCREASING</code>
Value:	0
	Specifies increasing order (from the low index to the high index).
	<code>DML_AXIS_DIRECTION_DECREASING</code>
Value:	1
	Specifies decreasing order (from the high index to the low index).

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_BINDING_TYPE enumeration (directml.h)

Article 02/22/2024

Defines constants that specify the nature of the resource(s) referred to by a binding description (a [DML_BINDING_DESC](#) structure).

Syntax

C++

```
typedef enum DML_BINDING_TYPE {
    DML_BINDING_TYPE_NONE,
    DML_BINDING_TYPE_BUFFER,
    DML_BINDING_TYPE_BUFFER_ARRAY
};
```

Constants

[\[+\] Expand table](#)

[DML_BINDING_TYPE_NONE](#)

Indicates that no resources are to be bound.

[DML_BINDING_TYPE_BUFFER](#)

Specifies a binding that binds a single buffer to the binding table. The corresponding binding desc type is [DML_BUFFER_BINDING](#).

[DML_BINDING_TYPE_BUFFER_ARRAY](#)

Specifies a binding that binds an array of buffers to the binding table. The corresponding binding desc type is [DML_BUFFER_ARRAY_BINDING](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	directml.h

See also

[Binding in DirectML, DML_BINDING_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_CONVOLUTION_DIRECTION enumeration (directml.h)

Article02/22/2024

Defines constants that specify a direction for the DirectML convolution operator (as described by the [DML_CONVOLUTION_OPERATOR_DESC](#) structure).

Syntax

C++

```
typedef enum DML_CONVOLUTION_DIRECTION {
    DML_CONVOLUTION_DIRECTION_FORWARD,
    DML_CONVOLUTION_DIRECTION_BACKWARD
} ;
```

Constants

[+] [Expand table](#)

`DML_CONVOLUTION_DIRECTION_FORWARD`

Indicates a forward convolution.

`DML_CONVOLUTION_DIRECTION_BACKWARD`

Indicates a backward convolution. Backward convolution is also known as *transposed convolution*.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_CONVOLUTION_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_CONVOLUTION_MODE enumeration (directml.h)

Article02/22/2024

Defines constants that specify a mode for the DirectML convolution operator (as described by the [DML_CONVOLUTION_OPERATOR_DESC](#) structure).

Syntax

C++

```
typedef enum DML_CONVOLUTION_MODE {
    DML_CONVOLUTION_MODE_CONVOLUTION,
    DML_CONVOLUTION_MODE_CROSS_CORRELATION
} ;
```

Constants

[] [Expand table](#)

`DML_CONVOLUTION_MODE_CONVOLUTION`

Specifies the convolution mode. When used along with [DML_CONVOLUTION_DIRECTION_FORWARD](#), this flips the filter along the height and width axes.

`DML_CONVOLUTION_MODE_CROSS_CORRELATION`

Specifies the cross-correlation mode. If in doubt, use this mode—it is appropriate for the vast majority of machine learning (ML) model inference. When used along with [DML_CONVOLUTION_DIRECTION_BACKWARD](#), this flips the filter along the height and width axes.

Requirements

[] [Expand table](#)

Requirement	Value
Header	directml.h

See also

[DML_CONVOLUTION_OPERATOR_DESC](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_CREATE_DEVICE_FLAGS enumeration (directml.h)

Article02/22/2024

Supplies additional device creation options to [DMLCreateDevice](#). Values can be bitwise OR'd together.

Syntax

C++

```
typedef enum DML_CREATE_DEVICE_FLAGS {
    DML_CREATE_DEVICE_FLAG_NONE = 0,
    DML_CREATE_DEVICE_FLAG_DEBUG = 0x1
};
```

Constants

[+] [Expand table](#)

`DML_CREATE_DEVICE_FLAG_NONE`

Value: `0`

No creation options are specified.

`DML_CREATE_DEVICE_FLAG_DEBUG`

Value: `0x1`

Enables the DirectML debug layers. To use the debug layers, developer mode must be enabled, and the DirectML debug layers must be installed. If the `DML_CREATE_DEVICE_FLAG_DEBUG` flag is specified and either condition is not met, then [DMLCreateDevice](#) returns `DXGI_ERROR_SDK_COMPONENT_MISSING`.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_DEPTH_SPACE_ORDER enumeration (directml.h)

Article 10/20/2021

Defines constants controlling the transform applied in the DirectML operators [DML_OPERATOR_DEPTH_TO_SPACE1](#) and [DML_OPERATOR_SPACE_TO_DEPTH1](#). These are used within the [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#) and [DML_SPACE_TO_DEPTH1_OPERATOR_DESC](#) structures.

Syntax

C++

```
typedef enum DML_DEPTH_SPACE_ORDER {
    DML_DEPTH_SPACE_ORDER_DEPTH_COLUMN_ROW,
    DML_DEPTH_SPACE_ORDER_COLUMN_ROW_DEPTH
} ;
```

Constants

[+] Expand table

[DML_DEPTH_SPACE_ORDER_DEPTH_COLUMN_ROW](#)

Causes tensors used in [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#) and [DML_SPACE_TO_DEPTH1_OPERATOR_DESC](#) to be interpreted with the following layouts, where dimensions in parenthesis are flattened together.

- **Depth version:** [Batch, (BlockHeight, BlockWidth, Channels), Height, Width]
- **Space version:** [Batch, Channels, (Height, BlockHeight), (Width, BlockWidth)]

[DML_DEPTH_SPACE_ORDER_COLUMN_ROW_DEPTH](#)

Causes tensors used in [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#) and [DML_SPACE_TO_DEPTH1_OPERATOR_DESC](#) to be interpreted with the following layouts, where dimensions in parenthesis are flattened together.

- **Depth version:** [Batch, (Channels, BlockHeight, BlockWidth), Height, Width]
- **Space version:** [Batch, Channels, (Height, BlockHeight), (Width, BlockWidth)]

Remarks

See [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#) and [DML_SPACE_TO_DEPTH1_OPERATOR_DESC](#) documentation for examples showing the effect of these values.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [DML_DEPTH_TO_SPACE1_OPERATOR_DESC](#)
- [DML_SPACE_TO_DEPTH1_OPERATOR_DESC](#)

Availability

This enumeration was introduced in [DML_FEATURE_LEVEL_2_1](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_EXECUTION_FLAGS enumeration (directml.h)

Article02/22/2024

Supplies options to DirectML to control execution of operators. These flags can be bitwise OR'd together to specify multiple flags at once.

Syntax

C++

```
typedef enum DML_EXECUTION_FLAGS {
    DML_EXECUTION_FLAG_NONE = 0,
    DML_EXECUTION_FLAG_ALLOW_HALF_PRECISION_COMPUTATION = 0x1,
    DML_EXECUTION_FLAG_DISABLE_META_COMMANDS = 0x2,
    DML_EXECUTION_FLAG_DESCRIPTOR_VOLATILE = 0x4
};
```

Constants

 Expand table

<code>DML_EXECUTION_FLAG_NONE</code>	Value: <i>0</i> No execution flags are specified.
<code>DML_EXECUTION_FLAG_ALLOW_HALF_PRECISION_COMPUTATION</code>	Value: <i>0x1</i> Allows DirectML to perform computation using half-precision floating-point (FP16), if supported by the hardware device.
<code>DML_EXECUTION_FLAG_DISABLE_META_COMMANDS</code>	Value: <i>0x2</i> Forces DirectML execute the operator using DirectCompute instead of meta commands. DirectML uses meta commands by default, if available.
<code>DML_EXECUTION_FLAG_DESCRIPTOR_VOLATILE</code>	Value: <i>0x4</i> Allows changes to bindings after an operator's execution has been recorded in a command list, but before it has been submitted to the command queue. By default, without this flag set, you must set all bindings on the binding table before you record an operator into a command list.

This flag allows you to perform late binding—that is, to set (or to change) bindings on operators that you've already recorded into a command list. However, this may result in a performance penalty on some hardware, as it prohibits drivers from promoting static descriptor accesses to root descriptor accesses.

For more info, see [DESCRIPTORS_VOLATILE](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

See also

[Binding in DirectML](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_FEATURE enumeration (directml.h)

Article 02/22/2024

Defines a set of optional features and capabilities that can be queried from the DirectML device. See [IDMLDevice::CheckFeatureSupport](#).

Syntax

C++

```
typedef enum DML_FEATURE {
    DML_FEATURE_TENSOR_DATA_TYPE_SUPPORT,
    DML_FEATURE_FEATURE_LEVELS
} ;
```

Constants

[+] [Expand table](#)

DML_FEATURE_TENSOR_DATA_TYPE_SUPPORT

Allows querying for tensor data type support. The query type is

[DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT](#), and the support data type is

[DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT](#).

DML_FEATURE_FEATURE_LEVELS

Allows querying for the feature levels supported by the device. The query type is

[DML_FEATURE_QUERY_FEATURE_LEVELS](#), and the support data type is

[DML_FEATURE_DATA_FEATURE_LEVELS](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

See also

- IDMLDevice::CheckFeatureSupport method
 - DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT
 - DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT
 - DML_FEATURE_QUERY_FEATURE_LEVELS
 - DML_FEATURE_DATA_FEATURE_LEVELS
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_GRAPH_EDGE_TYPE enumeration (directml.h)

Article 02/22/2024

Defines constants that specify a type of graph edge. See [DML_GRAPH_EDGE_DESC](#) for the usage of this enumeration.

Syntax

C++

```
typedef enum DML_GRAPH_EDGE_TYPE {
    DML_GRAPH_EDGE_TYPE_INVALID,
    DML_GRAPH_EDGE_TYPE_INPUT,
    DML_GRAPH_EDGE_TYPE_OUTPUT,
    DML_GRAPH_EDGE_TYPE_INTERMEDIATE
};
```

Constants

[] Expand table

DML_GRAPH_EDGE_TYPE_INVALID

Specifies an unknown graph edge type, and is never valid. Using this value results in an error.

DML_GRAPH_EDGE_TYPE_INPUT

Specifies that the graph edge is described by the [DML_INPUT_GRAPH_EDGE_DESC](#) structure.

DML_GRAPH_EDGE_TYPE_OUTPUT

Specifies that the graph edge is described by the [DML_OUTPUT_GRAPH_EDGE_DESC](#) structure.

DML_GRAPH_EDGE_TYPE_INTERMEDIATE

Specifies that the graph edge is described by the [DML_INTERMEDIATE_GRAPH_EDGE_DESC](#) structure.

Availability

This API was introduced in DirectML version **1.1.0**.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method](#)
- [DML_GRAPH_DESC structure](#)
- [DML_GRAPH_EDGE_DESC structure](#)
- [DML_INPUT_GRAPH_EDGE_DESC structure](#)
- [DML_OUTPUT_GRAPH_EDGE_DESC structure](#)
- [DML_INTERMEDIATE_GRAPH_EDGE_DESC structure](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_GRAPH_NODE_TYPE enumeration (directml.h)

Article 02/22/2024

Defines constants that specify a type of graph node. See [DML_GRAPH_NODE_DESC](#) for the usage of this enumeration.

Syntax

C++

```
typedef enum DML_GRAPH_NODE_TYPE {
    DML_GRAPH_NODE_TYPE_INVALID,
    DML_GRAPH_NODE_TYPE_OPERATOR,
    DML_GRAPH_NODE_TYPE_CONSTANT
};
```

Constants

[+] Expand table

`DML_GRAPH_NODE_TYPE_INVALID`

Specifies an unknown graph edge type, and is never valid. Using this value results in an error.

`DML_GRAPH_NODE_TYPE_OPERATOR`

Specifies that the graph edge is described by the [DML_OPERATOR_GRAPH_NODE_DESC](#) structure.

Availability

This API was introduced in DirectML version [1.1.0](#).

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348

Requirement	Value
Minimum supported server	Windows 10 Build 20348
Header	directml.h

See also

- [IDMLDevice1::CompileGraph method](#)
- [DML_GRAPH_DESC structure](#)
- [DML_GRAPH_NODE_DESC structure](#)
- [DML_OPERATOR_GRAPH_NODE_DESC](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_INTERPOLATION_MODE enumeration (directml.h)

Article02/22/2024

Defines constants that specify a mode for the DirectML upsample 2-D operator (as described by the [DML_UPSAMPLE_2D_OPERATOR_DESC](#) structure).

Syntax

C++

```
typedef enum DML_INTERPOLATION_MODE {
    DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR,
    DML_INTERPOLATION_MODE_LINEAR
} ;
```

Constants

[+] [Expand table](#)

`DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`

Specifies the nearest-neighbor mode.

`DML_INTERPOLATION_MODE_LINEAR`

Specifies a linear (including bilinear, trilinear, etc.) mode.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	directml.h

See also

- [DML_UPSAMPLE_2D_OPERATOR_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_MATRIX_TRANSFORM enumeration (directml.h)

Article02/22/2024

Defines constants that specify a matrix transform to be applied to a DirectML tensor.

Syntax

C++

```
typedef enum DML_MATRIX_TRANSFORM {
    DML_MATRIX_TRANSFORM_NONE,
    DML_MATRIX_TRANSFORM_TRANSPOSE
} ;
```

Constants

[] Expand table

DML_MATRIX_TRANSFORM_NONE Specifies that no transform is to be applied.
DML_MATRIX_TRANSFORM_TRANSPOSE Specifies that a transpose transform is to be applied.

Requirements

[] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_OPERATOR_TYPE enumeration (directml.h)

Article 08/22/2024

Defines the type of an operator description.

See [DML_OPERATOR_DESC](#) for the usage of this enumeration.

Syntax

C++

```
typedef enum DML_OPERATOR_TYPE {
    DML_OPERATOR_INVALID,
    DML_OPERATOR_ELEMENT_WISE_IDENTITY,
    DML_OPERATOR_ELEMENT_WISE_ABS,
    DML_OPERATOR_ELEMENT_WISE_ACOS,
    DML_OPERATOR_ELEMENT_WISE_ADD,
    DML_OPERATOR_ELEMENT_WISE_ASIN,
    DML_OPERATOR_ELEMENT_WISE_ATAN,
    DML_OPERATOR_ELEMENT_WISE_CEIL,
    DML_OPERATOR_ELEMENT_WISE_CLIP,
    DML_OPERATOR_ELEMENT_WISE_COS,
    DML_OPERATOR_ELEMENT_WISE_DIVIDE,
    DML_OPERATOR_ELEMENT_WISE_EXP,
    DML_OPERATOR_ELEMENT_WISE_FLOOR,
    DML_OPERATOR_ELEMENT_WISE_LOG,
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_AND,
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_EQUALS,
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN,
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN,
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_NOT,
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_OR,
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_XOR,
    DML_OPERATOR_ELEMENT_WISE_MAX,
    DML_OPERATOR_ELEMENT_WISE_MEAN,
    DML_OPERATOR_ELEMENT_WISE_MIN,
    DML_OPERATOR_ELEMENT_WISE_MULTIPLY,
    DML_OPERATOR_ELEMENT_WISE_POW,
    DML_OPERATOR_ELEMENT_WISE_CONSTANT_POW,
    DML_OPERATOR_ELEMENT_WISE_RECIP,
    DML_OPERATOR_ELEMENT_WISE_SIN,
    DML_OPERATOR_ELEMENT_WISE_SQRT,
    DML_OPERATOR_ELEMENT_WISE_SUBTRACT,
    DML_OPERATOR_ELEMENT_WISE_TAN,
    DML_OPERATOR_ELEMENT_WISE_THRESHOLD,
    DML_OPERATOR_ELEMENT_WISE_QUANTIZE_LINEAR,
    DML_OPERATOR_ELEMENT_WISE_DEQUANTIZE_LINEAR,
    DML_OPERATOR_ACTIVATION_ELU,
```

```
DML_OPERATOR_ACTIVATION_HARDMAX,  
DML_OPERATOR_ACTIVATION_HARD_SIGMOID,  
DML_OPERATOR_ACTIVATION_IDENTITY,  
DML_OPERATOR_ACTIVATION_LEAKY_RELU,  
DML_OPERATOR_ACTIVATION_LINEAR,  
DML_OPERATOR_ACTIVATION_LOG_SOFTMAX,  
DML_OPERATOR_ACTIVATION_PARAMETERIZED_RELU,  
DML_OPERATOR_ACTIVATION_PARAMETRIC_SOFTPLUS,  
DML_OPERATOR_ACTIVATION_RELU,  
DML_OPERATOR_ACTIVATION_SCALED_ELU,  
DML_OPERATOR_ACTIVATION_SCALED_TANH,  
DML_OPERATOR_ACTIVATION_SIGMOID,  
DML_OPERATOR_ACTIVATION_SOFTMAX,  
DML_OPERATOR_ACTIVATION_SOFTPLUS,  
DML_OPERATOR_ACTIVATION_SOFTSIGN,  
DML_OPERATOR_ACTIVATION_TANH,  
DML_OPERATOR_ACTIVATION_THRESHOLDED_RELU,  
DML_OPERATOR_CONVOLUTION,  
DML_OPERATOR_GEMM,  
DML_OPERATOR_REDUCE,  
DML_OPERATOR_AVERAGE_POOLING,  
DML_OPERATOR_LP_POOLING,  
DML_OPERATOR_MAX_POOLING,  
DML_OPERATOR_ROI_POOLING,  
DML_OPERATOR_SLICE,  
DML_OPERATOR_CAST,  
DML_OPERATOR_SPLIT,  
DML_OPERATOR_JOIN,  
DML_OPERATOR_PADDING,  
DML_OPERATOR_VALUE_SCALE_2D,  
DML_OPERATOR_UPSAMPLE_2D,  
DML_OPERATOR_GATHER,  
DML_OPERATOR_SPACE_TO_DEPTH,  
DML_OPERATOR_DEPTH_TO_SPACE,  
DML_OPERATOR_TILE,  
DML_OPERATOR_TOP_K,  
DML_OPERATOR_BATCH_NORMALIZATION,  
DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION,  
DML_OPERATOR_LOCAL_RESPONSE_NORMALIZATION,  
DML_OPERATOR_LP_NORMALIZATION,  
DML_OPERATOR_RNN,  
DML_OPERATOR_LSTM,  
DML_OPERATOR_GRU,  
DML_OPERATOR_ELEMENT_WISE_SIGN,  
DML_OPERATOR_ELEMENT_WISE_IS_NAN,  
DML_OPERATOR_ELEMENT_WISE_ERF,  
DML_OPERATOR_ELEMENT_WISE_SINH,  
DML_OPERATOR_ELEMENT_WISE_COSH,  
DML_OPERATOR_ELEMENT_WISE_TANH,  
DML_OPERATOR_ELEMENT_WISE_ASINH,  
DML_OPERATOR_ELEMENT_WISE_ACOSH,  
DML_OPERATOR_ELEMENT_WISE_ATANH,  
DML_OPERATOR_ELEMENT_WISE_IF,  
DML_OPERATOR_ELEMENT_WISE_ADD1,  
DML_OPERATOR_ACTIVATION_SHRINK,
```

```
DML_OPERATOR_MAX_POOLING1,  
DML_OPERATOR_MAX_UNPOOLING,  
DML_OPERATOR_DIAGONAL_MATRIX,  
DML_OPERATOR_SCATTER_ELEMENTS,  
DML_OPERATOR_SCATTER,  
DML_OPERATOR_ONE_HOT,  
DML_OPERATOR_RESAMPLE,  
DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_LEFT,  
DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_RIGHT,  
DML_OPERATOR_ELEMENT_WISE_ROUND,  
DML_OPERATOR_ELEMENT_WISE_IS_INFINITY,  
DML_OPERATOR_ELEMENT_WISE_MODULUS_TRUNCATE,  
DML_OPERATOR_ELEMENT_WISE_MODULUS_FLOOR,  
DML_OPERATOR_FILL_VALUE_CONSTANT,  
DML_OPERATOR_FILL_VALUE_SEQUENCE,  
DML_OPERATOR_CUMULATIVE_SUMMATION,  
DML_OPERATOR_REVERSE_SUBSEQUENCES,  
DML_OPERATOR_GATHER_ELEMENTS,  
DML_OPERATOR_GATHER_ND,  
DML_OPERATOR_SCATTER_ND,  
DML_OPERATOR_MAX_POOLING2,  
DML_OPERATOR_SLICE1,  
DML_OPERATOR_TOP_K1,  
DML_OPERATOR_DEPTH_TO_SPACE1,  
DML_OPERATOR_SPACE_TO_DEPTH1,  
DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION1,  
DML_OPERATOR_RESAMPLE1,  
DML_OPERATOR_MATRIX_MULTIPLY_INTEGER,  
DML_OPERATOR_QUANTIZED_LINEAR_MATRIX_MULTIPLY,  
DML_OPERATOR_CONVOLUTION_INTEGER,  
DML_OPERATOR_QUANTIZED_LINEAR_CONVOLUTION,  
DML_OPERATOR_ELEMENT_WISE_BIT_AND,  
DML_OPERATOR_ELEMENT_WISE_BIT_OR,  
DML_OPERATOR_ELEMENT_WISE_BIT_XOR,  
DML_OPERATOR_ELEMENT_WISE_BIT_NOT,  
DML_OPERATOR_ELEMENT_WISE_BIT_COUNT,  
DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL,  
DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL,  
DML_OPERATOR_ACTIVATION CELU,  
DML_OPERATOR_ACTIVATION RELU_GRAD,  
DML_OPERATOR_AVERAGE_POOLING_GRAD,  
DML_OPERATOR_MAX_POOLING_GRAD,  
DML_OPERATOR_RANDOM_GENERATOR,  
DML_OPERATOR_NONZERO_COORDINATES,  
DML_OPERATOR_RESAMPLE_GRAD,  
DML_OPERATOR_SLICE_GRAD,  
DML_OPERATOR_ADAM_OPTIMIZER,  
DML_OPERATOR_ARGMIN,  
DML_OPERATOR_ARGMAX,  
DML_OPERATOR_ROI_ALIGN,  
DML_OPERATOR_GATHER_ND1,  
DML_OPERATOR_ELEMENT_WISE_ATAN_YX,  
DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD,  
DML_OPERATOR_ELEMENT_WISE_DIFFERENCE_SQUARE,  
DML_OPERATOR_LOCAL_RESPONSE_NORMALIZATION_GRAD,
```

```
DML_OPERATOR_CUMULATIVE_PRODUCT,  
DML_OPERATOR_BATCH_NORMALIZATION_GRAD,  
DML_OPERATOR_ELEMENT_WISE_QUANTIZED_LINEAR_ADD,  
DML_OPERATOR_DYNAMIC_QUANTIZE_LINEAR,  
DML_OPERATOR_ROI_ALIGN1,  
DML_OPERATOR_ROI_ALIGN_GRAD,  
DML_OPERATOR_BATCH_NORMALIZATION_TRAINING,  
DML_OPERATOR_BATCH_NORMALIZATION_TRAINING_GRAD,  
DML_OPERATOR_ELEMENT_WISE_CLIP1,  
DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD1,  
DML_OPERATOR_PADDING1,  
DML_OPERATOR_ELEMENT_WISE_NEGATE,  
DML_OPERATOR_ACTIVATION_GELU,  
DML_OPERATOR_ACTIVATION_SOFTMAX1,  
DML_OPERATOR_ACTIVATION_LOG_SOFTMAX1,  
DML_OPERATOR_ACTIVATION_HARDMAX1,  
DML_OPERATOR_RESAMPLE2,  
DML_OPERATOR_RESAMPLE_GRAD1,  
DML_OPERATOR_DIAGONAL_MATRIX1,  
DML_OPERATOR_MULTIHEAD_ATTENTION,  
DML_OPERATOR_LP_POOLING1,  
DML_OPERATOR_AVERAGE_POOLING1,  
DML_OPERATOR_ACTIVATION_SWISH,  
DML_OPERATOR_ACTIVATION_HARD_SWISH,  
DML_OPERATOR_QUANTIZED_LINEAR_AVERAGE_POOLING,  
DML_OPERATOR_MATRIX_MULTIPLY_INTEGER_TO_FLOAT  
} ;
```

Constants

[+] [Expand table](#)

DML_OPERATOR_INVALID Indicates an unknown operator type, and is never valid. Using this value results in an error.
DML_OPERATOR_ELEMENT_WISE_IDENTITY Indicates the operator described by the DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ABS Indicates the operator described by the DML_ELEMENT_WISE_ABS_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ACOS Indicates the operator described by the DML_ELEMENT_WISE_ACOS_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ADD Indicates the operator described by the DML_ELEMENT_WISE_ADD_OPERATOR_DESC structure.

DML_OPERATOR_ELEMENT_WISE_ASIN

Indicates the operator described by the [DML_ELEMENT_WISE_ASIN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_ATAN

Indicates the operator described by the [DML_ELEMENT_WISE_ATAN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_CEIL

Indicates the operator described by the [DML_ELEMENT_WISE_CEIL_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_CLIP

Indicates the operator described by the [DML_ELEMENT_WISE_CLIP_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_COS

Indicates the operator described by the [DML_ELEMENT_WISE_COS_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_DIVIDE

Indicates the operator described by the [DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_EXP

Indicates the operator described by the [DML_ELEMENT_WISE_EXP_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_FLOOR

Indicates the operator described by the [DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOG

Indicates the operator described by the [DML_ELEMENT_WISE_LOG_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_AND

Indicates the operator described by the [DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_EQUALS

Indicates the operator described by the [DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN

Indicates the operator described by the
[DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN

Indicates the operator described by the
[DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_NOT

Indicates the operator described by the [DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_OR

Indicates the operator described by the [DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC](#)

structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_XOR

Indicates the operator described by the [DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_MAX

Indicates the operator described by the [DML_ELEMENT_WISE_MAX_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_MEAN

Indicates the operator described by the [DML_ELEMENT_WISE_MEAN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_MIN

Indicates the operator described by the [DML_ELEMENT_WISE_MIN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_MULTIPLY

Indicates the operator described by the [DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_POW

Indicates the operator described by the [DML_ELEMENT_WISE_POW_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_CONSTANT_POW

Indicates the operator described by the [DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_RECIP

Indicates the operator described by the [DML_ELEMENT_WISE_RECIP_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_SIN

Indicates the operator described by the [DML_ELEMENT_WISE_SIN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_SQRT

Indicates the operator described by the [DML_ELEMENT_WISE_SQRT_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_SUBTRACT

Indicates the operator described by the [DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_TAN

Indicates the operator described by the [DML_ELEMENT_WISE_TAN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_THRESHOLD

Indicates the operator described by the [DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_QUANTIZE_LINEAR

Indicates the operator described by the [DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_DEQUANTIZE_LINEAR`

Indicates the operator described by the [DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_ELU`

Indicates the operator described by the [DML_ACTIVATION_ELU_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_HARDMAX`

Indicates the operator described by the [DML_ACTIVATION_HARDMAX_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_HARD_SIGMOID`

Indicates the operator described by the [DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_IDENTITY`

Indicates the operator described by the [DML_ACTIVATION_IDENTITY_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_LEAKY_RELU`

Indicates the operator described by the [DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_LINEAR`

Indicates the operator described by the [DML_ACTIVATION_LINEAR_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_LOG_SOFTMAX`

Indicates the operator described by the [DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_PARAMETERIZED_RELU`

Indicates the operator described by the [DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_PARAMETRIC_SOFTPLUS`

Indicates the operator described by the [DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_RELU`

Indicates the operator described by the [DML_ACTIVATION_RELU_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_SCALED_ELU`

Indicates the operator described by the [DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_SCALED_TANH`

Indicates the operator described by the [DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_SIGMOID`

Indicates the operator described by the [DML_ACTIVATION_SIGMOID_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_SOFTMAX

Indicates the operator described by the [DML_ACTIVATION_SOFTMAX_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_SOFTPLUS

Indicates the operator described by the [DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_SOFTSIGN

Indicates the operator described by the [DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_TANH

Indicates the operator described by the [DML_ACTIVATION_TANH_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_THRESHOLDED_RELU

Indicates the operator described by the
[DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC](#) structure.

DML_OPERATOR_CONVOLUTION

Indicates the operator described by the [DML_CONVOLUTION_OPERATOR_DESC](#) structure.

DML_OPERATOR_GEMM

Indicates the operator described by the [DML_GEMM_OPERATOR_DESC](#) structure.

DML_OPERATOR_REDUCE

Indicates the operator described by the [DML_REDUCE_OPERATOR_DESC](#) structure.

DML_OPERATOR_AVERAGE_POOLING

Indicates the operator described by the [DML_AVERAGE_POOLING_OPERATOR_DESC](#) structure.

DML_OPERATOR_LP_POOLING

Indicates the operator described by the [DML_LP_POOLING_OPERATOR_DESC](#) structure.

DML_OPERATOR_MAX_POOLING

Indicates the operator described by the [DML_MAX_POOLING_OPERATOR_DESC](#) structure.

DML_OPERATOR_ROI_POOLING

Indicates the operator described by the [DML_ROI_POOLING_OPERATOR_DESC](#) structure.

DML_OPERATOR_SLICE

Indicates the operator described by the [DML_SLICE_OPERATOR_DESC](#) structure.

DML_OPERATOR_CAST

Indicates the operator described by the [DML_CAST_OPERATOR_DESC](#) structure.

DML_OPERATOR_SPLIT

Indicates the operator described by the [DML_SPLIT_OPERATOR_DESC](#) structure.

DML_OPERATOR_JOIN

Indicates the operator described by the [DML_JOIN_OPERATOR_DESC](#) structure.

DML_OPERATOR_PADDING

Indicates the operator described by the [DML_PADDING_OPERATOR_DESC](#) structure.

DML_OPERATOR_VALUE_SCALE_2D

Indicates the operator described by the [DML_VALUE_SCALE_2D_OPERATOR_DESC](#) structure.

DML_OPERATOR_UPSAMPLE_2D

Indicates the operator described by the [DML_UPSAMPLE_2D_OPERATOR_DESC](#) structure.

DML_OPERATOR_GATHER

Indicates the operator described by the [DML_GATHER_OPERATOR_DESC](#) structure.

DML_OPERATOR_SPACE_TO_DEPTH

Indicates the operator described by the [DML_SPACE_TO_DEPTH_OPERATOR_DESC](#) structure.

DML_OPERATOR_DEPTH_TO_SPACE

Indicates the operator described by the [DML_DEPTH_TO_SPACE_OPERATOR_DESC](#) structure.

DML_OPERATOR_TILE

Indicates the operator described by the [DML_TILE_OPERATOR_DESC](#) structure.

DML_OPERATOR_TOP_K

Indicates the operator described by the [DML_TOP_K_OPERATOR_DESC](#) structure.

DML_OPERATOR_BATCH_NORMALIZATION

Indicates the operator described by the [DML_BATCH_NORMALIZATION_OPERATOR_DESC](#) structure.

DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION

Indicates the operator described by the
[DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC](#) structure.

DML_OPERATOR_LOCAL_RESPONSE_NORMALIZATION

Indicates the operator described by the
[DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC](#) structure.

DML_OPERATOR_LP_NORMALIZATION

Indicates the operator described by the [DML_LP_NORMALIZATION_OPERATOR_DESC](#) structure.

DML_OPERATOR_RNN

Indicates the operator described by the [DML_RNN_OPERATOR_DESC](#) structure.

DML_OPERATOR_LSTM

Indicates the operator described by the [DML_LSTM_OPERATOR_DESC](#) structure.

DML_OPERATOR_GRU

Indicates the operator described by the [DML_GRU_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_SIGN`

Indicates the operator described by the [DML_ELEMENT_WISE_SIGN_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_IS_NAN`

Indicates the operator described by the [DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_ERF`

Indicates the operator described by the [DML_ELEMENT_WISE_ERF_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_SINH`

Indicates the operator described by the [DML_ELEMENT_WISE_SINH_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_COSH`

Indicates the operator described by the [DML_ELEMENT_WISE_COSH_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_TANH`

Indicates the operator described by the [DML_ELEMENT_WISE_TANH_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_ASINH`

Indicates the operator described by the [DML_ELEMENT_WISE_ASINH_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_ACOSH`

Indicates the operator described by the [DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_ATANH`

Indicates the operator described by the [DML_ELEMENT_WISE_ATANH_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_IF`

Indicates the operator described by the [DML_ELEMENT_WISE_IF_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ELEMENT_WISE_ADD1`

Indicates the operator described by the [DML_ELEMENT_WISE_ADD1_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ACTIVATION_SHRINK`

Indicates the operator described by the [DML_ACTIVATION_SHRINK_OPERATOR_DESC](#) structure.

`DML_OPERATOR_MAX_POOLING1`

Indicates the operator described by the [DML_MAX_POOLING1_OPERATOR_DESC](#) structure.

`DML_OPERATOR_MAX_UNPOOLING`

Indicates the operator described by the [DML_MAX_UNPOOLING_OPERATOR_DESC](#) structure.

`DML_OPERATOR_DIAGONAL_MATRIX`

Indicates the operator described by the [DML_DIAGONAL_MATRIX_OPERATOR_DESC](#) structure.

`DML_OPERATOR_SCATTER`

Indicates the operator described by the [DML_SCATTER_OPERATOR_DESC](#) structure.

`DML_OPERATOR_ONE_HOT`

	Indicates the operator described by the DML_ONE_HOT_OPERATOR_DESC structure.
DML_OPERATOR_RESAMPLE	Indicates the operator described by the DML_RESAMPLE_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_LEFT	Indicates the operator described by the DML_ELEMENT_WISE_BIT_SHIFT_LEFT_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_BIT_SHIFT_RIGHT	Indicates the operator described by the DML_ELEMENT_WISE_BIT_SHIFT_RIGHT_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ROUND	Indicates the operator described by the DML_ELEMENT_WISE_ROUND_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_IS_INFINITY	Indicates the operator described by the DML_ELEMENT_WISE_IS_INFINITY_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_MODULUS_TRUNCATE	Indicates the operator described by the DML_ELEMENT_WISE_MODULUS_TRUNCATE_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_MODULUS_FLOOR	Indicates the operator described by the DML_ELEMENT_WISE_MODULUS_FLOOR_OPERATOR_DESC structure.
DML_OPERATOR_FILL_VALUE_CONSTANT	Indicates the operator described by the DML_FILL_VALUE_CONSTANT_OPERATOR_DESC structure.
DML_OPERATOR_FILL_VALUE_SEQUENCE	Indicates the operator described by the DML_FILL_VALUE_SEQUENCE_OPERATOR_DESC structure.
DML_OPERATOR_CUMULATIVE_SUMMATION	Indicates the operator described by the DML_CUMULATIVE_SUMMATION_OPERATOR_DESC structure.
DML_OPERATOR_REVERSE_SUBSEQUENCES	Indicates the operator described by the DML_REVERSE_SUBSEQUENCES_OPERATOR_DESC structure.
DML_OPERATOR_GATHER_ELEMENTS	Indicates the operator described by the DML_GATHER_ELEMENTS_OPERATOR_DESC structure.
DML_OPERATOR_GATHER_ND	Indicates the operator described by the DML_GATHER_ND_OPERATOR_DESC structure.
DML_OPERATOR_SCATTER_ND	

	Indicates the operator described by the DML_SCATTER_ND_OPERATOR_DESC structure.
DML_OPERATOR_MAX_POOLING2	Indicates the operator described by the DML_MAX_POOLING2_OPERATOR_DESC structure.
DML_OPERATOR_SLICE1	Indicates the operator described by the DML_SLICE1_OPERATOR_DESC structure.
DML_OPERATOR_TOP_K1	Indicates the operator described by the DML_TOP_K1_OPERATOR_DESC structure.
DML_OPERATOR_DEPTH_TO_SPACE1	Indicates the operator described by the DML_DEPTH_TO_SPACE1_OPERATOR_DESC structure.
DML_OPERATOR_SPACE_TO_DEPTH1	Indicates the operator described by the DML_SPACE_TO_DEPTH1_OPERATOR_DESC structure.
DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION1	Indicates the operator described by the DML_MEAN_VARIANCE_NORMALIZATION1_OPERATOR_DESC structure.
DML_OPERATOR_RESAMPLE1	Indicates the operator described by the DML_RESAMPLE1_OPERATOR_DESC structure.
DML_OPERATOR_MATRIX_MULTIPLY_INTEGER	Indicates the operator described by the DML_MATRIX_MULTIPLY_INTEGER_OPERATOR_DESC structure.
DML_OPERATOR_QUANTIZED_LINEAR_MATRIX_MULTIPLY	Indicates the operator described by the DML_QUANTIZED_LINEAR_MATRIX_MULTIPLY_OPERATOR_DESC structure.
DML_OPERATOR_CONVOLUTION_INTEGER	Indicates the operator described by the DML_CONVOLUTION_INTEGER_OPERATOR_DESC structure.
DML_OPERATOR_QUANTIZED_LINEAR_CONVOLUTION	Indicates the operator described by the DML_QUANTIZED_LINEAR_CONVOLUTION_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_BIT_AND	Indicates the operator described by the DML_ELEMENT_WISE_BIT_AND_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_BIT_OR	Indicates the operator described by the DML_ELEMENT_WISE_BIT_OR_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_BIT_XOR	Indicates the operator described by the DML_ELEMENT_WISE_BIT_XOR_OPERATOR_DESC

structure.

DML_OPERATOR_ELEMENT_WISE_BIT_NOT

Indicates the operator described by the [DML_ELEMENT_WISE_BIT_NOT_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_BIT_COUNT

Indicates the operator described by the [DML_ELEMENT_WISE_BIT_COUNT_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL

Indicates the operator described by the [DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OR_EQUAL_OPERATOR_DESC](#) structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL

Indicates the operator described by the [DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OR_EQUAL_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_CELU

Indicates the operator described by the [DML_ACTIVATION_CELU_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_RELU_GRAD

Indicates the operator described by the [DML_ACTIVATION_RELU_GRAD_OPERATOR_DESC](#) structure.

DML_OPERATOR_AVERAGE_POOLING_GRAD

Indicates the operator described by the [DML_AVERAGE_POOLING_GRAD_OPERATOR_DESC](#) structure.

DML_OPERATOR_MAX_POOLING_GRAD

Indicates the operator described by the [DML_MAX_POOLING_GRAD_OPERATOR_DESC](#) structure.

DML_OPERATOR_RANDOM_GENERATOR

Indicates the operator described by the [DML_RANDOM_GENERATOR_OPERATOR_DESC](#) structure.

DML_OPERATOR_NONZERO_COORDINATES

Indicates the operator described by the [DML_NONZERO_COORDINATES_OPERATOR_DESC](#) structure.

DML_OPERATOR_RESAMPLE_GRAD

Indicates the operator described by the [DML_RESAMPLE_GRAD_OPERATOR_DESC](#) structure.

DML_OPERATOR_SLICE_GRAD

Indicates the operator described by the [DML_SLICE_GRAD_OPERATOR_DESC](#) structure.

DML_OPERATOR_ADAM_OPTIMIZER

Indicates the operator described by the [DML_ADAM_OPTIMIZER_OPERATOR_DESC](#) structure.

DML_OPERATOR_ARGMIN

	Indicates the operator described by the DML_ARGMIN_OPERATOR_DESC structure.
<code>DML_OPERATOR_ARGMAX</code>	Indicates the operator described by the DML_ARGMAX_OPERATOR_DESC structure.
<code>DML_OPERATOR_ROI_ALIGN</code>	Indicates the operator described by the DML_ROI_ALIGN_OPERATOR_DESC structure.
<code>DML_OPERATOR_GATHER_ND1</code>	Indicates the operator described by the DML_GATHER_ND1_OPERATOR_DESC structure.
<code>DML_OPERATOR_ELEMENT_WISE_ATAN_YX</code>	Indicates the operator described by the DML_ELEMENT_WISE_ATAN_YX_OPERATOR_DESC structure.
<code>DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD</code>	Indicates the operator described by the DML_ELEMENT_WISE_CLIP_GRAD_OPERATOR_DESC structure.
<code>DML_OPERATOR_ELEMENT_WISE_DIFFERENCE_SQUARE</code>	Indicates the operator described by the DML_ELEMENT_WISE_DIFFERENCE_SQUARE_OPERATOR_DESC structure.
<code>DML_OPERATOR_LOCAL_RESPONSE_NORMALIZATION_GRAD</code>	Indicates the operator described by the DML_LOCAL_RESPONSE_NORMALIZATION_GRAD_OPERATOR_DESC structure.
<code>DML_OPERATOR_CUMULATIVE_PRODUCT</code>	Indicates the operator described by the DML_CUMULATIVE_PRODUCT_OPERATOR_DESC structure.
<code>DML_OPERATOR_BATCH_NORMALIZATION_GRAD</code>	Indicates the operator described by the DML_BATCH_NORMALIZATION_GRAD_OPERATOR_DESC structure.
<code>DML_OPERATOR_ELEMENT_WISE_QUANTIZED_LINEAR_ADD</code>	Indicates the operator described by the DML_ELEMENT_WISE_QUANTIZED_LINEAR_ADD_OPERATOR_DESC structure.
<code>DML_OPERATOR_DYNAMIC_QUANTIZE_LINEAR</code>	Indicates the operator described by the DML_DYNAMIC_QUANTIZE_LINEAR_OPERATOR_DESC structure.
<code>DML_OPERATOR_ROI_ALIGN1</code>	Indicates the operator described by the DML_ROI_ALIGN1_OPERATOR_DESC structure.
<code>DML_OPERATOR_ROI_ALIGN_GRAD</code>	Indicates the operator described by the DML_ROI_ALIGN_GRAD_OPERATOR_DESC structure.
<code>DML_OPERATOR_BATCH_NORMALIZATION_TRAINING</code>	Indicates the operator described by the

[DML_BATCH_NORMALIZATION_TRAINING_OPERATOR_DESC](#) structure.

[DML_OPERATOR_BATCH_NORMALIZATION_TRAINING_GRAD](#)

Indicates the operator described by the

[DML_BATCH_NORMALIZATION_TRAINING_GRAD_OPERATOR_DESC](#) structure.

[DML_OPERATOR_ELEMENT_WISE_CLIP1](#)

Indicates the operator described by the [DML_ELEMENT_WISE_CLIP1_OPERATOR_DESC](#) structure.

[DML_OPERATOR_ELEMENT_WISE_CLIP_GRAD1](#)

Indicates the operator described by the [DML_ELEMENT_WISE_CLIP_GRAD1_OPERATOR_DESC](#) structure.

[DML_OPERATOR_PADDING1](#)

Indicates the operator described by the [DML_PADDING1_OPERATOR_DESC](#) structure.

[DML_OPERATOR_ELEMENT_WISE_NEGATE](#)

Indicates the operator described by the [DML_ELEMENT_WISE_NEGATE_OPERATOR_DESC](#) structure.

Remarks

DML_FEATURE_LEVEL_6_4

DirectML [feature level 6_4](#) introduces the following operator types.

DML_OPERATOR_FOLD

Indicates the operator described by the [DML_FOLD_OPERATOR_DESC](#) structure.

DML_OPERATOR_RESAMPLE3

Indicates the operator described by the [DML_RESAMPLE3_OPERATOR_DESC](#) structure.

DML_OPERATOR_UNFOLD

Indicates the operator described by the [DML_UNFOLD_OPERATOR_DESC](#) structure.

DML_FEATURE_LEVEL_6_3

DirectML [feature level 6_3](#) introduces the following operator types.

TBD

DML_FEATURE_LEVEL_6_2

DirectML [feature level 6_2](#) introduces the following operator types.

DML_OPERATOR_ACTIVATION_HARD_SWISH

Indicates the operator described by the [DML_ACTIVATION_HARD_SWISH_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_SWISH

Indicates the operator described by the [DML_ACTIVATION_SWISH_OPERATOR_DESC](#) structure.

DML_OPERATOR_AVERAGE_POOLING1

Indicates the operator described by the [DML_AVERAGE_POOLING1_OPERATOR_DESC](#) structure.

DML_OPERATOR_LP_POOLING1

Indicates the operator described by the [DML_LP_POOLING1_OPERATOR_DESC](#) structure.

DML_OPERATOR_MATRIX_MULTIPLY_INTEGER_TO_FLOAT

Indicates the operator described by the [DML_MATRIX_MULTIPLY_INTEGER_TO_FLOAT_OPERATOR_DESC](#) structure.

DML_OPERATOR_MATRIX_MULTIPLY_INTEGER_TO_FLOAT

Indicates the operator described by the [DML_MATRIX_MULTIPLY_INTEGER_TO_FLOAT_OPERATOR_DESC](#) structure.

DML_FEATURE_LEVEL_6_1

DirectML [feature level 6_1](#) introduces the following operator type.

DML_OPERATOR_MULTIHEAD_ATTENTION

Indicates the operator described by the [DML_MULTIHEAD_ATTENTION_OPERATOR_DESC](#) structure.

DML_FEATURE_LEVEL_5_1

DirectML [feature level 5_1](#) introduces the following operator types.

DML_OPERATOR_ACTIVATION_GELU

Indicates the operator described by the [DML_ACTIVATION_GELU_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_HARDMAX1

Indicates the operator described by the [DML_ACTIVATION_HARDMAX1_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_LOG_SOFTMAX1

Indicates the operator described by the [DML_ACTIVATION_LOG_SOFTMAX1_OPERATOR_DESC](#) structure.

DML_OPERATOR_ACTIVATION_SOFTMAX1

Indicates the operator described by the [DML_ACTIVATION_SOFTMAX1_OPERATOR_DESC](#) structure.

DML_OPERATOR_DIAGONAL_MATRIX1

Indicates the operator described by the [DML_DIAGONAL_MATRIX1_OPERATOR_DESC](#) structure.

DML_OPERATOR_RESAMPLE_GRAD1

Indicates the operator described by the [DML_RESAMPLE_GRAD1_OPERATOR_DESC](#) structure.

DML_OPERATOR_RESAMPLE2

Indicates the operator described by the [DML_RESAMPLE2_OPERATOR_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_PADDING_MODE enumeration (directml.h)

Article02/22/2024

Defines constants that specify a mode for the DirectML pad operator (as described by the [DML_PADDING_OPERATOR_DESC](#) structure).

Syntax

C++

```
typedef enum DML_PADDING_MODE {
    DML_PADDING_MODE_CONSTANT,
    DML_PADDING_MODE_EDGE,
    DML_PADDING_MODE_REFLECTION,
    DML_PADDING_MODE_SYMMETRIC
};
```

Constants

 Expand table

DML_PADDING_MODE_CONSTANT Indicates padding with a constant.
DML_PADDING_MODE_EDGE Indicates edge mode for padding.
DML_PADDING_MODE_REFLECTION Indicates reflection mode for padding.

Requirements

 Expand table

Requirement	Value
Header	directml.h

See also

- [DML_PADDING_OPERATOR_DESC](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_RANDOM_GENERATOR_TYPE enumeration (directml.h)

Article02/22/2024

Defines constants that specify types of random random-number generator.

Syntax

C++

```
typedef enum DML_RANDOM_GENERATOR_TYPE {
    DML_RANDOM_GENERATOR_TYPE_PHILOX_4X32_10
} ;
```

Constants

[+] Expand table

DML_RANDOM_GENERATOR_TYPE_PHILOX_4X32_10

Specifies a generator for pseudo-random numbers according to the [Philox 4x32-10 algorithm](#).

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_RECURRENT_NETWORK_DIRECTION enumeration (directml.h)

Article02/22/2024

Defines constants that specify a direction for a recurrent DirectML operator.

Syntax

C++

```
typedef enum DML_RECURRENT_NETWORK_DIRECTION {
    DML_RECURRENT_NETWORK_DIRECTION_FORWARD,
    DML_RECURRENT_NETWORK_DIRECTION_BACKWARD,
    DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL
} ;
```

Constants

[Expand table](#)

`DML_RECURRENT_NETWORK_DIRECTION_FORWARD`

Indicates the forward pass.

`DML_RECURRENT_NETWORK_DIRECTION_BACKWARD`

Indicates the backward pass.

`DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL`

Indicates both passes.

Requirements

[Expand table](#)

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_REDUCE_FUNCTION enumeration (directml.h)

Article 10/05/2021

Defines constants that specify the specific reduction algorithm to use for the DirectML reduce operator (as described by the [DML_REDUCE_OPERATOR_DESC](#) structure).

Syntax

C++

```
typedef enum DML_REDUCE_FUNCTION {
    DML_REDUCE_FUNCTION_ARGMAX,
    DML_REDUCE_FUNCTION_ARGMIN,
    DML_REDUCE_FUNCTION_AVERAGE,
    DML_REDUCE_FUNCTION_L1,
    DML_REDUCE_FUNCTION_L2,
    DML_REDUCE_FUNCTION_LOG_SUM,
    DML_REDUCE_FUNCTION_LOG_SUM_EXP,
    DML_REDUCE_FUNCTION_MAX,
    DML_REDUCE_FUNCTION_MIN,
    DML_REDUCE_FUNCTION_MULTIPLY,
    DML_REDUCE_FUNCTION_SUM,
    DML_REDUCE_FUNCTION_SUM_SQUARE
} ;
```

Constants

[+] Expand table

DML_REDUCE_FUNCTION_ARGMAX

Indicates a reduction function that computes the indices of the max elements of the input tensor's elements along the specified axis, $\text{int32 } \{i j k ..\} = \text{maxindex}(X Y Z ...)$.

DML_REDUCE_FUNCTION_ARGMIN

Indicates a reduction function that computes the indices of the min elements of the input tensor's elements along the specified axis, $\text{int32 } \{i j k ..\} = \text{minindex}(X Y Z ...)$.

DML_REDUCE_FUNCTION_AVERAGE

Indicates a reduction function that computes the mean of the input tensor's elements along the specified axes, $x = (x_1 + x_2 + \dots + x_n) / n$.

DML_REDUCE_FUNCTION_L1

Indicates a reduction function that computes the L1 norm of the input tensor's elements along the specified axes, $x = |x_1| + |x_2| + \dots + |x_n|$.

DML_REDUCE_FUNCTION_L2

Indicates a reduction function that computes the L2 norm of the input tensor's elements along the specified axes, $x = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$.

DML_REDUCE_FUNCTION_LOG_SUM

Indicates a reduction function that computes the log sum of the input tensor's elements along the specified axes, $x = \log(x_1 + x_2 + \dots + x_n)$.

DML_REDUCE_FUNCTION_LOG_SUM_EXP

Indicates a reduction function that computes the log sum exponent of the input tensor's elements along the specified axes, $x = \log(\exp(x_1) + \exp(x_2) + \dots + \exp(x_n))$.

DML_REDUCE_FUNCTION_MAX

Indicates a reduction function that computes the max of the input tensor's elements along the specified axes, $x = \max(\max(\max(x_1, x_2), x_3), \dots, x_n)$.

DML_REDUCE_FUNCTION_MIN

Indicates a reduction function that computes the min of the input tensor's elements along the specified axes, $x = \min(\min(\min(x_1, x_2), x_3), \dots, x_n)$.

DML_REDUCE_FUNCTION_MULTIPLY

Indicates a reduction function that computes the product of the input tensor's elements along the specified axes, $x = (x_1 * x_2 * \dots * x_n)$.

DML_REDUCE_FUNCTION_SUM

Indicates a reduction function that computes the sum of the input tensor's elements along the specified axes, $x = (x_1 + x_2 + \dots + x_n)$.

DML_REDUCE_FUNCTION_SUM_SQUARE

Indicates a reduction function that computes the sum square of the input tensor's elements along the specified axes, $x = x_1^2 + x_2^2 + \dots + x_n^2$.

Requirements

 Expand table

Requirement	Value
Header	directml.h

See also

- DML_REDUCE_OPERATOR_DESC
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DML_TENSOR_DATA_TYPE enumeration (directml.h)

Article02/19/2025

Specifies the data type of the values in a tensor. DirectML operators may not support all data types; see the documentation for each specific operator to find which data types it supports.

Syntax

C++

```
typedef enum DML_TENSOR_DATA_TYPE {
    DML_TENSOR_DATA_TYPE_UNKNOWN,
    DML_TENSOR_DATA_TYPE_FLOAT32,
    DML_TENSOR_DATA_TYPE_FLOAT16,
    DML_TENSOR_DATA_TYPE_UINT32,
    DML_TENSOR_DATA_TYPE_UINT16,
    DML_TENSOR_DATA_TYPE_UINT8,
    DML_TENSOR_DATA_TYPE_INT32,
    DML_TENSOR_DATA_TYPE_INT16,
    DML_TENSOR_DATA_TYPE_INT8,
    DML_TENSOR_DATA_TYPE_FLOAT64,
    DML_TENSOR_DATA_TYPE_UINT64,
    DML_TENSOR_DATA_TYPE_INT64
} ;
```

Constants

 Expand table

DML_TENSOR_DATA_TYPE_UNKNOWN Indicates an unknown data type. This value is never valid.
DML_TENSOR_DATA_TYPE_FLOAT32 Indicates a 32-bit floating-point data type.
DML_TENSOR_DATA_TYPE_FLOAT16 Indicates a 16-bit floating-point data type.
DML_TENSOR_DATA_TYPE_UINT32

Indicates a 32-bit unsigned integer data type.

`DML_TENSOR_DATA_TYPE_UINT16`

Indicates a 16-bit unsigned integer data type.

`DML_TENSOR_DATA_TYPE_UINT8`

Indicates an 8-bit unsigned integer data type.

`DML_TENSOR_DATA_TYPE_INT32`

Indicates a 32-bit signed integer data type.

`DML_TENSOR_DATA_TYPE_INT16`

Indicates a 16-bit signed integer data type.

`DML_TENSOR_DATA_TYPE_INT8`

Indicates an 8-bit signed integer data type.

Remarks

DML_FEATURE_LEVEL_6_3

DirectML [feature level 6_3](#) introduces the following data types.

DML_TENSOR_DATA_TYPE_UINT4

Indicates a 4-bit unsigned integer data type. The **UINT4** data type puts the first element in the low nibble of the byte; and the second element in the high nibble (standard little-endian order). Dimensions don't need to be byte-aligned (odd sizes are fine). For example, if you have the **UINT4** element data [1,2,3,4,5], then that yields 3 bytes [0x21, 0x43, 0x05] with an ignorable tail nibble.

DML_TENSOR_DATA_TYPE_INT4

Indicates a 4-bit signed integer data type.

Requirements

[] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_TENSOR_FLAGS enumeration (directml.h)

Article 02/22/2024

Specifies additional options in a tensor description. Values can be bitwise OR'd together.

Syntax

C++

```
typedef enum DML_TENSOR_FLAGS {
    DML_TENSOR_FLAG_NONE = 0x0,
    DML_TENSOR_FLAG_OWNED_BY_DML = 0x1
};
```

Constants

[] Expand table

DML_TENSOR_FLAG_NONE

Value: *0x0*

No options are specified.

DML_TENSOR_FLAG_OWNED_BY_DML

Value: *0x1*

Indicates that the tensor data should be owned and managed by DirectML. The effect of this flag is that DirectML makes a copy of the tensor data during initialization of an operator, storing it in the persistent resource. This allows DirectML to perform reformatting of the tensor data into other, more efficient forms. Setting this flag may increase performance, but is typically only useful for tensors whose data doesn't change for the lifetime of the operator (for example, weight tensors).

This flag can only be used on input tensors.

When this flag is set on a particular tensor description, the corresponding tensor must be bound to the binding table during operator initialization, and not during execution. Attempting to bind the tensor during execution while this flag is set results in an error. This is the opposite of the default behavior (the behavior without the **DML_TENSOR_FLAG_OWNED_BY_DML** flag), where the tensor is expected to be bound during execution, and not during initialization.

Requirements

 Expand table

Requirement	Value
Header	directml.h

See also

[Binding in DirectML](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DML_TENSOR_TYPE enumeration (directml.h)

Article 02/22/2024

Identifies a type of tensor description.

Syntax

C++

```
typedef enum DML_TENSOR_TYPE {
    DML_TENSOR_TYPE_INVALID,
    DML_TENSOR_TYPE_BUFFER
} ;
```

Constants

[+] Expand table

`DML_TENSOR_TYPE_INVALID`

Indicates an unknown tensor description type. This value is never valid.

`DML_TENSOR_TYPE_BUFFER`

Indicates a tensor description that is represented by a Direct3D 12 buffer. The corresponding struct type is [DML_BUFFER_TENSOR_DESC](#).

Requirements

[+] Expand table

Requirement	Value
Header	directml.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

DirectML constants

Article • 02/10/2025

The following constants are declared in `DirectML.h`.

[+] Expand table

Constant	Value	Description
DML_TENSOR_DIMENSION_COUNT_MAX	5	DirectML tensors support a maximum of 5 dimensions for <code>DML_TARGET_VERSION < DML_FEATURE_LEVEL_3_0</code> .
DML_TENSOR_DIMENSION_COUNT_MAX1	8	DirectML tensors support a maximum of 8 dimensions for <code>DML_TARGET_VERSION >= DML_FEATURE_LEVEL_3_0</code> .
DML_TEMPORARY_BUFFER_ALIGNMENT	256	Temporary and persistent buffers must have a base address that is aligned to 256 bytes.
DML_PERSISTENT_BUFFER_ALIGNMENT	256	Temporary and persistent buffers must have a base address that is aligned to 256 bytes.
DML_MINIMUM_BUFFER_TENSOR_ALIGNMENT	16	Buffer tensors have a minimum base address alignment requirement of 16 bytes.

Requirements

[+] Expand table

Requirement	Value
Header	<code>DirectML.h</code>

See also

- [DirectML reference](#)
- [Windows AI](#)
- [Core reference](#)

- Direct3D 12 Reference
-

Feedback

Was this page helpful?

 Yes

 No

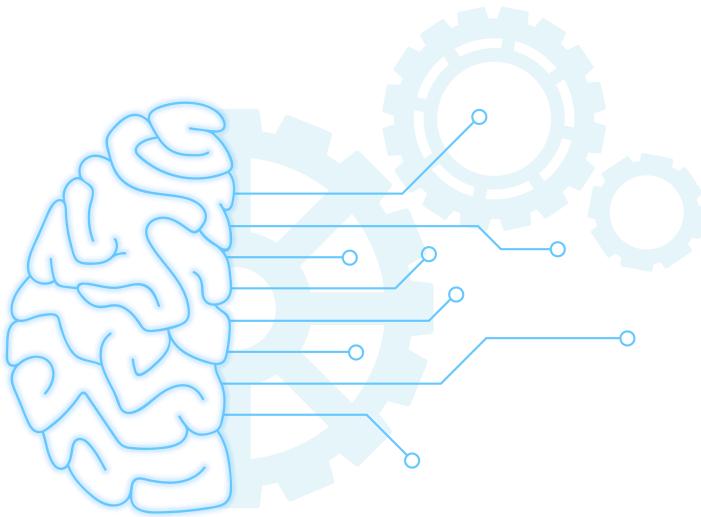
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Windows Machine Learning

ⓘ Important

For the latest documentation about Windows Machine Learning, see [What is Windows ML](#). That documentation describes APIs that are in the `Microsoft.Windows.AI.MachineLearning` namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the `Windows.AI.MachineLearning` namespace, and were shipped in 2018.

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 11, Windows 10, Windows Server 2022, 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 11 and Windows Server 2022, 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.

Machine Learning models

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, or check out [What is a machine learning model](#) to learn more.

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.

Get Started

To learn a bit more about the different ways to incorporate Windows Machine Learning into your app, check out our [get started page](#).

Looking to create your first app using Windows Machine Learning? Check out the [WinML tutorials](#) for an overview of the different ways to train a model, and incorporate it into your WinML application.

FAQ

Interested in learning more about Machine Learning solutions and your options? For a full overview of the choices available, see [Compare AI solutions](#), or learn more with the [WinML FAQ](#).

!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](#) ↗.

Last updated on 07/31/2025

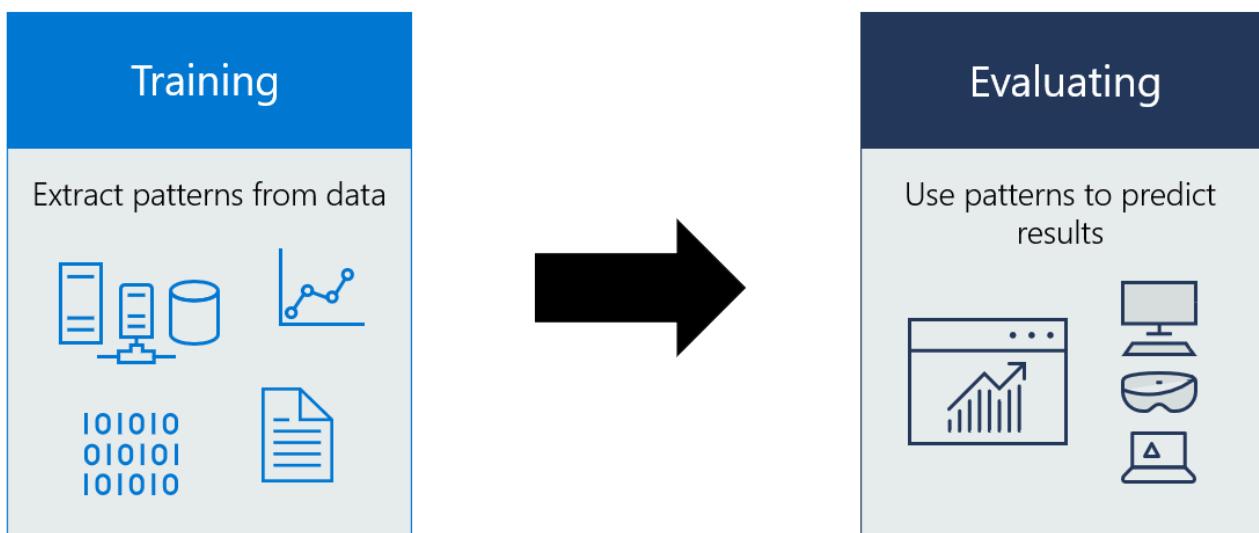
What is a machine learning model?

ⓘ Important

For the latest documentation about Windows Machine Learning, see [What is Windows ML](#). That documentation describes APIs that are in the `Microsoft.Windows.AI.MachineLearning` namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the `Windows.AI.MachineLearning` namespace, and were shipped in 2018.

A machine learning model is an object (stored locally in 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.



When to use Machine Learning

Good machine learning scenarios often have the following common properties:

1. They involve a repeated decision or evaluation which you want to automate and need consistent results.
2. It is difficult or impossible to explicitly describe the solution or criteria behind a decision.

3. You have labeled data, or existing examples where you can describe the situation and map it to the correct result.

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.

Get started

You can get started with Windows Machine Learning by following [one of our full-app tutorials](#) or jumping straight to the [Windows Machine Learning samples](#).

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](#) .

Last updated on 07/31/2025

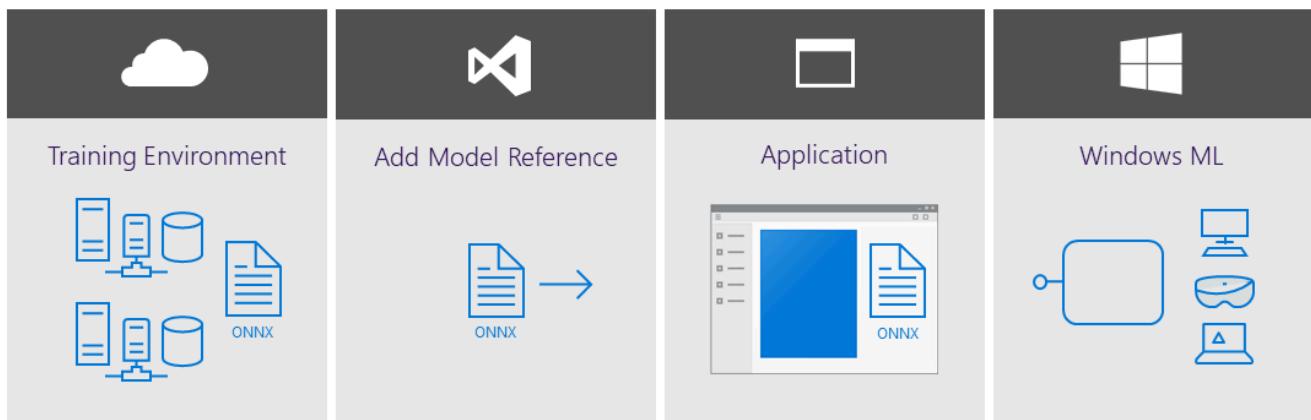
Get Started with Windows Machine Learning

ⓘ Important

For the latest documentation about Windows Machine Learning, see [What is Windows ML](#). That documentation describes APIs that are in the `Microsoft.Windows.AI.MachineLearning` namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the `Windows.AI.MachineLearning` namespace, and were shipped in 2018.

There are several ways to use Windows Machine Learning in your app. At the core, you just need a couple of straightforward steps.

1. Get a trained Open Neural Network Exchange (ONNX) model, or convert models trained in other ML frameworks into ONNX with [ONNXMLTools](#).
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.



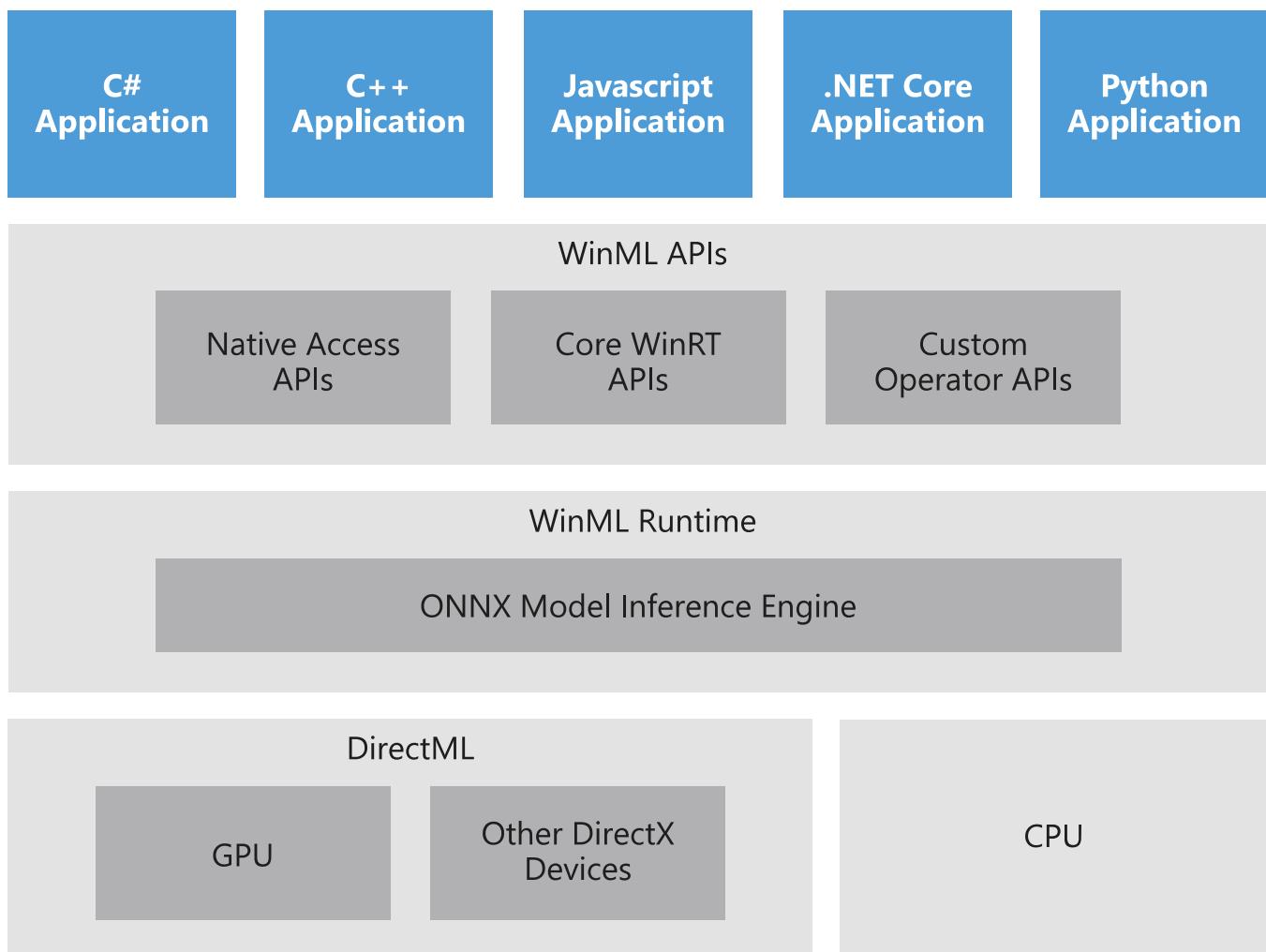
In-box vs NuGet WinML solutions

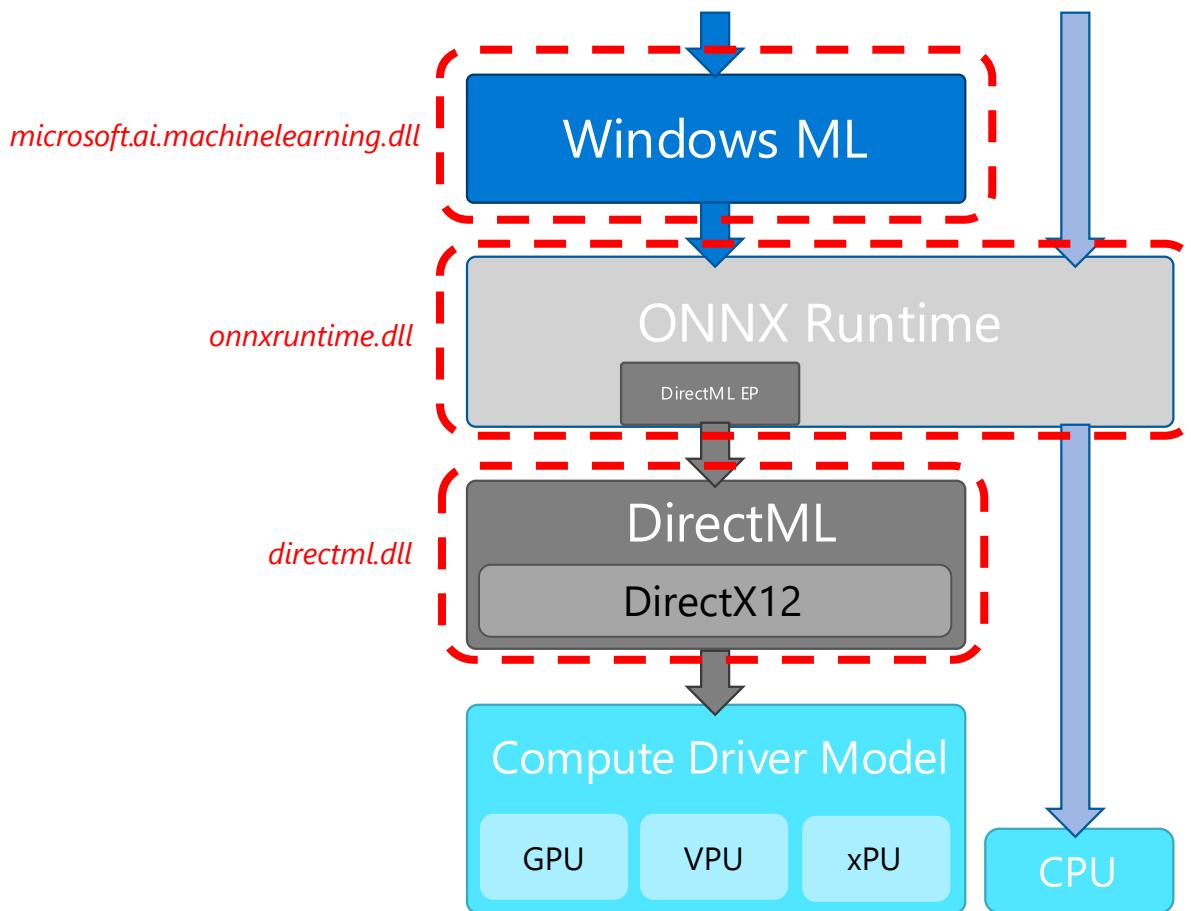
The table below highlights the availability, distribution, language support, servicing, and forward compatibility aspects of the In-Box and NuGet package for Windows ML.

[] Expand table

Properties	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.

Learn more

If you want to use the Windows ML 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](#).

 **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](#).

Last updated on 07/30/2025

Windows ML APIs in Windows.AI.MachineLearning

ⓘ Important

For the latest documentation about Windows Machine Learning, see [What is Windows ML](#). That documentation describes APIs that are in the `Microsoft.Windows.AI.MachineLearning` namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the `Windows.AI.MachineLearning` namespace, and were shipped in 2018.

The `Windows.AI.MachineLearning` Windows ML (WinML) APIs are divided into core APIs, custom operators, and native APIs.

[+] Expand table

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

ⓘ 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](#).

Windows.AI.MachineLearning Namespace

Important

For the latest documentation about Windows Machine Learning, see [What is Windows ML](#). That documentation describes APIs that are in the **Microsoft.Windows.AI.MachineLearning** namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the **Windows.AI.MachineLearning** namespace, and were shipped in 2018.

Enables apps to load machine learning models, bind features, and evaluate the results.

Classes

 Expand table

Name	Description
ImageFeatureDescriptor	<p> Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>Describes the properties of the image the model is expecting.</p>
ImageFeatureValue	<p> Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>Describes the properties of the image used to pass into a model.</p>

Name	Description
Learning Model	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>Represents a trained machine learning model.</p>
Learning ModelBinding	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>Used to bind values to named input and output features.</p>
Learning ModelDevice	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>The device used to evaluate the machine learning model.</p>
Learning Model Evaluation Result	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the</p>

Name	Description
	<p>Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>Get the results of the evaluation.</p>
Learning ModelSession	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>Used to evaluate machine learning models.</p>
Learning ModelSession Options	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>Describes inference options that are used during the creation of LearningModelSession objects.</p>
MapFeature Descriptor	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>

Name	Description
	A map is a collection of (key, value) pairs.
Sequence Feature Descriptor	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	A sequence is an array of elements.
Tensor Boolean	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	A boolean tensor object.
TensorDouble	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	A 64-bit float tensor object.
TensorFeature Descriptor	<p>Important</p>

Name	Description
	<p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>Tensors are multi-dimensional arrays of values.</p>
TensorFloat	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>A 32-bit float tensor object.</p>
Tensor Float16Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>A 16-bit float tensor object.</p>
Tensor Int16Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here,</p>

Name	Description
	which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.
	A 16-bit signed integer tensor object.
Tensor Int32Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	A 32-bit signed integer tensor object.
Tensor Int64Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	A 64-bit signed integer tensor object.
TensorInt8Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	An 8-bit signed integer tensor object.

Name	Description
TensorString	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>A string tensor object.</p>
Tensor UInt16Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>A 16-bit unsigned integer tensor object.</p>
Tensor UInt32Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>A 32-bit unsigned integer tensor object.</p>
Tensor UInt64Bit	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the</p>

Name	Description
	<p>Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
Tensor UInt8Bit	<p>A 64-bit unsigned integer tensor object.</p> <p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>A 8-bit unsigned integer tensor object.</p>

Interfaces

[] [Expand table](#)

Name	Description
ILearning ModelFeature Descriptor	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>Describes the common properties that all features have.</p>
ILearning ModelFeature Value	<p>Important</p>

Name	Description
	<p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>The instantiated value for a feature.</p>
ILearningModelOperatorProvider	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
ITensor	<p>Describes the operators for a learning model.</p> <p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	<p>Tensors are multi-dimensional values.</p>

Enums

 Expand table

Name	Description
LearningModelDeviceKind	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	Defines the list of device kinds that can evaluate a machine learning model.
LearningModelFeatureKind	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	Input and output feature kinds for a machine learning model.
LearningModelPixelRange	<p>Important</p> <p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p>
	Defines the list of image nominal pixel range supported by Windows ML. The proper value is specified in a machine learning model's metadata.
TensorKind	<p>Important</p>

Name	Description
	<p>For the latest documentation about Windows Machine Learning, see What is Windows ML. That documentation describes APIs that are in the Microsoft.Windows.AI.MachineLearning namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the Windows.AI.MachineLearning namespace, and were shipped in 2018.</p> <p>Defines the list of supported tensor data types.</p>

Examples

The following example loads a model, creates an evaluation session, gets the input and output features of the model, binds those features, and evaluates.

C#

```
private async Task LoadAndEvaluateModelAsync(VideoFrame _inputFrame, string _modelName)
{
    LearningModel _model;
    ImageFeatureDescriptor _inputImageDescription;
    TensorFeatureDescriptor _outputImageDescription;
    LearningModelBinding _binding = null;
    VideoFrame _outputFrame = null;
    LearningModelSession _session;

    try
    {
        // Load and create the model
        var modelFile =
            await StorageFile.GetFileFromApplicationUriAsync(new Uri($"ms-
appx:///Assets/{_modelName}"));
        _model = await LearningModel.LoadFromStorageFileAsync(modelFile);

        // Create the evaluation session with the model
        _session = new LearningModelSession(_model);

        //Get input and output features of the model
        List<ILearningModelFeatureDescriptor> inputFeatures =
        _model.InputFeatures.ToList();
        List<ILearningModelFeatureDescriptor> outputFeatures =
        _model.OutputFeatures.ToList();

        // Retrieve the first input feature which is an image
        _inputImageDescription =
            inputFeatures.FirstOrDefault(feature => feature.Kind ==
LearningModelFeatureKind.Image)
```

```

    as ImageFeatureDescriptor;

    // Retrieve the first output feature which is a tensor
    _outputImageDescription =
        outputFeatures.FirstOrDefault(feature => feature.Kind ==
LearningModelFeatureKind.Tensor)
            as TensorFeatureDescriptor;

    //Create output frame based on expected image width and height
    _outputFrame = new VideoFrame(
        BitmapPixelFormat.Bgra8,
        (int)_inputImageDescription.Width,
        (int)_inputImageDescription.Height);

    //Create binding and then bind input/output features
    _binding = new LearningModelBinding(_session);

    _binding.Bind(_inputImageDescription.Name, _inputFrame);
    _binding.Bind(_outputImageDescription.Name, _outputFrame);

    //Evaluate and get the results
    var results = await _session.EvaluateAsync(_binding, "test");
}

catch (Exception ex)
{
    StatusBlock.Text = $"error: {ex.Message}";
    _model = null;
}

}

```

Remarks

Windows Server

To use this API on Windows Server, you must use Windows Server 2019 with Desktop Experience.

Thread safety

This API is thread-safe.

See also

- [Windows ML](#)
- [Windows ML samples \(GitHub\)](#) ↗

Custom operators

Article • 12/30/2021

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.
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.

Name	Description
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](#).

MLOperatorAttributeType enum

Article • 12/30/2021

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

Requirement	
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](#).

MLOperatorEdgeType enum

Article • 12/30/2021

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

Fields

Name	Value	Description
Undefined	0	
Tensor	1	

Requirements

Requirement	
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](#).

MLOperatorExecutionType enum

Article • 12/30/2021

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

Fields

Name	Value	Description
Undefined	0	
Cpu	1	
D3D12	2	

Requirements

Requirement	
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](#).

MLOperatorKernelOptions enum

Article • 12/30/2021

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

Requirement	
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](#).

MLOperatorParameterOptions enum

Article • 12/30/2021

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

Requirement	
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](#).

MLOperatorSchemaEdgeTypeFormat enum

Article • 12/30/2021

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

Requirement	
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](#) ↗.

MLOperatorTensorDataType enum

Article • 12/30/2021

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

Requirement

Requirement	
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](#).

MLCreateOperatorRegistry function

Article • 12/30/2021

Creates an instance of [IMLOperatorRegistry](#) which may be used to register a custom operator kernel and custom operator schema.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorAttributes interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorAttributes.GetAttribute method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorAttributes.GetAttributeElementCount method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorAttributes.GetStringAttributeElement method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorAttributes.GetStringAttributeElementLength method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernel interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorKernel.Compute method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelContext interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorKernelContext::AllocateTemporaryData method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelContext::GetExecutionInterface method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelContext.GetInputTensor method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelContext.GetOutputTensor method

Article • 12/30/2021

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.

C++

```
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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelContext.GetOutputTensor method

Article • 12/30/2021

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.

C++

```
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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelCreationContext interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorKernelCreationContext::GetExecutionInterface method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelCreationContext.GetInputCount method

Article • 12/30/2021

Gets the number of inputs to the operator.

C++

```
uint32_t GetInputCount()
```

Requirements

Requirement	
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](#).

IMLOperatorKernelCreationContext.GetInputEdgeDescription method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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 ↗](#).

IMLOperatorKernelCreationContext.GetOutputCount method

Article • 12/30/2021

Gets the number of outputs to the operator.

C++

```
uint32_t GetOutputCount()
```

Requirements

Requirement	
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](#).

IMLOperatorKernelCreationContext.GetOutputEdgeDescription method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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 ↗](#).

IMLOperatorKernelCreationContext.GetTensorShapeDescription method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelCreationContext::HasTensorShapeDescription method

Article • 12/30/2021

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.

C++

```
bool HasTensorShapeDescription()
```

Requirements

Requirement	
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](#).

IMLOperatorKernelCreationContext::IsInputValid method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelCreationContext::IsOutputValid method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorKernelFactory interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorKernelFactory.CreateKernel method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorRegistry interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorRegistry.RegisterOperatorKernel method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorRegistry.RegisterOperatorSetSchema method

Article • 12/30/2021

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.

C++

```
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

Requirement	
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](#) ↗.

IMLOperatorShapeInferenceContext interface

Article • 12/30/2021

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

Requirement	
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](#) ↗.

IMLOperatorShapeInferenceContext.GetInputCount method

Article • 12/30/2021

Gets the number of inputs to the operator.

C++

```
uint32_t GetInputCount()
```

Requirements

Requirement	
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](#).

IMLOperatorShapeInferenceContext.GetInputEdgeDescription method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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 ↗](#).

IMLOperatorShapeInferenceContext.GetInputTensorDimensionCount method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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 ↗](#).

IMLOperatorShapeInferenceContext.GetInputTensorShape method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorShapeInferenceContext.GetOutputCount method

Article • 12/30/2021

Gets the number of outputs to the operator.

C++

```
uint32_t GetOutputCount()
```

Requirements

Requirement	
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](#).

IMLOperatorShapeInferenceContext::IsInputValid method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorShapeInferenceContext.IsOutputValid method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorShapeInferenceContext.SetOutputTensorShape method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorShapeInerrer interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorShapeInferrer.InferOutputShapes method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTensor interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorTensor.GetData method

Article • 12/30/2021

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.

C++

```
void* GetData()
```

Requirements

Requirement	
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](#).

IMLOperatorTensor.GetDatInterface method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTensor.GetDimensionCount method

Article • 12/30/2021

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

C++

```
uint32_t GetDimensionCount()
```

Requirements

Requirement	
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](#).

IMLOperatorTensor.GetShape method

Article • 12/30/2021

Gets the size of dimensions in the tensor.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTensor.GetTensorDataType method

Article • 12/30/2021

Gets the data type of the tensor.

C++

```
MLOperatorTensorDataType GetTensorDataType()
```

Requirements

Requirement	
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](#).

IMLOperatorTensor::IsCpuData method

Article • 12/30/2021

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

C++

```
bool IsCpuData()
```

Requirements

Requirement	
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](#) ↗.

IMLOperatorTensor::IsDataInterface method

Article • 12/30/2021

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](#).

C++

```
bool IsDataInterface()
```

Requirements

Requirement	
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](#) ↗.

IMLOperatorTensorShapeDescription interface

Article • 12/30/2021

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

Requirement	
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](#) ↗.

IMLOperatorTensorShapeDescription.GetInputTensorDimensionCount method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTensorShapeDescription.GetInputTensorShape method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTensorShapeDescription.GetOutputTensorDimensionCount method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTensorShapeDescription.GetOutputTensorShape method

Article • 12/30/2021

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.

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTensorShapeDescription::HasOutputShapeDescription method

Article • 12/30/2021

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.

C++

```
bool HasOutputShapeDescription()
```

Requirements

Requirement	
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](#).

IMLOperatorTypeInferenceContext interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorTypeInferenceContext.GetInputCount method

Article • 12/30/2021

Gets the number of inputs to the operator.

C++

```
uint32_t GetInputCount()
```

Requirements

Requirement	
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](#).

IMLOperatorTypeInferenceContext.GetInputEdgeDescription method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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 ↗](#).

IMLOperatorTypeInferenceContext.GetOutputCount method

Article • 12/30/2021

Gets the number of outputs to the operator.

C++

```
uint32_t GetOutputCount()
```

Requirements

Requirement	
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](#).

IMLOperatorTypeInferenceContext::IsInputValid method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTypeInferenceContext::IsOutputValid method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

IMLOperatorTypeInferenceContext.SetOutputEdgeDescription method

Article • 12/30/2021

Sets the inferred type of an output edge.

C++

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

Requirements

Requirement	
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 ↗](#).

IMLOperatorTypeInferer interface

Article • 12/30/2021

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

Requirement	
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](#).

IMLOperatorTypeInferrer.InferOutputTypes method

Article • 12/30/2021

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

C++

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

Requirements

Requirement	
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](#).

MLOperatorAttribute struct

Article • 12/30/2021

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

Requirement	
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](#).

MLOperatorAttributeNameValue struct

Article • 12/30/2021

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	<code>const float*</code>	32-bit floating point value(s). Used when the type field is <code>MLOperatorAttributeType::Float</code> or <code>MLOperatorAttributeType::FloatArray</code> .
ints	<code>const int64_t*</code>	64-bit integer value(s). Used when the type field is <code>MLOperatorAttributeType::Int</code> or <code>MLOperatorAttributeType::IntArray</code> .
name	<code>const char*</code>	NULL-terminated UTF-8 string representing the name of the attribute in the associated operator type.
reserved	<code>const void*</code>	
strings	<code>const char* const*</code>	NULL-terminated UTF-8 string value(s). Used when the type field is <code>MLOperatorAttributeType::String</code> or <code>MLOperatorAttributeType::StringArray</code> .
type	<code>MLOperatorAttributeType</code>	The type of the attribute in the associated operator type.
valueCount	<code>uint32_t</code>	The number of elements in the attribute value. This must be 1, except for attributes which are of array types.

Requirements

Requirement	
Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	<code>MLOperatorAuthor.h</code>

 **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](#).

MLOperatorEdgeDescription struct

Article • 12/30/2021

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

Fields

Name	Type	Description
edgeType	MLOperatorEdgeType	The type of the edge.
reserved	<code>uint64_t</code>	
tensorDataType	MLOperatorTensorDataType	The data type of a tensor. Used when <code>edgeType</code> is set to <code>Tensor</code> .

Requirements

Requirement	
Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	<code>MLOperatorAuthor.h</code>

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](#).

MLOperatorEdgeTypeConstraint struct

Article • 12/30/2021

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

Requirement	
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](#).

MLOperatorKernelDescription struct

Article • 12/30/2021

Description of a custom operator kernel used to register that schema.

Fields

Name	Type	Description
defaultAttributeCount	<code>uint32_t</code>	The number of provided default attribute values.
defaultAttributes	<code>const MLOperatorAttributeNameValue*</code>	The default values of attributes. These will be applied when the attributes are missing in a model containing the operator type.
domain	<code>const char*</code>	NULL-terminated UTF-8 string representing the name of the operator's domain.
executionOptions	<code>uint32_t</code>	Reserved for additional options. Must be 0.
executionType	<code>MLOperatorExecutionType</code>	Specifies whether a kernel uses the CPU or GPU for computation.
minimumOperatorSetVersion	<code>int32_t</code>	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	<code>const char*</code>	NULL-terminated UTF-8 string representing the name of the operator.
options	<code>MLOperatorKernelOptions</code>	Options for the kernel which apply to all execution provider types.
typeConstraintCount	<code>uint32_t</code>	The number of type constraints provided.

Name	Type	Description
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

Requirement	
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](#).

MLOperatorSchemaDescription struct

Article • 12/30/2021

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.

Name	Type	Description
outputCount	<code>uint32_t</code>	The number of outputs of the operator.
outputs	<code>const MLOperatorSchemaEdgeDescription*</code>	An array containing the descriptions of the operator's output edges.
typeConstraintCount	<code>uint32_t</code>	The number of type constraints provided.
typeConstraints	<code>const MLOperatorEdgeTypeConstraint*</code>	An array of type constraints. Each constraint restricts input and outputs associated with a type label string to one or more edge types.

Requirements

Requirement	
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](#).

MLOperatorSchemaEdgeDescription struct

Article • 12/30/2021

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 <code>typeFormat</code> is MLOperatorSchemaEdgeTypeFormat::EdgeDescription .
options	MLOperatorParameterOptions	Options of the parameter, including whether it is optional or variadic.
reserved	<code>void*</code>	
typeFormat	MLOperatorSchemaEdgeTypeFormat	The manner in which the type constraints and type mapping are defined.
typeLabel	<code>char*</code>	A type label string constructed as in ONNX operator schema. For example, "T". This is used when <code>typeFormat</code> is MLOperatorSchemaEdgeTypeFormat::Label .

Requirements

Requirement	
Minimum supported client	Windows 10, build 17763
Minimum supported server	Windows Server 2019 with Desktop Experience
Header	<code>MLOperatorAuthor.h</code>

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](#).

MLOperatorSetId struct

Article • 12/30/2021

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

Requirement	
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](#).

WinML native APIs

Article • 12/30/2021

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](#) ↗.

ILearningModelDeviceFactoryNative interface

Article • 12/30/2021

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

	Requirement
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](#) ↗.

ILearningModelDeviceFactoryNative.CreateFromD3D12CommandQueue method

Article • 12/30/2021

Creates a [LearningModelDevice](#) that will run inference on the user-specified [ID3D12CommandQueue](#).

C++

```
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

C++

```
// 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

Requirement	
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 ↗](#).

ITensorNative interface

Article • 12/30/2021

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

Requirement	
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](#).

ITensorNative.GetBuffer method

Article • 12/30/2021

Gets the tensor's buffer as an array of bytes.

C++

```
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

C++

```
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

Requirement	
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 ↗](#).

ITensorNative.GetD3D12Resource method

Article • 12/30/2021

Gets the tensor buffer as an [ID3D12Resource](#).

C++

```
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

Requirement	
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](#).

ITensorStaticsNative interface

Article • 12/30/2021

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

Requirement	
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](#) ↗.

ITensorStaticsNative.CreateFromD3D12Resource method

Article • 12/30/2021

Creates a tensor object ([TensorFloat](#), [TensorInt32Bit](#)) from a user-specified [ID3D12Resource](#).

C++

```
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	<code>__int64*</code>	The shape of the tensor.
shapeCount	<code>int</code>	The number of dimensions of the tensor.
result	IUnknown **	The resulting tensor.

Returns

`HRESULT` The result of the operation.

Examples

C++

```
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

Requirement	
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 ↗](#).

IntraopNumThreads interface

Article • 12/30/2021

Provides an ability to change the number of threads used in the threadpool for Intra Operator Execution for CPU operators through [LearningModelSessionOptions](#). By default, WinML sets the value as the maximum number of threads, which is the same number of logical cores on the user's CPU. Setting this value higher than the number of logical cores on the CPU may result in an inefficient threadpool and a slower evaluation.

Sample code

C++

```
void SetIntraOpNumThreads(LearningModel model) {
    // Create LearningModelSessionOptions
    auto options = LearningModelSessionOptions();
    auto nativeOptions = options.as<ILearningModelSessionOptionsNative>();

    // Set the number of intra op threads to half of logical cores.
    uint32_t desiredThreads = std::thread::hardware_concurrency() / 2;
    nativeOptions->SetIntraOpNumThreadsOverride(desiredThreads);

    // Create session
    LearningModelSession session = nullptr;
    WINML_EXPECT_NO_THROW(session = LearningModelSession(model,
    LearningModelDeviceKind::Cpu, options));
}
```

Requirements

Requirement	
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](#) ↗.

NamedDimensionOverride interface

Article • 12/30/2021

Provides the ability to override named input dimensions to concrete values through [LearningModelSessionOptions](#) in order to achieve better runtime performance. Using this API can yield performance improvements, as it allows for preallocation of tensors during session creation that would otherwise be allocated during model evaluation.

Sample Code

C++

```
void SetNamedDimensionOverrides(LearningModel model) {
    // Create LearningModelSessionOptions
    auto options = LearningModelSessionOptions();

    // Override a named input dimension to a concrete value
    options->OverrideNamedDimension(L"dimension_name", 256);

    // Create session
    LearningModelSession session = nullptr;
    session = LearningModelSession(model, LearningModelDeviceKind::GPU,
        options);
}
```

Requirements

Requirement	
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](#).

ILearningModelOperatorProviderNative struct

Article • 12/30/2021

Provides access to [IMLOperatorRegistry](#) objects containing custom operator registrations.

Methods

Name	Description
GetRegistry	Gets an IMLOperatorRegistry object containing custom operator definitions.

Requirements

Requirement	
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](#).

ILearningModelOperatorProviderNative. GetRegistry method

Article • 12/30/2021

Gets an [IMLOperatorRegistry](#) object containing custom operator definitions.

C++

```
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

Requirement	
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](#).

ONNX models

Article • 12/30/2021

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, Caffe2, 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 ONNXMLTools](#): 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](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ONNX versions and Windows builds

Article • 06/01/2022

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 11, version 2104	1.2 - 1.7	7 - 12
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](#).

Train a model with CNTK

Article • 12/30/2021

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.

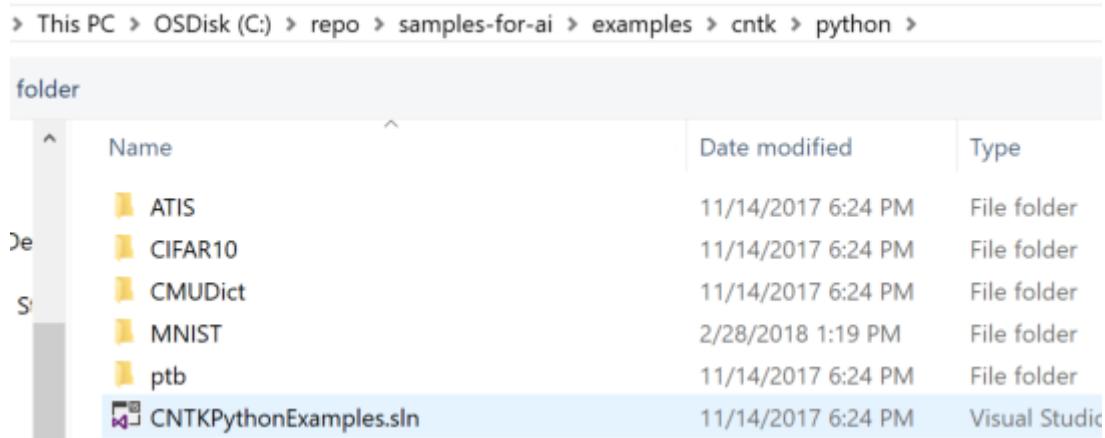
Install CNTK

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.

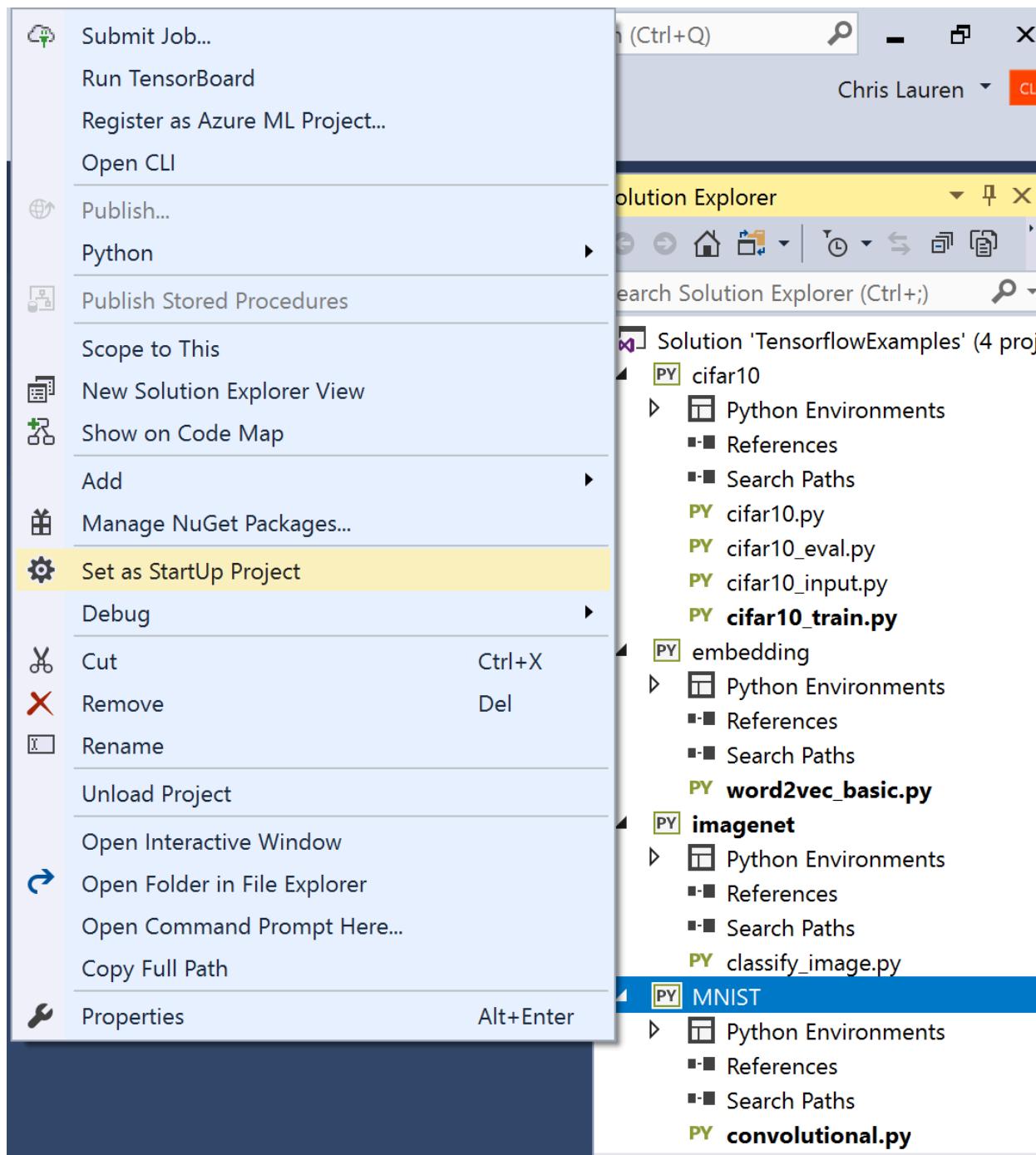
1. Open project

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.



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](#).

 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](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Train a model with PyTorch and export to ONNX

Article • 12/30/2021

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](#).

Windows Machine Learning tutorials

Article • 12/30/2021

Windows Machine Learning can be used in a variety of customizable app solutions. Here, we provide several full tutorials covering how to create a Machine Learning model from a variety of potential non-code or programmatic services, and integrate them into a basic Windows ML app. In addition, we cover several advanced methods to tweak the functionality of your app. And if you're looking for just a basic introductory use of the APIs with an existing model, or if you want to check out our samples, check out further links below.

Full app tutorials

These following tutorials cover the creation of a Machine Learning model, and how to incorporate it into a Windows 10 app with Windows ML.

No-code training environment

Want to use an existing utility to train a machine learning model? These tutorials cover end-to-end walkthroughs of how to create Windows ML apps with models trained by existing services.

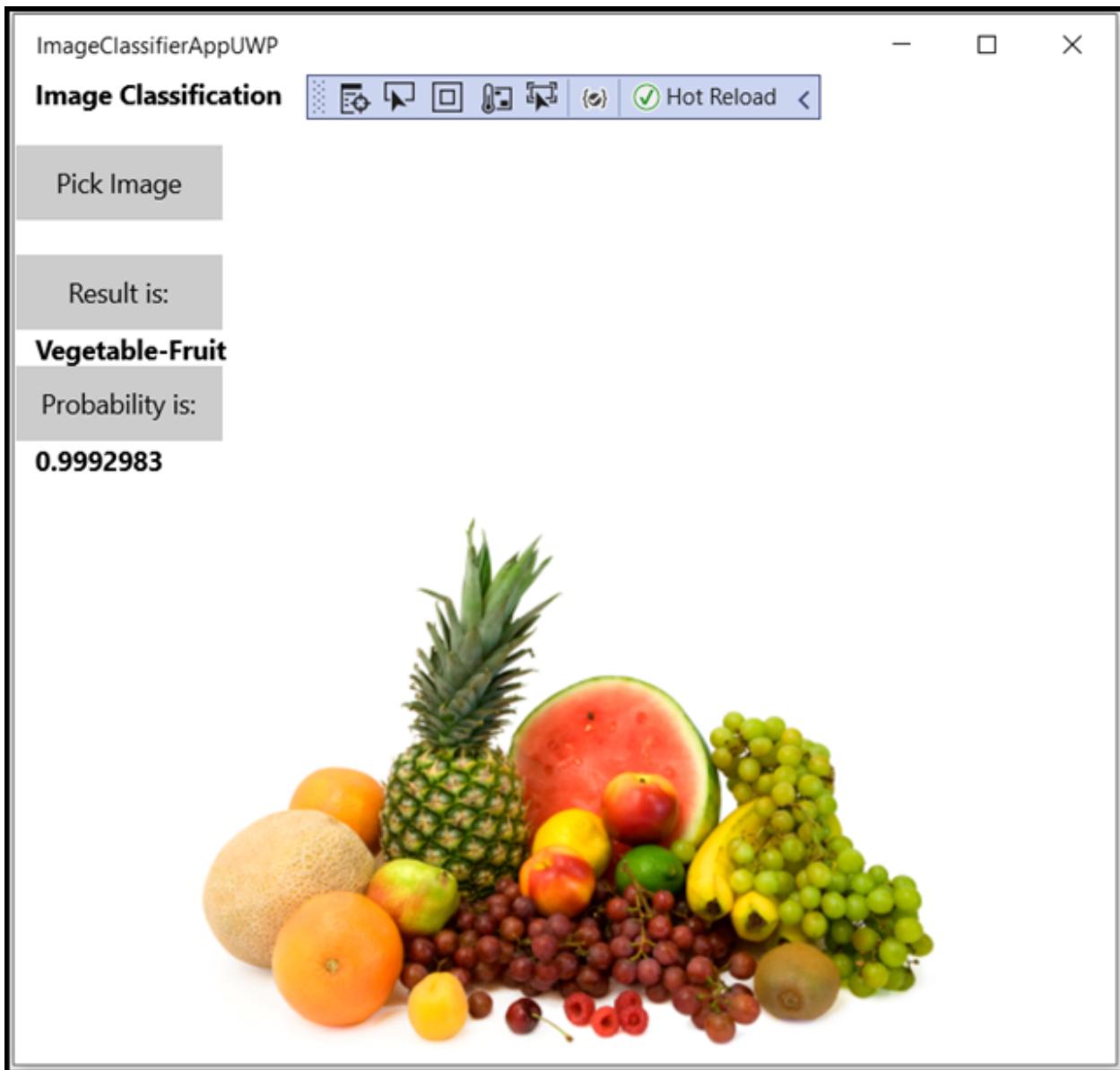


Image classification with Custom Vision and Windows ML

Learn how to use the Azure Custom Vision service to train a model for image classification, and deploy that model in a Windows ML application to run locally on your machine.



Image classification with ML.NET and Windows ML

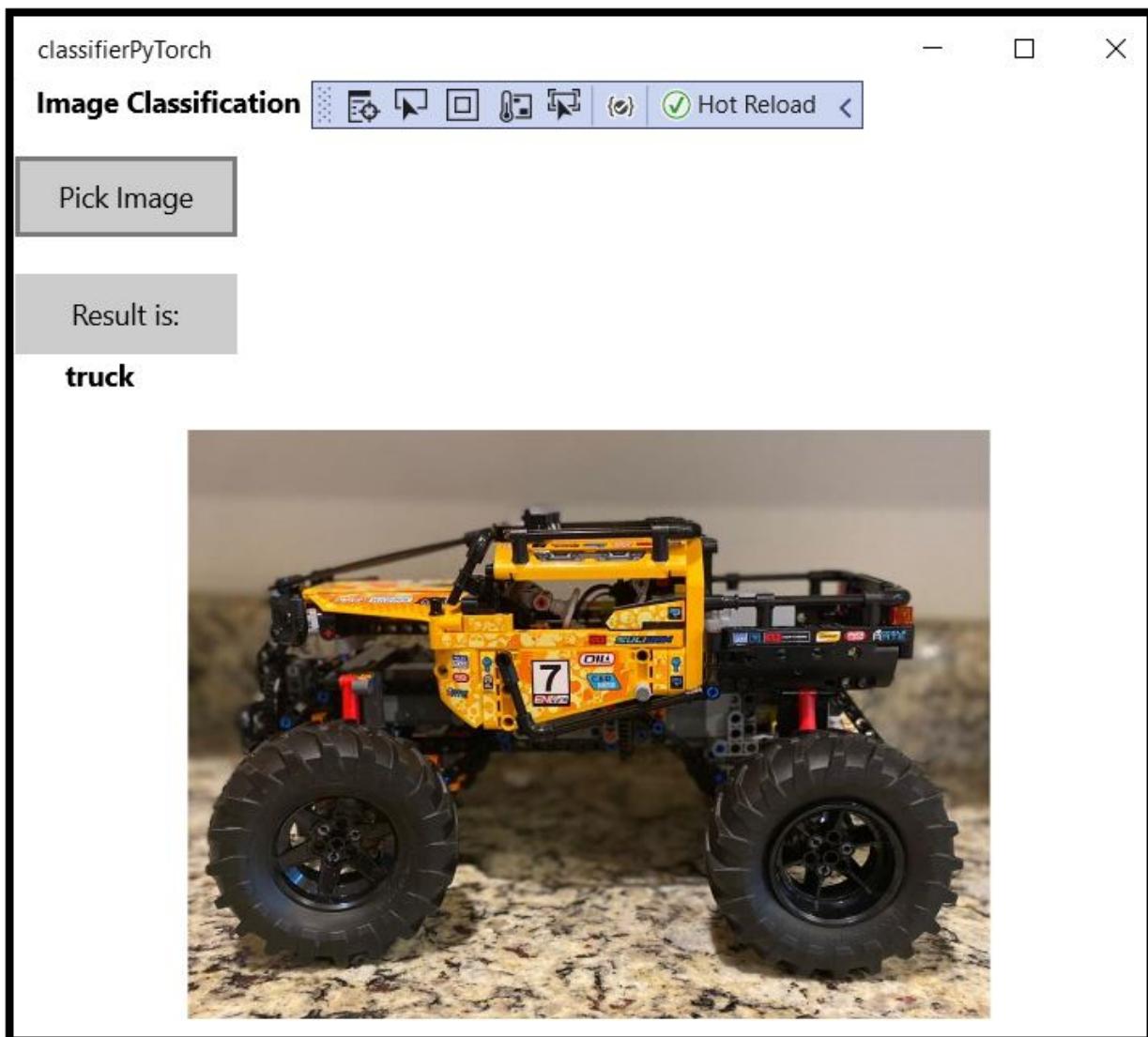
Learn how to use the ML.NET Model Builder Visual Studio extension to create an ONNX model and deploy that model in a Windows ML application to run locally on your machine.

Code training environment

These tutorials cover ways to create your own code to train a Windows ML model, instead of using a pre-existing service.

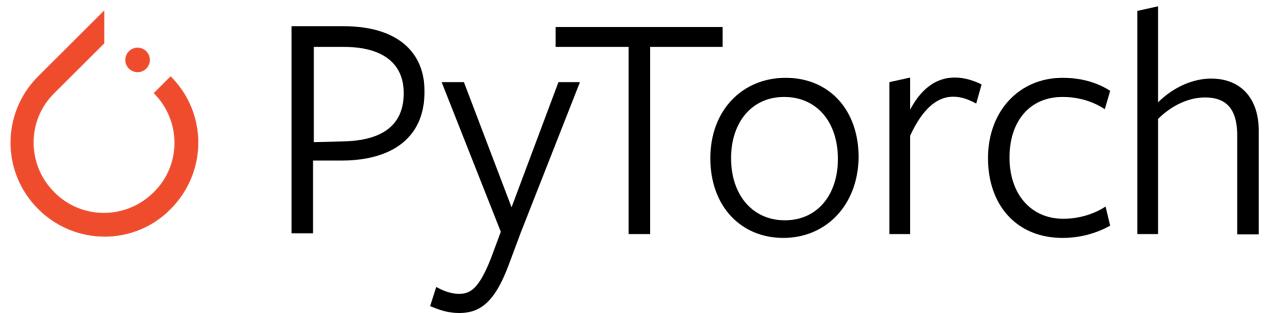
[Image classification with PyTorch and Windows ML](#)

Learn how to install PyTorch on your machine, how to use it to train an image classification model, how to convert that model to the ONNX format, and how to deploy it in a Windows ML application to run locally on your machine.



[Data analysis with PyTorch and Windows ML](#)

Learn how to install PyTorch on your machine, how to use it to train a data analysis model, how to convert that model to the ONNX format, and how to deploy it in a Windows ML application to run locally on your machine.



[Object detection with TensorFlow and Windows ML](#)

Learn how to install TensorFlow on your machine, implement transfer learning with the YOLO architecture, convert to the model to ONNX, and deploy it in a Windows ML application to run locally on your machine.



TensorFlow

Advanced features:

If you want to use the Windows ML NuGet package, please see [Tutorial: Port an Existing Windows ML App to NuGet Package](#).

For the latest Windows ML features and fixes, see our [release notes](#).

Important

PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.
TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

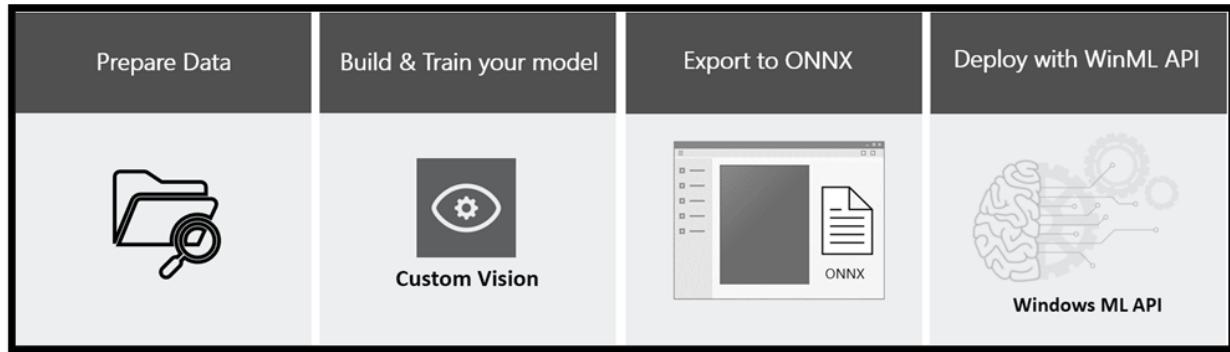
Note

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](#).

Image classification with Custom Vision and Windows Machine Learning

Article • 09/23/2022



This guide shows you how to: train a neural network model to classify images of food using Azure Custom Vision service; export the model to ONNX format; and deploy the model in a Windows Machine Learning (Windows ML) application running locally on a Windows device. You don't need any previous expertise in machine learning! We'll guide you step by step through the process.

If you'd like to learn how to build and train a model with Custom Vision, then you can proceed to [Train a model](#).

If you have a model and you want to learn how to create a Windows ML app from scratch, then see the complete [Windows ML app tutorial](#).

If you want to start with a pre-existing Visual Studio project for a Windows ML app, then you can clone the [Custom Vision and Windows ML](#) tutorial sample app, and use that as your starting point.

Scenario

In this tutorial, we'll create a machine learning food classification application that runs on Windows devices. The model will be trained to recognize certain types of patterns to classify an image of food, and when given an image it will return a classification tag and the associated percentage confidence value of that classification.

Prerequisites for model training

To build and train a model, you'll need a subscription to Azure Custom Vision services.

If you're new to Azure, then you can sign up for an [Azure free account](#). That will give you an opportunity to build, train, and deploy machine learning models with Azure AI.

💡 Tip

Are you interested in learning more about Azure sign-up options and Azure free accounts? Then check out [Create an Azure account](#).

Prerequisites for Windows ML app deployment

To create and deploy a Windows ML app, you'll need the following:

- Windows 10, version 1809 (build 17763) or later. You can check your build version number by running `winver` via the **Run** command (Windows logo key + R).
- Windows SDK for build 17763 or later. To download, see [Windows SDK](#).
- Visual Studio 2017 version 15.7 or later; but we recommend that you use Visual Studio 2022 or later. Some screenshots in this tutorial might differ from the UI you'll see. To download Visual Studio, see [Downloads and tools for Windows development](#).
- Windows ML Code Generator (`m1gen`) Visual Studio extension. Download it for [Visual Studio 2019 or later](#) or for [Visual Studio 2017](#).
- If you decide to create a Universal Windows Platform (UWP) app, then you'll need to enable the Universal Windows Platform development workload in Visual Studio.
- Enable Developer mode on your PC—see [Enable your device for development](#).

⚠ Note

Windows ML APIs are built in to the latest versions of Windows 10 (1809 or higher) and Windows Server 2019. If your target platform is an older version of Windows, then you can [port your Windows ML app to the redistributable NuGet package \(Windows 8.1 or higher\)](#).

Prepare the data

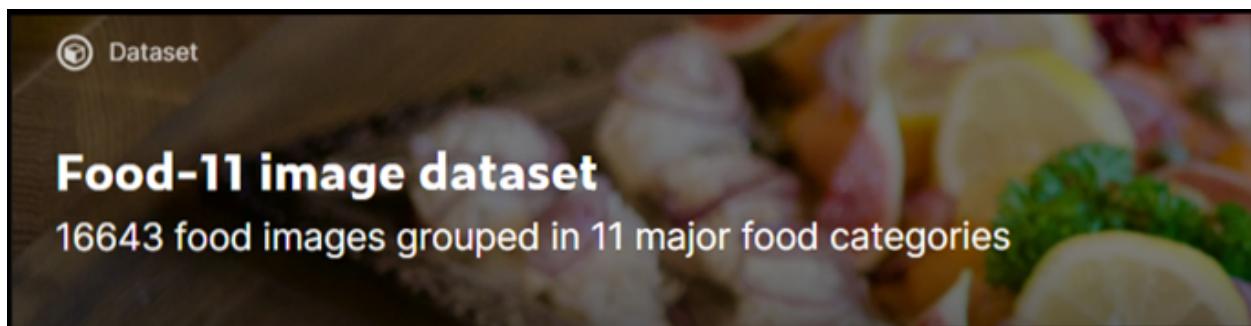
Machine learning models must be trained with existing data. In this guide, you'll use a dataset of food images from Kaggle Open Datasets. That dataset is distributed under the public domain license.

ⓘ Important

To use this dataset, you need to adhere to the term of using the Kaggle site, and the license terms accompanying the [Food-11](#) dataset itself. Microsoft makes no warranty or representation concerning the site or this dataset.

The dataset has three splits—evaluation, training, and validation—and contains 16,643 food images grouped into 11 major food categories. The images in the dataset of each category of food are placed in a separate folder, which makes the model training process more convenient.

Download the dataset from [Food-11 image dataset](#). The dataset is around 1GB in size, and you might be asked to create an account on the Kaggle website to download the data.



If you prefer, you're welcome to use any other dataset of relevant images. As a minimum, we recommend that you use at least 30 images per tag in the initial training set. You'll also want to collect a few extra images to test your model once it's trained.

Additionally, make sure that all of your training images meet the following criteria:

- .jpg, .png, .bmp, or .gif format.
- No greater than 6MB in size (4MB for prediction images).
- No less than 256 pixels on the shortest edge; any images shorter than this will be automatically scaled up by the Custom Vision Service.

Next steps

Now that you've gotten your prerequisites sorted out, and have prepared your dataset, you can proceed to the creation of your Windows ML model. In the next part ([Train your model with Custom Vision](#)), you'll use the web-based Custom Vision interface to create and train your classification model.

Train your model with Custom Vision

Article • 12/30/2021

In the [previous stage of this tutorial](#), we discussed the prerequisites of creating your own Windows Machine Learning model and app, and downloaded an image set to use. In this stage, we'll learn how to use the web-based Custom Vision interface to turn our image set into an image classification model.

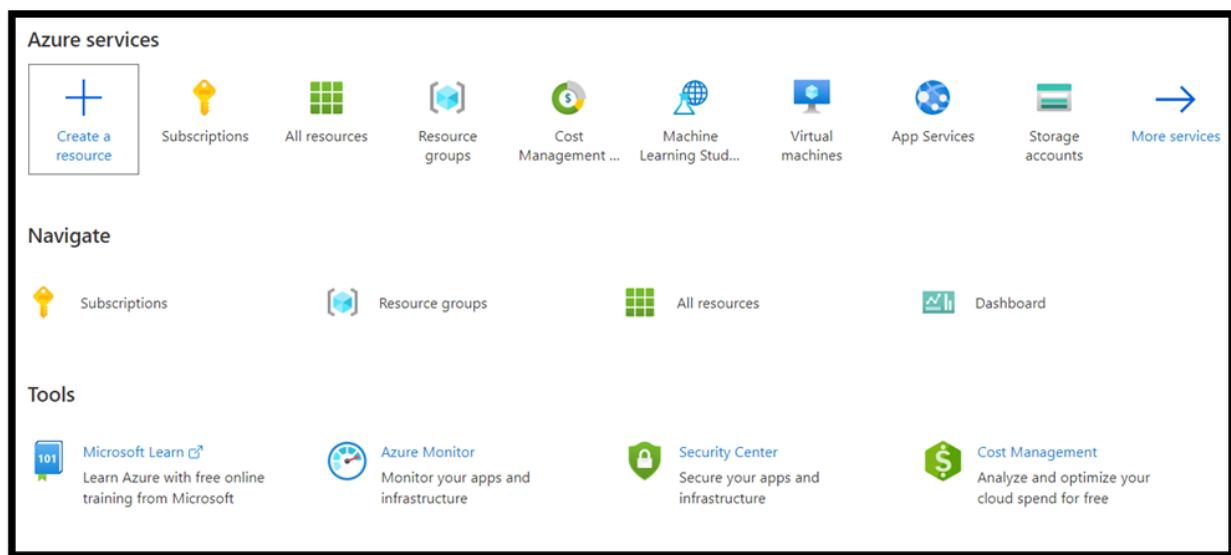
Azure Custom Vision is an image recognition service that lets you build, deploy, and improve your own image identifiers. The Custom Vision Service is available as a set of native SDKs, as well as through a web-based interface on the Custom Vision website.

Create Custom Vision resources and project

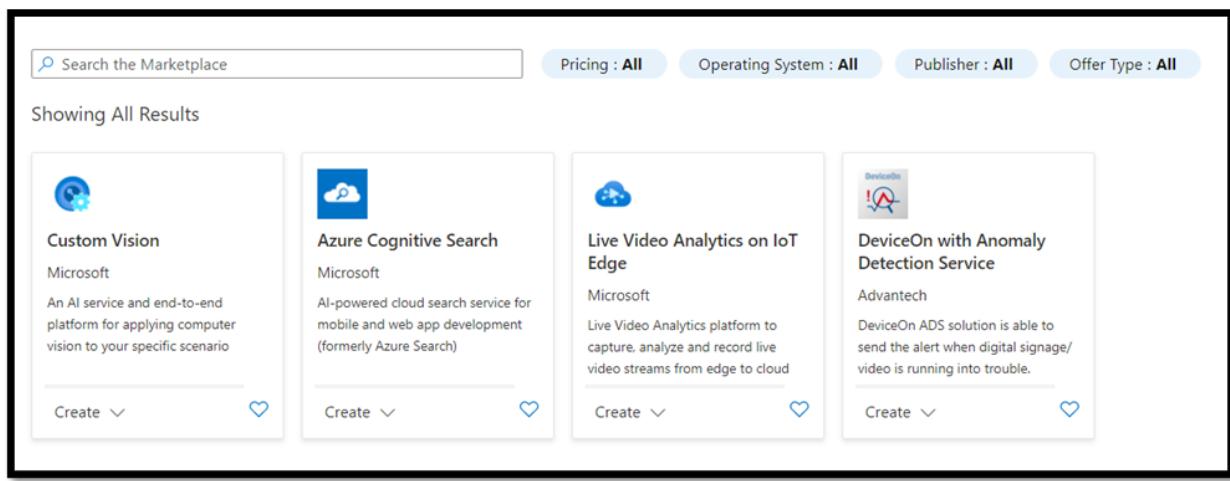
Create Custom Vision resource

To use the Custom Vision Service, you'll need to create Custom Vision resources in Azure.

1. Navigate to the main page of your Azure account and select `Create a resource`.



2. In the search box, search for `Custom Vision`, and you'll enter the Azure marketplace. Select `Create Custom Vision` to open the dialog window on the Create Custom Vision page.



3. On the Custom Vision dialog page, choose the following:

- Select both **Training** and **Prediction** resources.
- Select the subscription to manage deployed resources. If you do not see your Azure Subscription in the menu, sign out and reopen your Azure account using the same credentials you opened your account with.
- Create a new resource group and give it a name. In this tutorial, we've called ours **MLTraining**, but feel free to choose your own name or use the existing resource group if you have one.
- Give a name to your project. In this tutorial, we've called ours **classificationApp**, but you can use any name of your choice.
- For both **Training** and **Prediction** resources, set the location as **(US) East US**, and **Pricing tier as Free FO**.

4. Press **Review + create** to deploy your Custom Vision resources. It may take a few minutes to deploy your resources.

[Home](#) > [New](#) > [Marketplace](#) >

Create

Cognitive Services

[*Basics](#) [Tags](#) [Review + create](#)

Customize and embed state-of-the-art computer vision for specific domains. Build frictionless customer experiences, optimize manufacturing processes, accelerate digital marketing campaigns -- and more. No machine learning expertise is required. [Learn more](#) ↗

Create options

[Both](#) [Training](#) [Prediction](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Resource group * ⓘ

 [Create new](#)

Name * ⓘ

Training Resource

Select pricing and location for Training Resource

Training location *

 Training pricing tier ([Learn More](#)) * ⓘ

Prediction Resource

Select pricing and location for Prediction Resource

Prediction location *

 Prediction pricing tier ([Learn More](#)) * ⓘ [Review + create](#)[Next : Tags >](#)

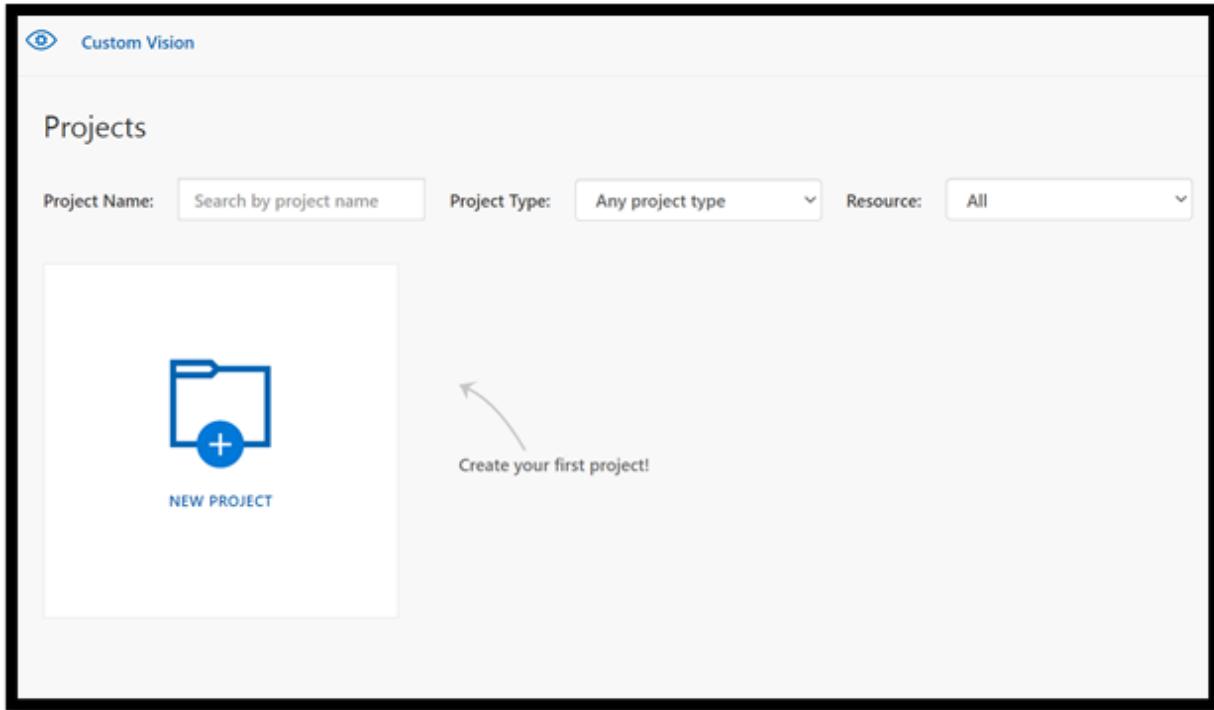
Create a new project within Custom Vision

Now that you've created your resource, it's time to create your training project within Custom Vision.

1. In your web browser, navigate to the [Custom Vision](#) ↗ page and select [Sign in](#).

Sign in with the same account you used to sign into the Azure Portal.

2. Select **New Project** to open a new project dialog.



3. Create a new project as follow:

- **Name:** FoodClassification.
- **Description:** Classification of different types of food.
- **Resource:** keep the same resource you opened previously – `ClassificationApp [F0]`.
- **Project Types:** `classification`
- **Classification Types:** `Multilabel (Multiple tags per image)`
- **Domains:** `Food (compact)`.
- **Export Capabilities:** `Basic platforms (Tensorflow, CoreML, ONNX, ...)`

ⓘ Note

In order to export to the ONNX format, ensure that you choose the `Food (compact)` domain. Non-compact domains cannot be exported to ONNX.

ⓘ Important

If your signed-in account is associated with an Azure account, the Resource Group dropdown will display all of your Azure Resource Groups that include a Custom Vision Service Resource. If no resource group is available, please confirm that you have logged in to [customvision.ai](#) with the same account as you used to log into the Azure Portal.

4. Once you filled the dialog, select `Create project`.

Create new project X

Name*
FoodClassification

Description
Classification of different types of food

Resource [create new](#)
 classificationApp [F0] ▼

[Manage Resource Permissions](#)

Project Types (i)
 Classification
 Object Detection

Classification Types (i)
 Multilabel (Multiple tags per image)
 Multiclass (Single tag per image)

Domains:
 General
 Food
 Landmarks
 Retail
 General (compact)
 Food (compact)
 Landmarks (compact)
 Retail (compact)

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. [Learn More](#)

Export Capabilities: (i)
 Basic platforms (Tensorflow, CoreML, ONNX, ...)
 Vision AI Dev Kit

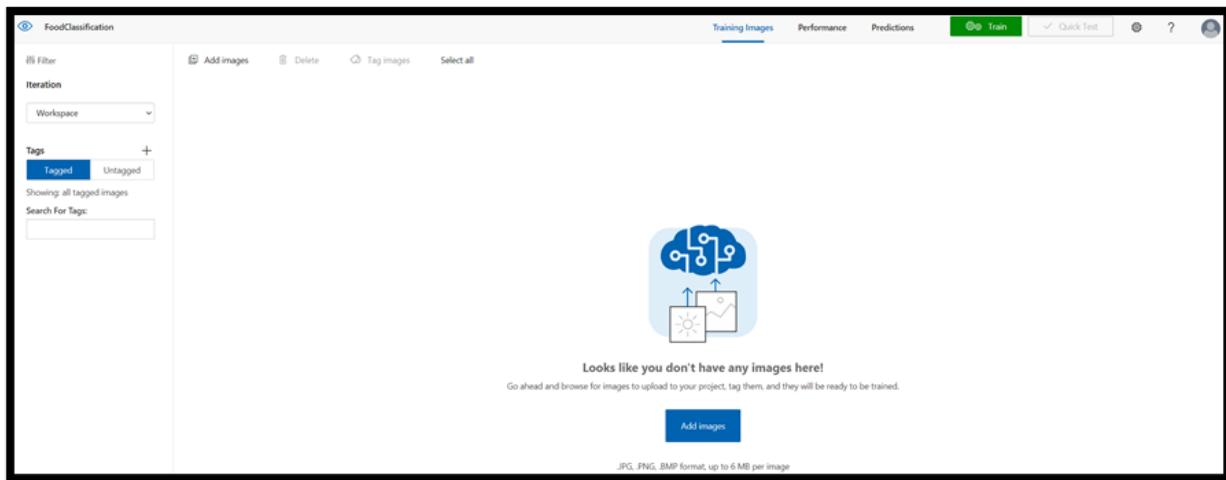
Cancel

Create project

Upload the training dataset

Now that you've created your project, you'll upload a previously prepared dataset of food images from Kaggle Open Datasets.

1. Select your `FoodClassification` project to open the web-based interface of the Custom Vision website.
2. Select the `Add images` button and choose `Browse local files`.



3. Navigate to the location of the image dataset and select the training folder – `vegetable-fruit`. Select all images in the folder, and select `open`. The tagging option will open.
4. Enter `vegetable-fruit` in the `My Tags` field and press `Upload`.

Image upload

X

Add Tags Uploading Summary

709 images will be added...

Add some tags to this batch of images...

My Tags

Vegetable-Fruit

Upload 709 files

Wait until the first group of images are uploaded to your project and then press **done**. The tag selection will be applied to the entire group of images you have selected to upload. That's why it is easier to upload images from already pre-built groups of images. You can always change the tags for individual images after they've been uploaded.

Image upload

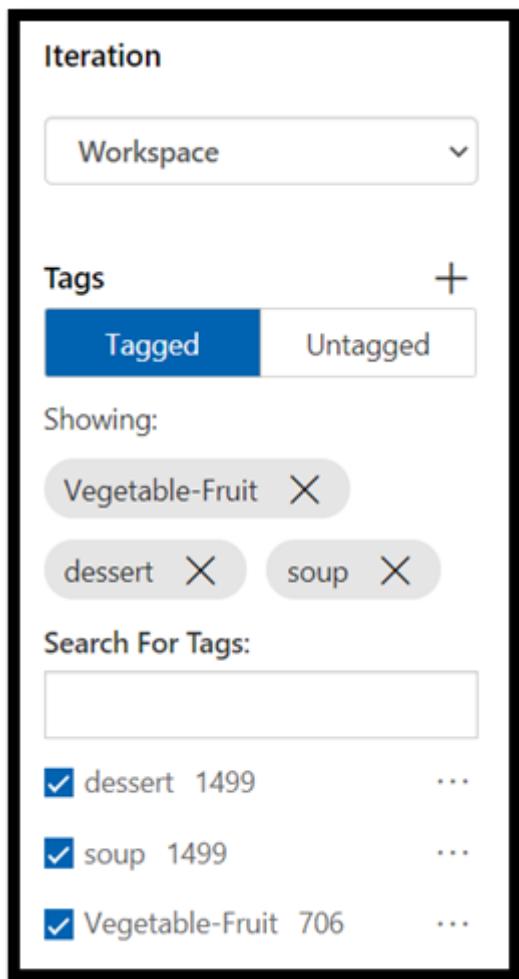
X

Add Tags Uploading Summary

✓ 706 images uploaded successfully

5. After the first group of images successfully uploaded, repeat the process twice more to upload the images of dessert and soup. Make sure you label them with the relevant tags.

At the end, you will have three different groups of images ready for training.



Train the model classifier

You'll now train the model to classify the vegetables, soup, and desserts from the set of images you downloaded in the previous part.

1. To start the training process, select the **Train** button from the upper right corner.
The classifier will use the images to create a model that identifies the visual qualities of each tag.

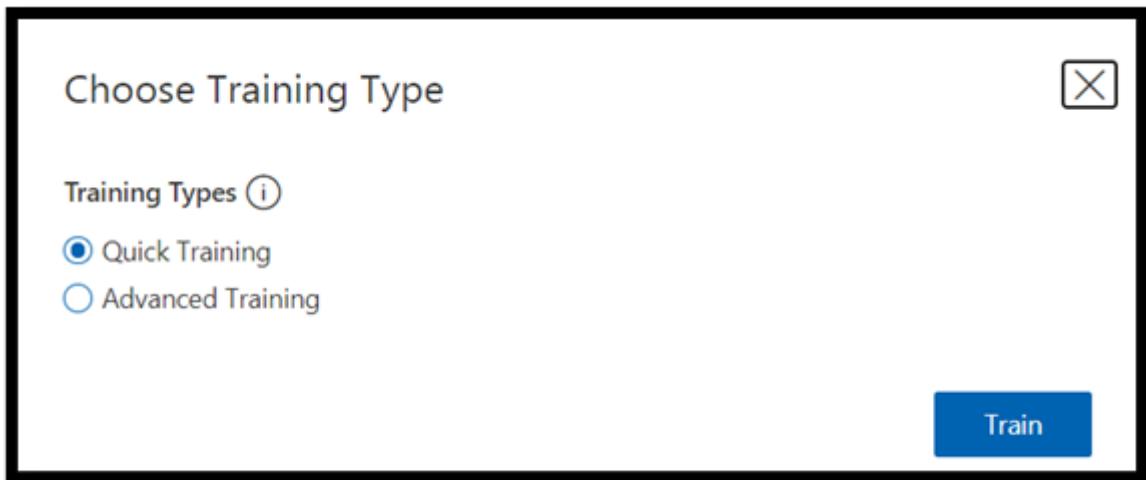


There's an option to change the probability threshold using the slider on the left upper corner. The probability threshold sets the level of confidence that a prediction needs to have in order to be considered correct. If the probability threshold is too high, you'll get

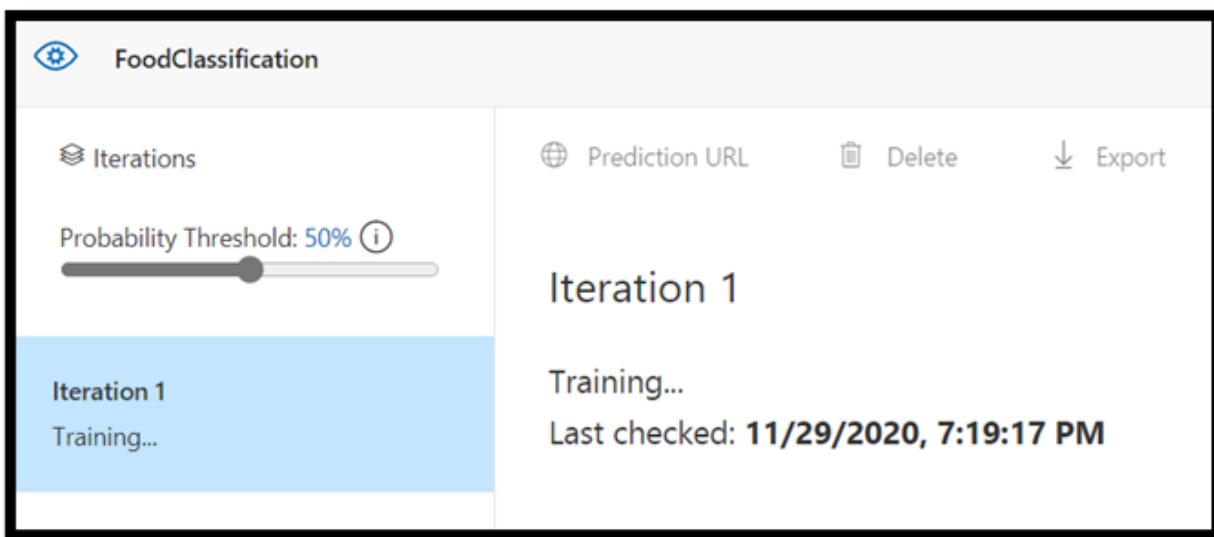
more correct classification, but fewer will be detected. On the other hand, if the probability threshold is too low, you'll detect many more classifications, but with a lower confidence or more false positive results.

In this tutorial, you can keep probability threshold at 50%.

2. Here, we'll use the `Quick Training` process. `Advanced Training` has more settings, and allows you to specifically set the time used for training, but we don't need that level of control here. Press `Train` to initiate the training process.



A quick training process will only take a few minutes to complete. During this time, information about the training process is displayed in the `Performance` tab.

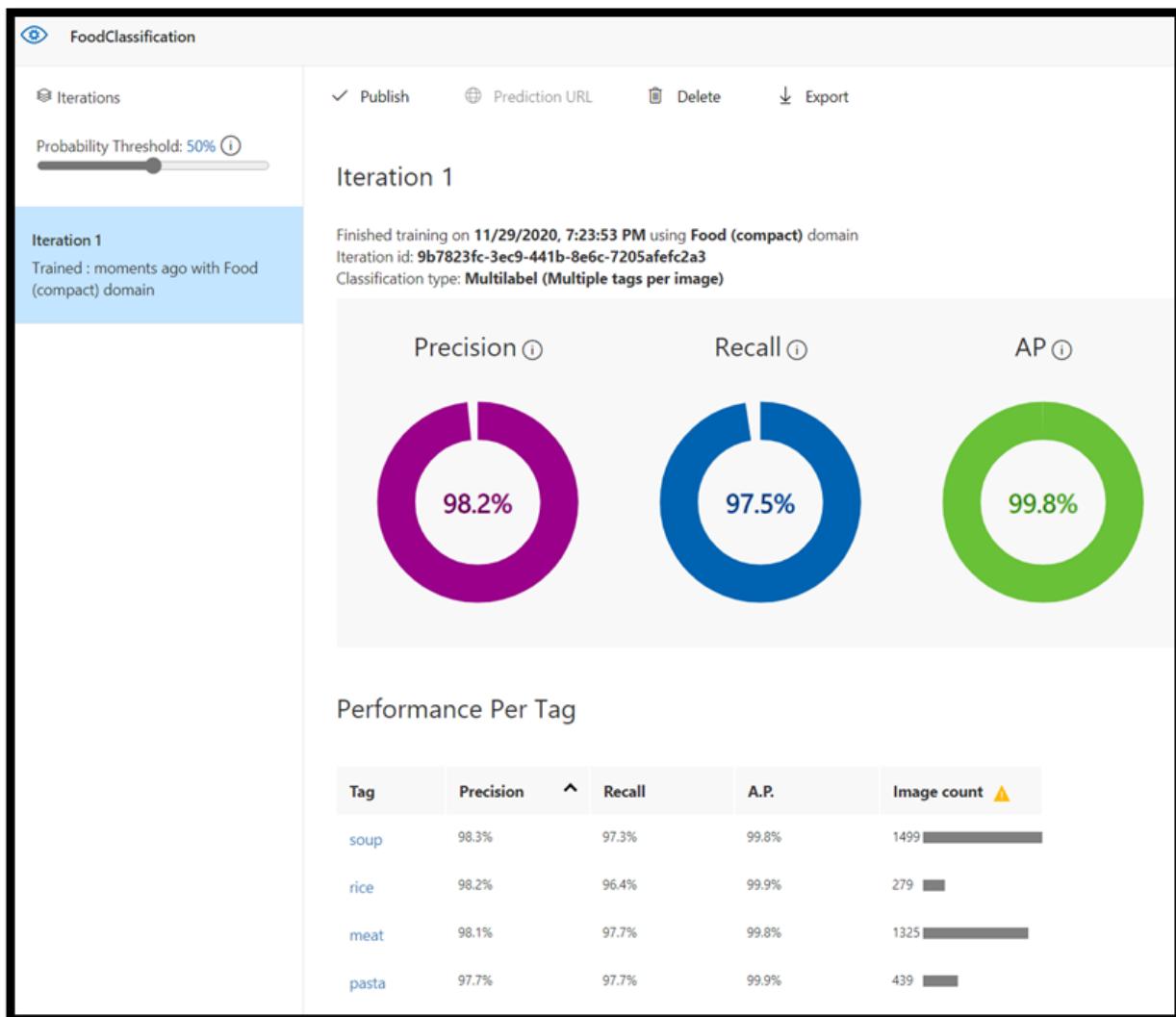


Evaluate and Test

Evaluate the results

Once the training is completed, you'll see the summary of the first training iteration. It includes the estimation of the model performance – **Precision** and **Recall**.

- **Precision** indicates the fraction of identified classifications that were correct. In our model, precision is 98.2%, so if our model classifies an image, it's very likely to be predicted correctly.
- **Recall** indicates the fraction of actual classifications that were correctly identified. In our model, recall is 97.5%, so our model properly classifies the vast majority of images presented to it.
- **AP** stands for Additional Performance. This provides an additional metric, that summarizes the precision and recall at different thresholds.



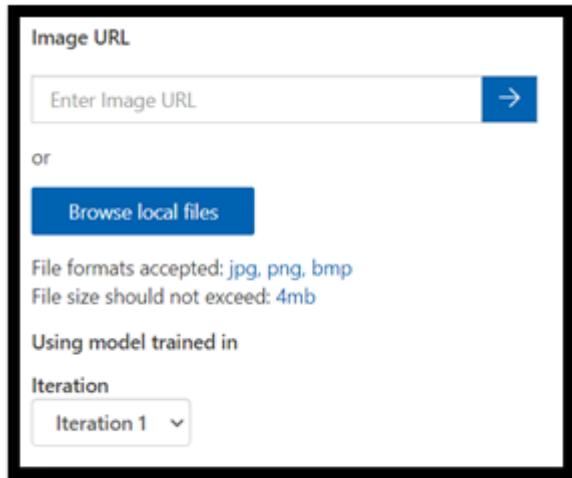
Test the model

Before you export the model, you can test its performance.

1. Select **Quick Test** on the top right corner of the top menu bar, to open a new testing window.

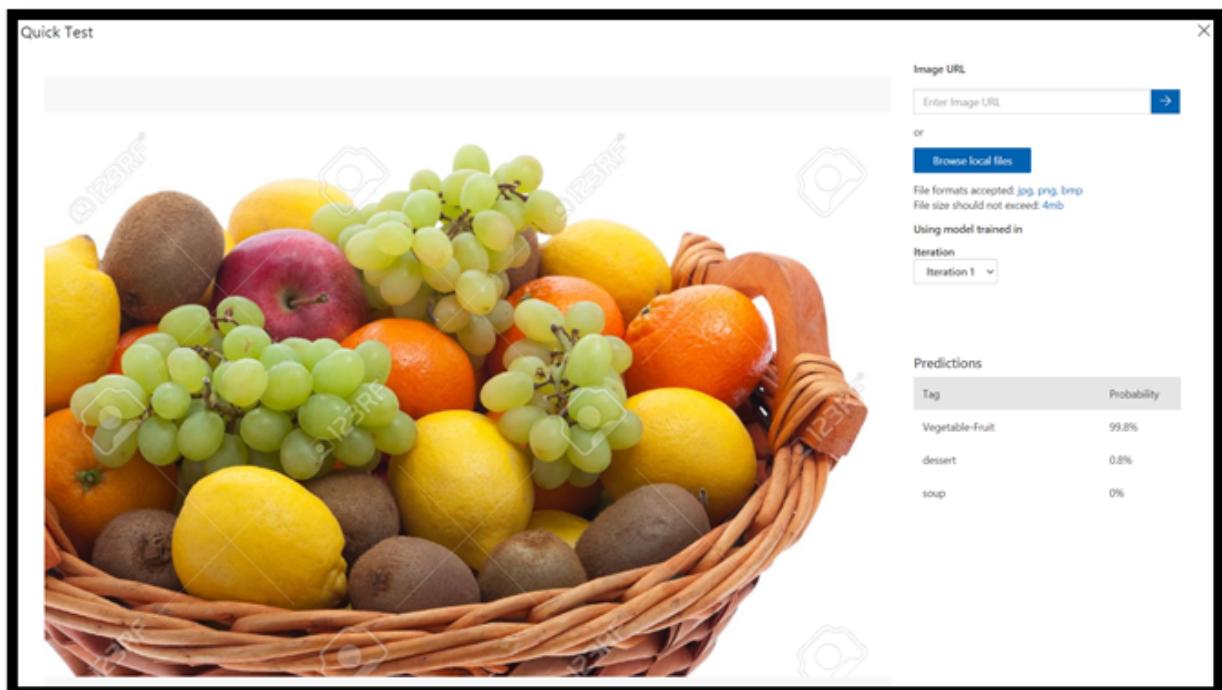
The screenshot shows the top navigation bar of the AI Platform. The 'Training Images' tab is active, indicated by a blue underline. Other tabs like 'Performance' and 'Predictions' are also present. A green 'Train' button with a gear icon is on the left, followed by a white 'Quick Test' button with a checkmark icon. To the right are icons for settings, help, and user profile.

In this window, you can provide a URL of the image to test, or select **Browse local files** to use a locally stored image.



2. Choose **Browse local files**, navigate to the food data set, and open a validation folder. Choose any random image from the **fruit-vegetable** folder and press **open**.

The result of the testing will appear on the screen. In our test, the mode successfully classified the image with 99.8% certainty.



You can use the prediction for training in the **Predictions** tab, which can improve the model performance. For more information, see [How to improve your classifier](#).

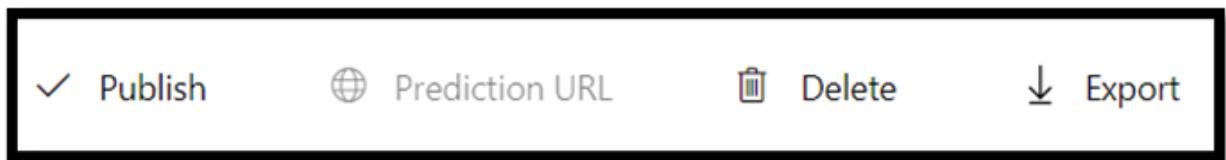
Note

Interested in learning more about Azure Custom Vision APIs? The [Custom Vision Service documentation](#) has more information on Custom Vision web portal and SDK.

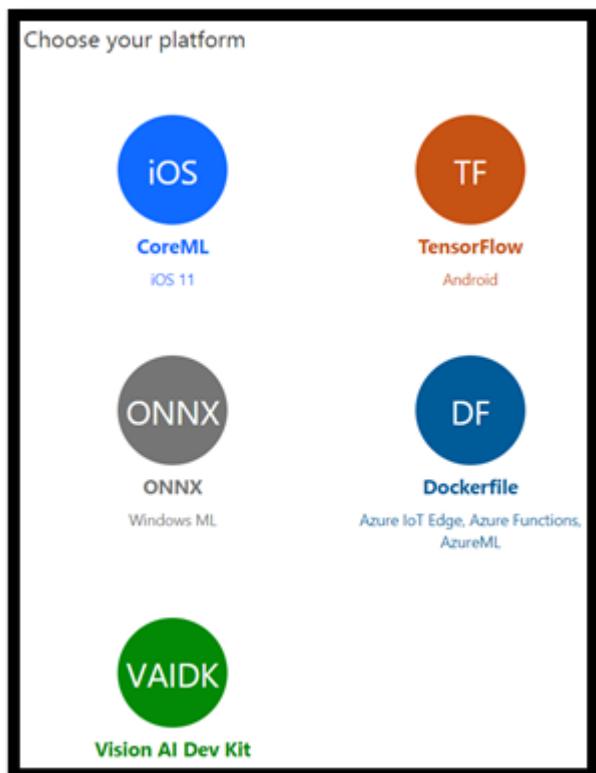
Export the model to ONNX

Now that we've trained our model, we can export it to ONNX.

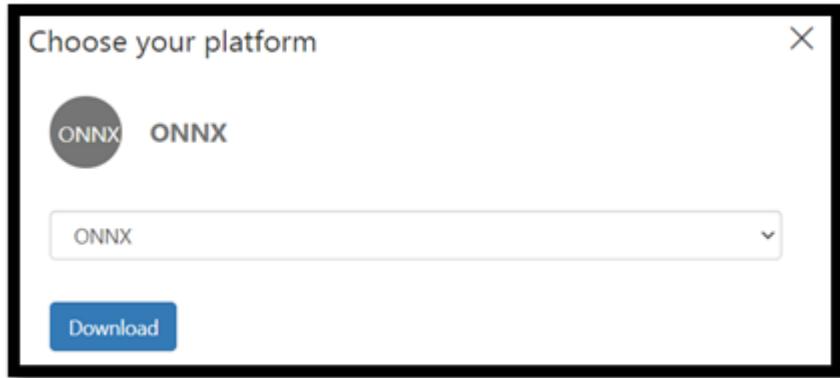
1. Select the `Performance` tab, and then choose `Export` to open an export window.



2. Select `ONNX` to export your model to ONNX format.



3. You can choose the `ONNX 16` float option if required, but in this tutorial we don't need to change any settings. Select `Export` and `Download`.



4. Open the downloaded .zip file and extract the `model.onnx` file from it. This file contains your classifier model.

Congratulations! You've successfully built and exported the classification model.

Next Steps

Now that we have a classification model, the next step is to [build a Windows application and run it locally on your Windows device](#).

Deploy your model in a Windows app with the Windows Machine Learning APIs

Article • 12/30/2021

In the previous part of this tutorial, you learned how to build and export a model in ONNX format. Now that you have that model, you can embed it into a Windows application and run it locally on a device by calling WinML APIs.

Once we're done, you'll have a working Image classifier WinML UWP app (C#).

About the sample app

Using our model, we'll create an app that can classify images of food. It allows you to select an image from your local device and process it by a locally stored classification ONNX model you built and trained in the previous part. The tags returned are displayed next to the image, as well as the the confidence probability of the classification.

If you've been following this tutorial so far, you should already have the necessary prerequisites for app development in place. If you need a refresher, see [the first part of this tutorial](#).

Note

If you prefer to download the complete sample code, you can clone [the solution file](#). Clone the repository, navigate to this sample, then open the `ImageClassifierAppUWP.sln` file with Visual Studio. Then, you can skip to the [Launch the application](#Launch the application) step.

Create a WinML UWP (C#)

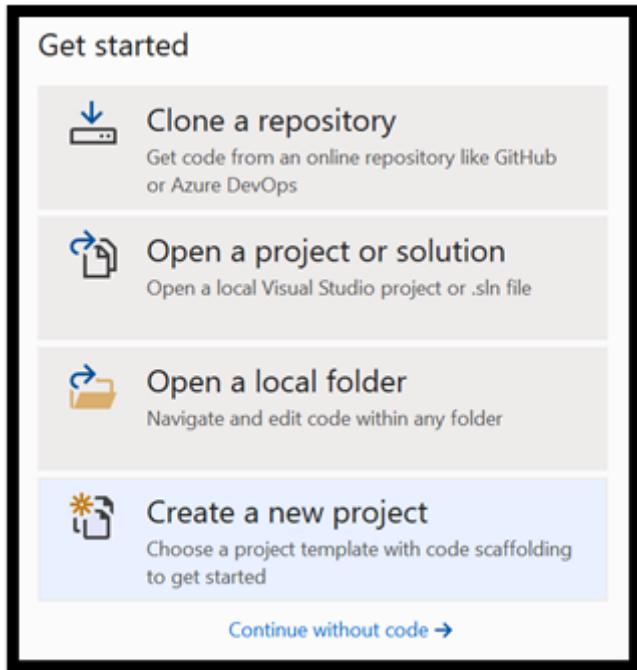
Below, we'll show you how to create your app and WinML code from scratch. You'll learn how to:

- Load a machine learning model.
- Load an image in the required format.
- Bind the model's inputs and outputs.
- Evaluate the model and display meaningful results.

You'll also use basic XAML to create a simple GUI, so you can test the image classifier.

Create the app

1. Open Visual Studio and choose `create a new project`.



2. In the search bar, type `UWP` and then select `Blank APP (Universal Windows)`. This opens a new C# project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout. Select `Next` to open a configuration window for the project.

The screenshot shows the 'Create New Project' dialog in Visual Studio. The search bar at the top contains the text 'UWP'. Below the search bar, there are three dropdown filters: 'C#' (selected), 'Windows' (selected), and 'UWP' (selected). The results list includes:

- Blank App (Universal Windows)**: A project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout. Tags: C#, XAML, Windows, Xbox, Desktop, UWP.
- Unit Test App (Universal Windows)**: A project to create a unit test app for Universal Windows Platform (UWP) apps using MSTest. Tags: C#, Windows, Test, UWP.
- Class Library (Universal Windows)**: A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps. Tags: C#, Windows, Library, UWP.
- Windows Runtime Component (Universal Windows)**: A project for creating a managed Windows Runtime component (.winmd) for Universal Windows Platform (UWP) apps, regardless of the programming languages in which the apps are written. Tags: C#, Windows, Library, UWP.
- Coded UI Test Project (Universal Windows) [Deprecated]**: A deprecated project type.

At the bottom right are 'Back' and 'Next' buttons.

3. In the configuration window:

- Choose a name for your project. Here, we use **ImageClassifierAppUWP**.
- Choose the location of your project.
- If you're using VS 2019, ensure **Place solution and project in the same directory** is unchecked.
- If you're using VS 2017, ensure **Create directory for solution** is checked.

Press **Create** to create your project. The minimum target version window may pop up. Be sure your minimum version is set to **Windows 10 build 17763** or later.

To create an app and deploy a model with a WinML app, you'll need the following:

4. After the project was created, navigate to the project folder, open the assets folder **[...]\ImageClassifierAppUWP\Assets**, and copy your model to this location.

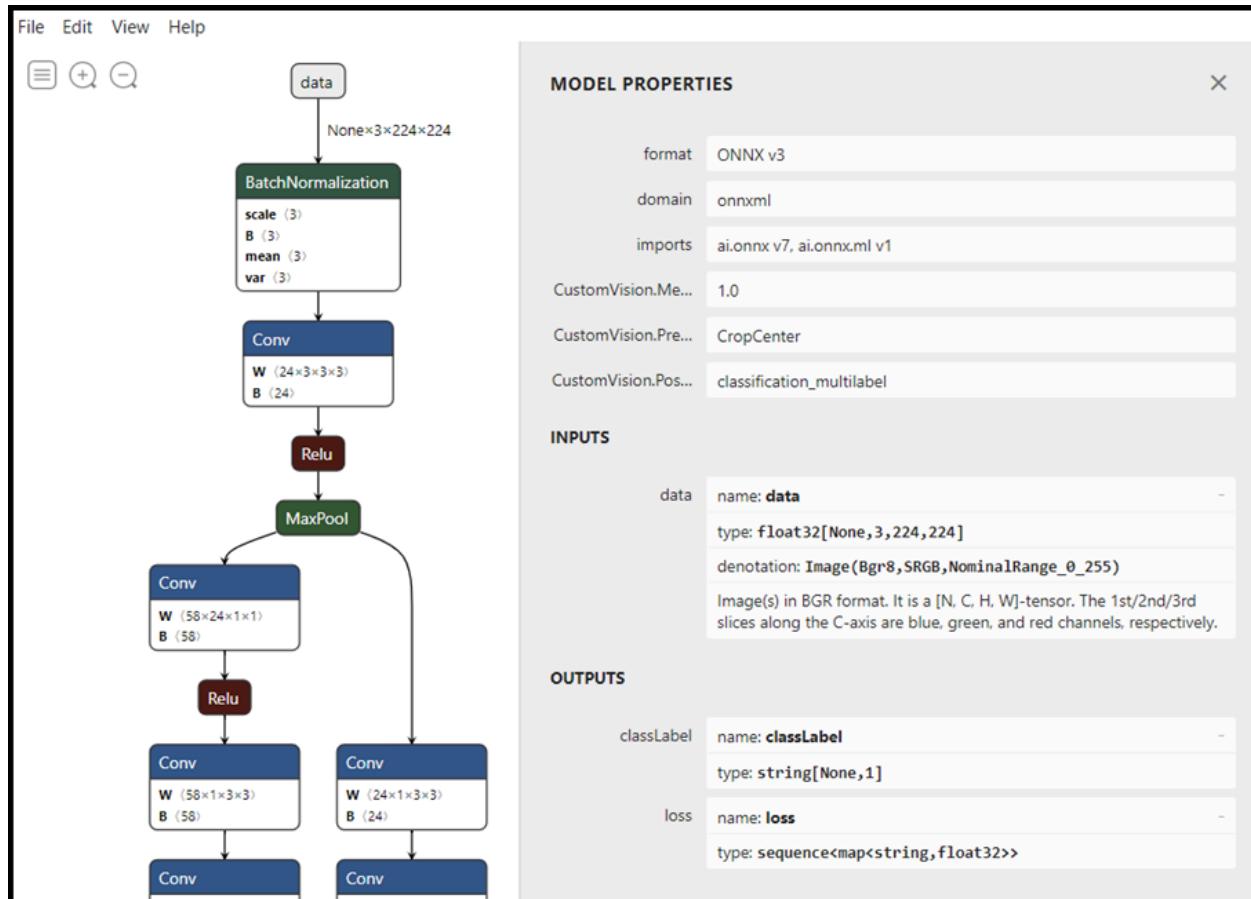
5. Change the model name from `model.onnx` to `classifier.onnx`. This makes things a little clearer, and aligns it to the format of the tutorial.

Explore your model

Let's get familiar with the structure of your Model file.

1. Open your `classifier.onnx` model file, using [Netron](#).

2. Press `Data` to open the model properties.



As you can see, the model requires 32-bit Tensor (multidimensional array) float object as an input and returns two outputs: the first named `classLabel` is tensor of strings and the second named `loss` is a sequence of string-to-float maps that describe the probability for each labeled classification. You'll need this information to successfully display the model output in your Windows app.

Explore project solution

Let's explore your project solution.

Visual Studio automatically created several cs-code files inside the Solution Explorer. `MainPage.xaml` contains the XAML code for your GUI, and `MainPage.xaml.cs` contains

your application code. If you've created a UWP app before, these files should be very familiar to you.

Create the Application GUI

First, let's create a simple GUI for your app.

1. Double-click on the `MainPage.xaml` file. In your blank app, the XAML template for your app's GUI is empty, so we'll need to add some UI features.
2. Add the below code to the main body of `MainPage.xaml`.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <StackPanel Margin="1,0,-1,0">
        <TextBlock x:Name="Menu"
            FontWeight="Bold"
            TextWrapping="Wrap"
            Margin="10,0,0,0"
            Text="Image Classification"/>
        <TextBlock Name="space" />
        <Button Name="recognizeButton"
            Content="Pick Image"
            Click="OpenFileButton_Click"
            Width="110"
            Height="40"
            IsEnabled="True"
            HorizontalAlignment="Left"/>
        <TextBlock Name="space3" />
        <Button Name="Output"
            Content="Result is:"
            Width="110"
            Height="40"
            IsEnabled="True"
            HorizontalAlignment="Left"
            VerticalAlignment="Top">
            </Button>
        <!--Display the Result-->
        <TextBlock Name="displayOutput"
            FontWeight="Bold"
            TextWrapping="Wrap"
            Margin="30,0,0,0"
            Text="" Width="1471" />
        <Button Name="ProbabilityResult"
            Content="Probability is:"
            Width="110"
            Height="40"
            IsEnabled="True"
            HorizontalAlignment="Left"/>
```

```
<!--Display the Result-->
<TextBlock Name="displayProbability"
    FontWeight="Bold"
    TextWrapping="Wrap"
    Margin="30,0,0,0"
    Text="" Width="1471" />
<TextBlock Name="space2" />
<!--Image preview -->
<Image Name="UIPreviewImage" Stretch="Uniform" MaxWidth="300"
MaxHeight="300"/>
</StackPanel>
</Grid>
```

Windows Machine Learning Code Generator

Windows Machine Learning Code Generator, or `mlgen`, is a Visual Studio extension to help you get started using WinML APIs on UWP apps. It generates template code when you add a trained ONNX file into the UWP project.

Windows Machine Learning's code generator `mlgen` creates an interface (for C#, C++/WinRT, and C++/CX) with wrapper classes that call the Windows ML API for you. This allows you to easily load, bind, and evaluate a model in your project. We'll use it in this tutorial to handle many of those functions for us.

Code generator is available for Visual Studio 2017 and later. Please be aware that 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. If you've been following this tutorial from the introduction, you will have already handled this, but if not, you should download for [VS 2019](#) or for [VS 2017](#).

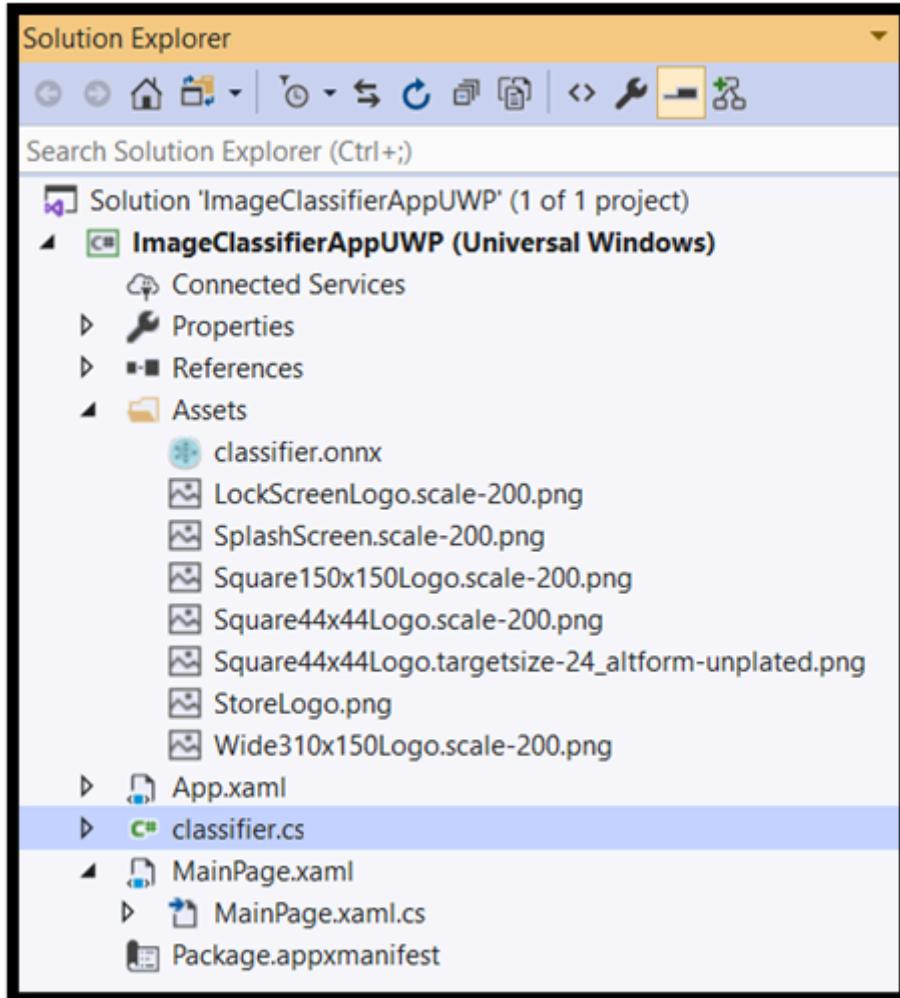
① Note

To learn more about `mlgen`, please see the [mlgen documentation](#)

1. If you haven't already, install `mlgen`.
2. Right-click on the `Assets` folder in the Solution Explorer in Visual studio, and select `Add > Existing Item`.
3. Navigate to the assets folder inside `ImageClassifierAppUWP` `[...\\ImageClassifierAppUWP\\Assets]`, find the ONNX model you previously copied there, and select `add`.

4. After you added an ONNX model (name: "classifier") to the assets folder in solution explorer in VS, the project should now have two new files:

- `classifier.onnx` - this is your model in ONNX format.
- `classifier.cs` – automatically generated WinML code file.



5. To make sure the model builds when you compile our application, select the `classifier.onnx` file and choose `Properties`. For `Build Action`, select `Content`.

Now, let's explore the newly generated code in the `classifier.cs` file.

The generated code includes three classes:

- `classifierModel`: This class includes two methods for model instantiation and model evaluation. It will help us to create the machine learning model representation, create a session on the system default device, bind the specific inputs and outputs to the model, and evaluate the model asynchronously.
- `classifierInput`: This class initializes the input types that the model expects. The model input depends on the model requirements for input data. In our case, the input expects an `ImageFeatureValue`, a class which describes the properties of the image used to pass into a model.

- `classifierOutput`: This class initializes the types that the model will output. The model output depends on how it is defined by the model. In our case, the output will be a sequence of map (dictionaries) of type String and TensorFloat (Float32) called loss.

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

Load the model & inputs

Load the model

1. Double-click on the `MainPage.xaml.cs` code file to open the application code.
2. Replace the “using” statements with the following, to get an access to all the APIs that you'll need.

C#

```
// Specify all the using statements which give us the access to all the APIs
// that you'll need
using System;
using System.Threading.Tasks;
using Windows.AI.MachineLearning;
using Windows.Graphics.Imaging;
using Windows.Media;
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.Storage.Streams;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Imaging;
```

3. Add the following variable declarations after the using statements inside your `MainPage` class, under the namespace `ImageClassifierAppUWP`.

C#

```
// All the required variable declaration
private classifierModel modelGen;
private classifierInput input = new classifierModelInput();
private classifierOutput output;
private StorageFile selectedStorageFile;
private string result = "";
private float resultProbability = 0;
```

The result will look as follows.

C#

```
// Specify all the using statements which give us the access to all the APIs  
// that we'll need  
using System;  
using System.Threading.Tasks;  
using Windows.AI.MachineLearning;  
using Windows.Graphics.Imaging;  
using Windows.Media;  
using Windows.Storage;  
using Windows.Storage.Pickers;  
using Windows.Storage.Streams;  
using Windows.UI.Xaml;  
using Windows.UI.Xaml.Controls;  
using Windows.UI.Xaml.Media.Imaging;  
  
namespace ImageClassifierAppUWP  
{  
    public sealed partial class MainPage : Page  
    {  
        // All the required fields declaration  
        private classifierModel modelGen;  
        private classifierInput input = new classifierInput();  
        private classifierOutput output;  
        private StorageFile selectedStorageFile;  
        private string result = "";  
        private float resultProbability = 0;
```

Now, you'll implement the `LoadModel` method. The method will access the ONNX model and store it in memory. Then, you'll use the `CreateFromStreamAsync` method to instantiate the model as a `LearningModel` object. The `LearningModel` class represents a trained machine learning model. Once instantiated, the `LearningModel` is the initial object you use to interact with Windows ML.

To load the model, you can use several static methods in the `LearningModel` class. In this case, you'll use the `CreateFromStreamAsync` method.

The `CreateFromStreamAsync` method was automatically created with mlgen, so you don't need to implement this method. You can review this method by double clicking on the `classifier.cs` file generated file by mlgen.

To learn more about `LearningModel` class, please review the [LearningModel Class documentation](#). To learn more about additional ways of loading the model, please review the [Load a model documentation](#)

4. Add a `loadModel` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
private async Task loadModel()
{
    // Get an access the ONNX model and save it in memory.
    StorageFile modelFile = await
StorageFile.GetFileFromApplicationUriAsync(new Uri($"ms-
appx:///Assets/classifier.onnx"));
    // Instantiate the model.
    modelGen = await
classifierModel.CreateFromStreamAsync(modelFile);
}
```

5. Now, add a call to the new method to the constructor of the class.

C#

```
// The main page to initialize and execute the model.
public MainPage()
{
    this.InitializeComponent();
    loadModel();
}
```

The result will look as follows.

C#

```
// The main page to initialize and execute the model.
public MainPage()
{
    this.InitializeComponent();
    loadModel();
}

// A method to load a machine learning model.
private async Task loadModel()
{
    // Get an access the ONNX model and save it in memory.
    StorageFile modelFile = await
StorageFile.GetFileFromApplicationUriAsync(new Uri($"ms-
appx:///Assets/classifier.onnx"));
    // Instantiate the model.
    modelGen = await
classifierModel.CreateFromStreamAsync(modelFile);
}
```

Load the Image

1. We need to define a click event to initiate the sequence of four method calls for model execution – conversion, binding and evaluation, output extraction and display the results. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
// Waiting for a click event to select a file
private async void OpenFileButton_Click(object sender,
RoutedEventArgs e)
{
    if (!await getImage())
    {
        return;
    }
    // After the click event happened and an input selected, begin
    // the model execution.
    // Bind the model input
    await imageBind();
    // Model evaluation
    await evaluate();
    // Extract the results
    extractResult();
    // Display the results
    await displayResult();
}
```

2. Now, you'll implement the `getImage()` method. This method will select an input image file and save it in memory. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
// A method to select an input image file
private async Task<bool> getImage()
{
    try
    {
        // Trigger file picker to select an image file
        FileOpenPicker fileOpenPicker = new FileOpenPicker();
        fileOpenPicker.SuggestedStartLocation =
PickerLocationId.PicturesLibrary;
        fileOpenPicker.FileTypeFilter.Add(".jpg");
        fileOpenPicker.FileTypeFilter.Add(".png");
        fileOpenPicker.ViewMode = PickerViewMode.Thumbnail;
        selectedStorageFile = await
fileOpenPicker.PickSingleFileAsync();
        if (selectedStorageFile == null)
        {
            return false;
        }
    }
}
```

```

        }
    }
    catch (Exception)
    {
        return false;
    }
    return true;
}

```

Now, you will implement an image `Bind()` method to get the representation of the file in the bitmap BGRA8 format.

3. Add the implementation of the `convert()` method to your `MainPage.xaml.cs` code file inside the `MainPage` class. The `convert` method will get us a representation of the input file in a BGRA8 format.

C#

```

// A method to convert and bind the input image.
private async Task imageBind()
{
    UIPreviewImage.Source = null;
    try
    {
        SoftwareBitmap softwareBitmap;
        using (IRandomAccessStream stream = await
selectedStorageFile.OpenAsync(FileAccessMode.Read))
        {
            // Create the decoder from the stream
            BitmapDecoder decoder = await
BitmapDecoder.CreateAsync(stream);
            // Get the SoftwareBitmap representation of the file in
BGRA8 format
            softwareBitmap = await decoder.GetSoftwareBitmapAsync();
            softwareBitmap = SoftwareBitmap.Convert(softwareBitmap,
BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied);
        }
        // Display the image
        SoftwareBitmapSource imageSource = new
SoftwareBitmapSource();
        await imageSource.SetBitmapAsync(softwareBitmap);
        UIPreviewImage.Source = imageSource;
        // Encapsulate the image within a VideoFrame to be bound and
evaluated
        VideoFrame inputImage =
VideoFrame.CreateWithSoftwareBitmap(softwareBitmap);
        // bind the input image
        ImageFeatureValue imageTensor =
ImageFeatureValue.CreateFromVideoFrame(inputImage);
        input.data = imageTensor;
    }
    catch (Exception e)

```

```
        }  
    }  
}
```

The result of the work done in this section will look as follows.

C#

```
// Waiting for a click event to select a file  
private async void OpenFileButton_Click(object sender,  
RoutedEventArgs e)  
{  
    if (!await getImage())  
    {  
        return;  
    }  
    // After the click event happened and an input selected, we  
begin the model execution.  
    // Bind the model input  
    await imageBind();  
    // Model evaluation  
    await evaluate();  
    // Extract the results  
    extractResult();  
    // Display the results  
    await displayResult();  
}  
  
// A method to select an input image file  
private async Task<bool> getImage()  
{  
    try  
    {  
        // Trigger file picker to select an image file  
        FileOpenPicker fileOpenPicker = new FileOpenPicker();  
        fileOpenPicker.SuggestedStartLocation =  
PickerLocationId.PicturesLibrary;  
        fileOpenPicker.FileTypeFilter.Add(".jpg");  
        fileOpenPicker.FileTypeFilter.Add(".png");  
        fileOpenPicker.ViewMode = PickerViewMode.Thumbnail;  
        selectedStorageFile = await  
fileOpenPicker.PickSingleFileAsync();  
        if (selectedStorageFile == null)  
        {  
            return false;  
        }  
    }  
    catch (Exception)  
    {  
        return false;  
    }  
    return true;  
}
```

```

// A method to convert and bind the input image.
private async Task imageBind()
{
    UIPreviewImage.Source = null;

    try
    {
        SoftwareBitmap softwareBitmap;
        using (IRandomAccessStream stream = await
selectedStorageFile.OpenAsync(FileAccessMode.Read))
        {
            // Create the decoder from the stream
            BitmapDecoder decoder = await
BitmapDecoder.CreateAsync(stream);

            // Get the SoftwareBitmap representation of the file in
BGRA8 format
            softwareBitmap = await decoder.GetSoftwareBitmapAsync();
            softwareBitmap = SoftwareBitmap.Convert(softwareBitmap,
BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied);
        }
        // Display the image
        SoftwareBitmapSource imageSource = new
SoftwareBitmapSource();
        await imageSource.SetBitmapAsync(softwareBitmap);
        UIPreviewImage.Source = imageSource;

        // Encapsulate the image within a VideoFrame to be bound and
evaluated
        VideoFrame inputImage =
VideoFrame.CreateWithSoftwareBitmap(softwareBitmap);

        // bind the input image
        ImageFeatureValue imageTensor =
ImageFeatureValue.CreateFromVideoFrame(inputImage);
        input.data = imageTensor;
    }
    catch (Exception e)
    {
    }
}

```

Bind and Evaluate the model

Next, you'll create a session based on the model, bind the input and output from the session, and evaluate the model.

Create a session to bind the model:

To create a session, you use the `LearningModelSession` class. This class is used to evaluate machine learning models, and binds the model to a device that then runs and evaluates the model. You can select a device when you create a session to execute your model on a specific device of your machine. The default device is the CPU.

ⓘ Note

To learn more about how to choose a device, please review the [Create a session](#) documentation.

Bind model inputs and outputs:

To bind input and output, you use the `LearningModelBinding` class. A machine learning model has input and output features, which pass information into and out of the model. Be aware that required features must be supported by the Window ML APIs. The `LearningModelBinding` class is applied on a `LearningModelSession` to bind values to named input and output features.

The implementation of the binding is automatically generated by mlgen, so you don't have to take care of it. The binding is implemented by calling the predefined methods of the `LearningModelBinding` class. In our case, it uses the `Bind` method to bind a value to the named feature type.

Currently, Windows ML supports all ONNX feature types like Tensors (multi-dimensional arrays), Sequences (vectors of values), Map (value pairs of information) and Images (specific formats). All images will be represented in Windows ML in a tensor format. Tensorization is the process of converting an image into a tensor and happens during bind.

Fortunately, you don't have to take care of tensorization conversion. The `ImageFeatureValue` method you used in the previous part takes care of both conversion and tensorization, so the images match the model's required image format.

ⓘ Note

To learn more about how to bind a `LearningModel` and about types of features supported by WinML, please review the [Bind a model](#) documentation.

Evaluate the model:

After you create a session to bind the model and bounded values to a model's inputs and outputs, you can evaluate the model's inputs and get its predictions. To run the model execution, you should call any of the predefined evaluate methods on the `LearningModelSession`. In our case, we'll use the `EvaluateAsync` method.

Similar to `CreateFromStreamAsync`, the `EvaluateAsync` method was also automatically generated by WinML Code Generator, so you don't need to implement this method. You can review this method in the `classifier.cs` file.

The `EvaluateAsync` method will asynchronously evaluate the machine learning model using the feature values already bound in bindings. It will create a session with `LearningModelSession`, bind the input and output with `LearningModelBinding`, execute the model evaluation, and get the output features of the model using the `LearningModelEvaluationResult` class.

ⓘ Note

To learn about other evaluation methods to run the model, please check which methods can be implemented on the `LearningModelSession` by reviewing the [LearningModelSession Class documentation](#).

1. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class to create a session, bind and evaluate the model.

C#

```
// A method to evaluate the model
private async Task evaluate()
{
    output = await modelGen.EvaluateAsync(input);
}
```

Extract and display the results

You will now need to extract the model output and display the right results. You will do it by implementing the `extractResult` and `displayResult` methods.

As you explored earlier, the model returns two outputs: the first named `classLabel` is tensor of strings and the second named `loss` is a sequence of string-to-float maps that describes the probability for each labeled classification. So, to successfully display the

result and the probability, all we need is to extract the output from the loss output. We'll need to find the highest probability to return the correct result.

1. Add the `extractResult` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

```
C#  
  
private void extractResult()  
{  
    // A method to extract output (result and a probability) from the  
    "loss" output of the model  
    var collection = output.loss;  
    float maxProbability = 0;  
    string keyOfMax = "";  
  
    foreach (var dictionary in collection)  
    {  
        foreach (var key in dictionary.Keys)  
        {  
            if (dictionary[key] > maxProbability)  
            {  
                maxProbability = dictionary[key];  
                keyOfMax = key;  
            }  
        }  
    }  
    result = keyOfMax;  
    resultProbability = maxProbability;  
}
```

2. Add the `displayResult` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

```
C#  
  
// A method to display the results  
private async Task displayResult()  
{  
    displayOutput.Text = result.ToString();  
    displayProbability.Text = resultProbability.ToString();  
}
```

The result of the **Bind and Evaluate** and the **Extract and display the results** parts of the WinML code of our app will look as following.

```
C#
```

```

// A method to evaluate the model
private async Task evaluate()
{
    output = await modelGen.EvaluateAsync(input);
}

// A method to extract output (string and a probability) from the
"loss" output of the model
private void extractResult()
{
    var collection = output.loss;
    float maxProbability = 0;
    string keyOfMax = "";

    foreach (var dictionary in collection)
    {
        foreach (var key in dictionary.Keys)
        {
            if (dictionary[key] > maxProbability)
            {
                maxProbability = dictionary[key];
                keyOfMax = key;
            }
        }
    }
    result = keyOfMax;
    resultProbability = maxProbability;
}

// A method to display the results
private async Task displayResult()
{
    displayOutput.Text = result.ToString();
    displayProbability.Text = resultProbability.ToString();
}

```

That's it! You have successfully created the Windows machine learning app with a basic GUI to test our classification model. The next step is to launch the application and run it locally on your Windows device.

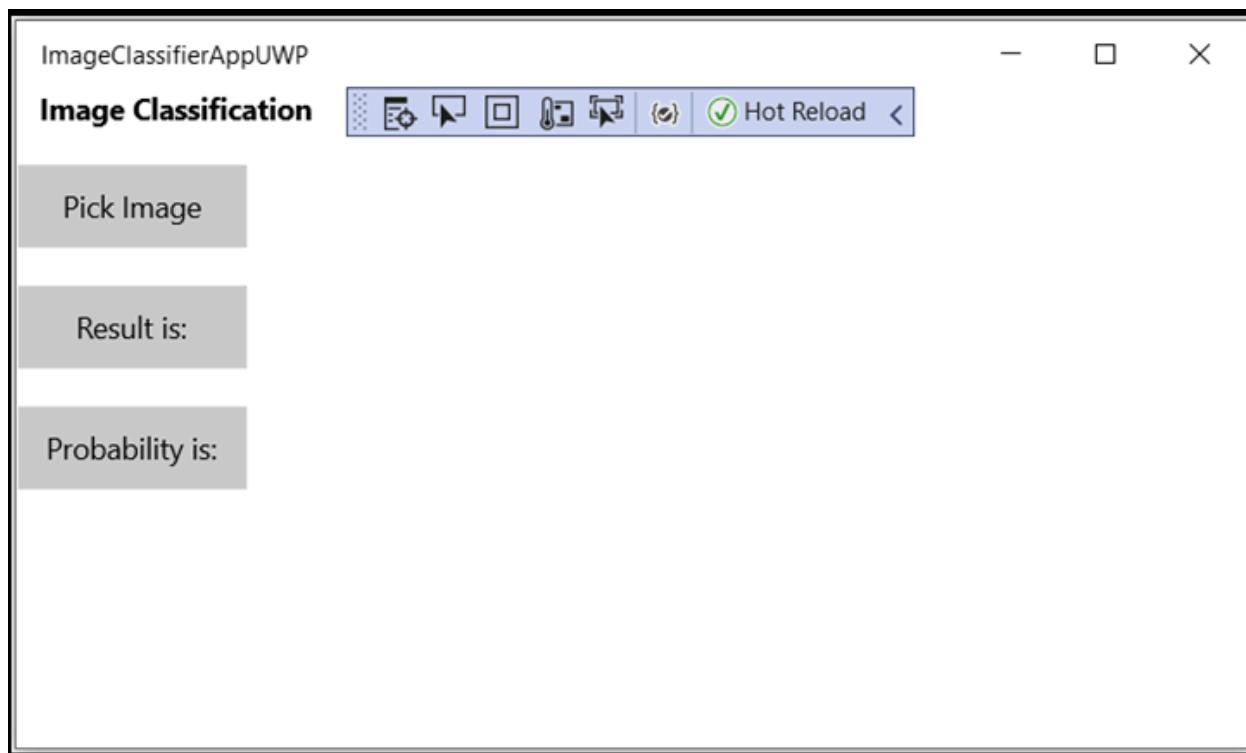
Launch the application

Once you completed the application interface, added the model, and generated the WinML code, you can test the application. Ensure the dropdown menus in the top toolbar are set to `Debug`. 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.

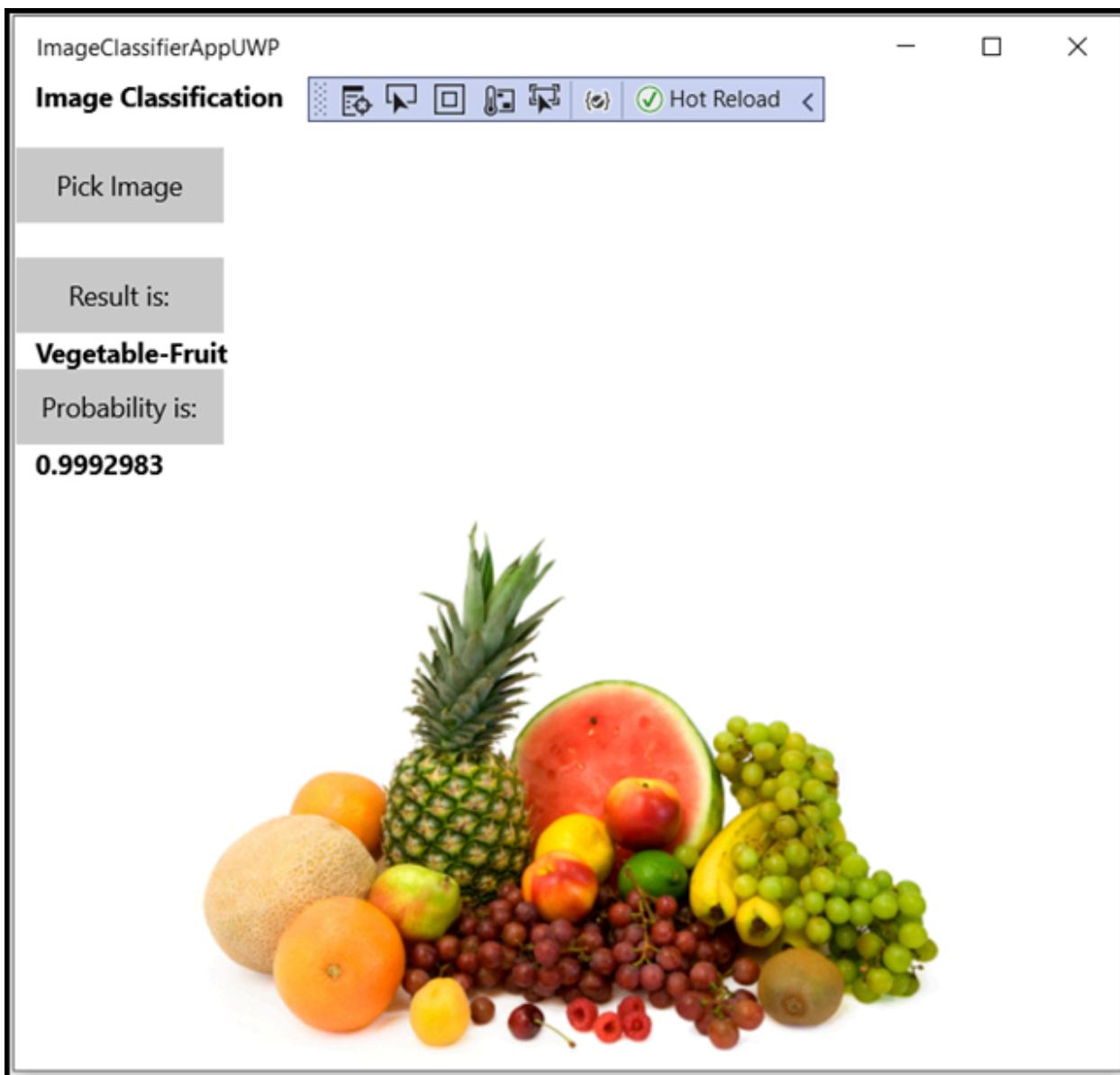
To test our app, you will use the below image of fruits. Let's see how our app classifies the content of the image.



1. Save this image on your local device to test the app. Change the image format to jpg if required. You can also add any other relevant image from your local device in a suitable format - .jpg, .png, .bmp, or .gif formats.
2. To run the project, press the `Start Debugging` button on the toolbar, or press `F5`.
3. When the application starts, press `Pick Image` and select the image from your local device.



The result will appear on the screen right away. As you can see, our WinML app successfully classified the image as fruits or vegetables, with a 99.9% confidence rating.



Summary

You've just made your first Windows Machine Learning app, from model creation to successful execution.

Additional Resources

To learn more about topics mentioned in this tutorial, visit the following resources:

- Windows ML tools: Learn more tools like the [Windows ML Dashboard](#), [WinMLRunner](#), and the [mglenn](#) Windows ML code generator.
- ONNX model: Learn more about the ONNX format.
- Windows ML performance and memory: Learn more how to manage app performance with Windows ML.
- Windows Machine Learning API reference: Learn more about three areas of Windows ML APIs.

Feedback

Was this page helpful?

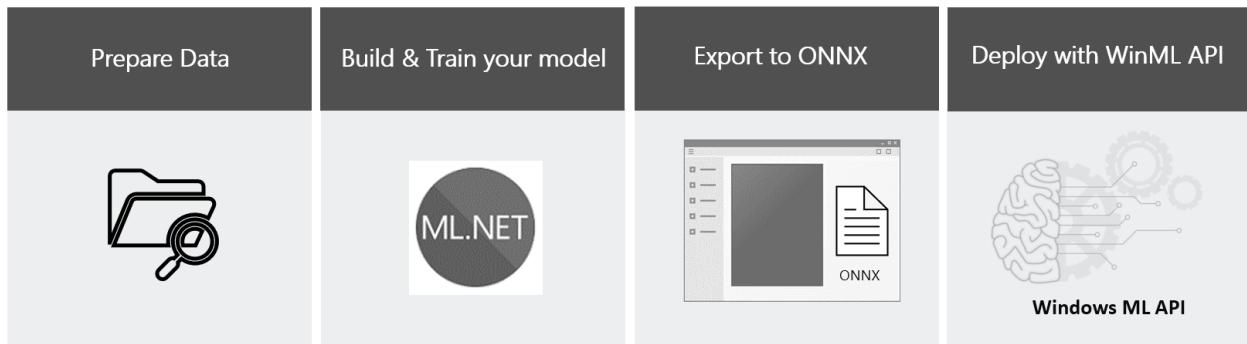
 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Image Classification with ML.NET and Windows Machine Learning

Article • 09/23/2022



This guide will show you how to train a neural network model to classify images of food using ML.NET Model Builder, export the model to ONNX format, and deploy the model in a Windows Machine Learning application running locally on a Windows device. No previous expertise in machine learning is required, and we'll guide you step by step through the process.

If you like to learn how to build and train a model with ML.NET Model Builder, you can proceed to [Train a Model](#).

If you have a model and want to learn how to create a WinML app from scratch, navigate to the complete [WinML app tutorial](#).

If you want to get the predefined solution for a WinML app, you can clone [the solution file](#) and test it right away.

Scenario

In this tutorial, we'll create a machine learning food classification application that runs on Windows devices. The model will be trained to recognize certain types of patterns to classify an image of food, and when given an image will return a classification tag and the associated percentage confidence value of that classification.

Prerequisites for model training

To build and train your model, you'll use the ML.NET Model Buider in Visual Studio.

- You'll need Visual Studio 2019 16.6.1 or later to use a ML.NET Model Builder. [You can get Visual Studio here](#).

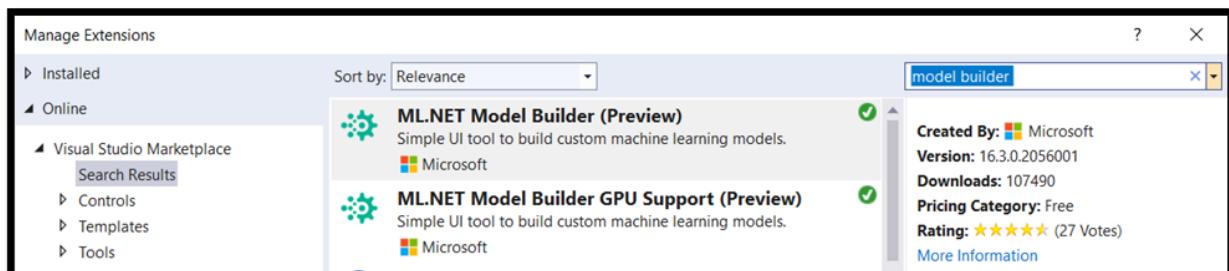
- You'll need an Azure account to train a model with ML.NET Model Builder within the Azure ML Workspace. If you're new to Azure, you may sign up for an [Azure free account](#).

Note

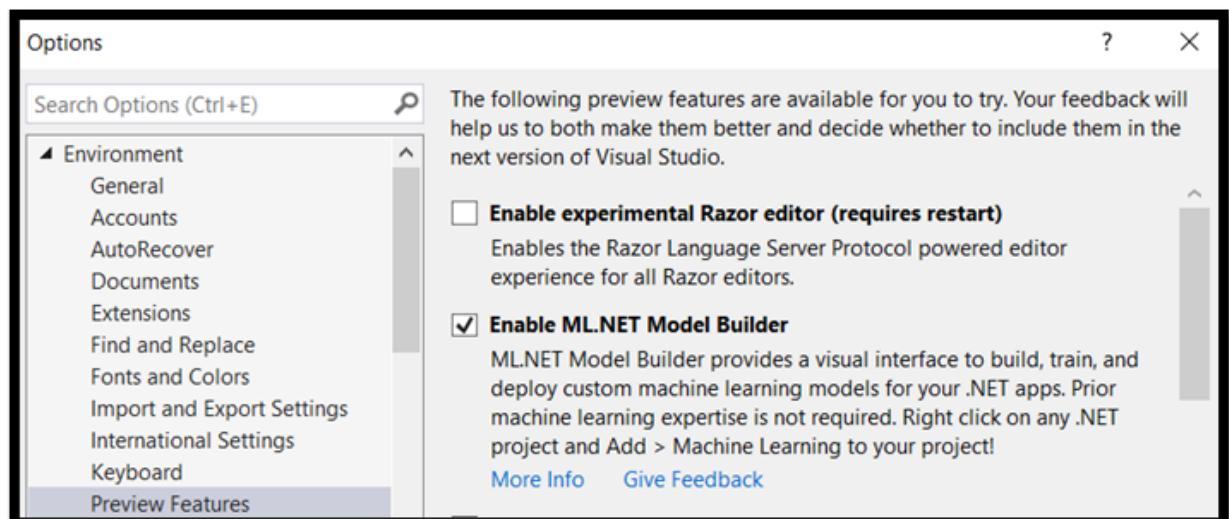
Interested in learning more about Azure sign-up options and Azure free accounts? Check out [Create an Azure account](#).

ML.NET Model Builder is an intuitive graphical Visual Studio extension, used to build, train, and deploy custom machine learning models. It uses automated machine learning (AutoML) to explore different machine learning algorithms and settings to help you find the one that best suits your scenario.

ML.NET Model Builder ships with Visual Studio version 16.6.1 or later, when you install one of the .NET workloads. Make sure to have the ML.NET Model Builder component checked in the installer when you download or modify Visual Studio. To check if your VS has the ML.NET Model Builder components, go to Extensions and select Manage Extensions. Type Model Builder in the search bar to review the extension results.



ML.NET Model Builder is currently a Preview feature. So, in order to use the tool, in Visual Studio you must go to Tools > Options > Environment > Preview Features and enable ML.NET Model Builder:



Note

Interested in learning more about ML.NET Model Builder and different scenarios it supports? Please review the [Model Builder documentation](#).

Prerequisites for Windows ML app deployment

To create and deploy a Widows ML app, you'll need the following:

- Windows 10 version 1809 (build 17763) or higher. You can check your build version number by running `winver` via the Run command (`Windows logo key + R`).
- Windows SDK for build 17763 or higher. [You can get the SDK here.](#)
- Visual Studio 2019 version 16.6.1 or later. [You can get Visual Studio here.](#)
- Windows ML Code Generator (mlgen) Visual Studio extension. Download for [VS 2019](#).
- If you decide to create a UWP app, you'll need to enable the Universal Windows Platform development workload in Visual Studio.
- You'll also need to [enable Developer Mode on your PC](#)

Note

Windows ML APIs are built into the latest versions of Windows 10 (1809 or higher) and Windows Server 2019. If your target platform is older versions of Windows, you can port your WinML app to the redistributable NuGet package (Windows 8.1 or higher).

Prepare the data

Machine learning models must be trained with existing data. In this guide, you will use a dataset of food images from Kaggle Open Datasets. This dataset is distributed under the public domain license.

Important

To use this dataset, you need to adhere to the term of using the Kaggle site and the liscence terms accompanying the Food-11 dataset itself. Microsoft makes no warranty or representation concerning the site or this dataset.

The dataset has three splits - evaluation, training and validation - and contains 16643 food images grouped in 11 major food categories. The images in the dataset of each category of food are placed in a separate folder, which makes the model training process more convenient.

Download the dataset [here](#). Please note that the dataset is around 1 gb in size, and you may be asked to create an account on the Kaggle website to download the data.



If you want, you're welcome to use any other dataset of relevant images. As a minimum, we recommend you use at least 30 images per tag in the initial training set. You'll also want to collect a few extra images to test your model once it is trained.

Additionally, make sure all your training images meet the following criteria:

- .jpg, .png, .bmp, or .gif format.
- no greater than 6MB in size (4MB for prediction images).
- no less than 256 pixels on the shortest edge; any images shorter than this will be automatically scaled up by the Custom Vision Service.

Next Steps

Now that you've gotten your prerequisites sorted out and have prepared your dataset, you can proceed to creation of your WinML model. [In the next part](#), you'll use the ML.NET Model Builder to create and train your classification model.

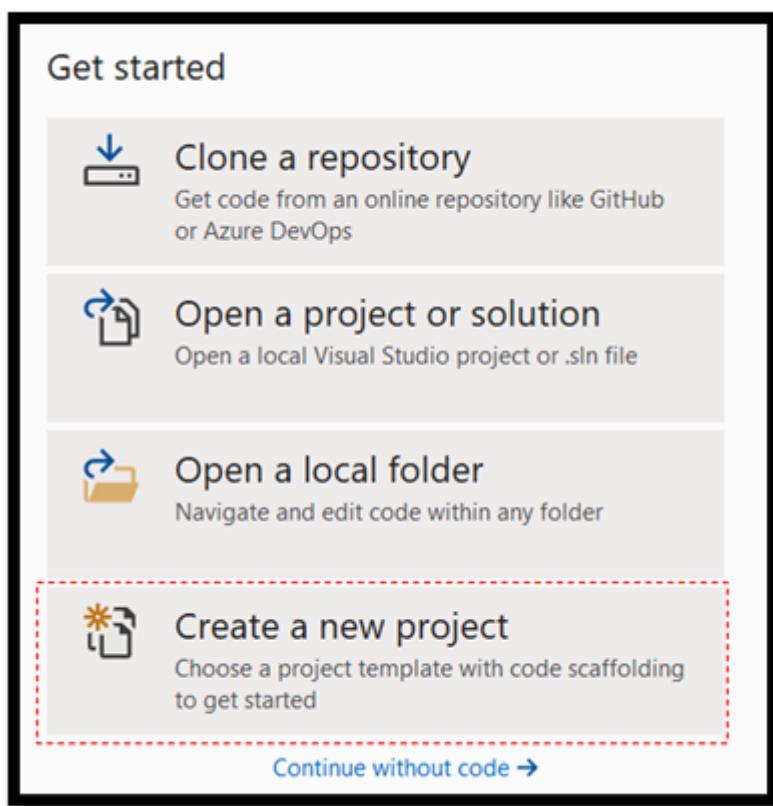
Train your model with ML.NET

Article • 06/22/2022

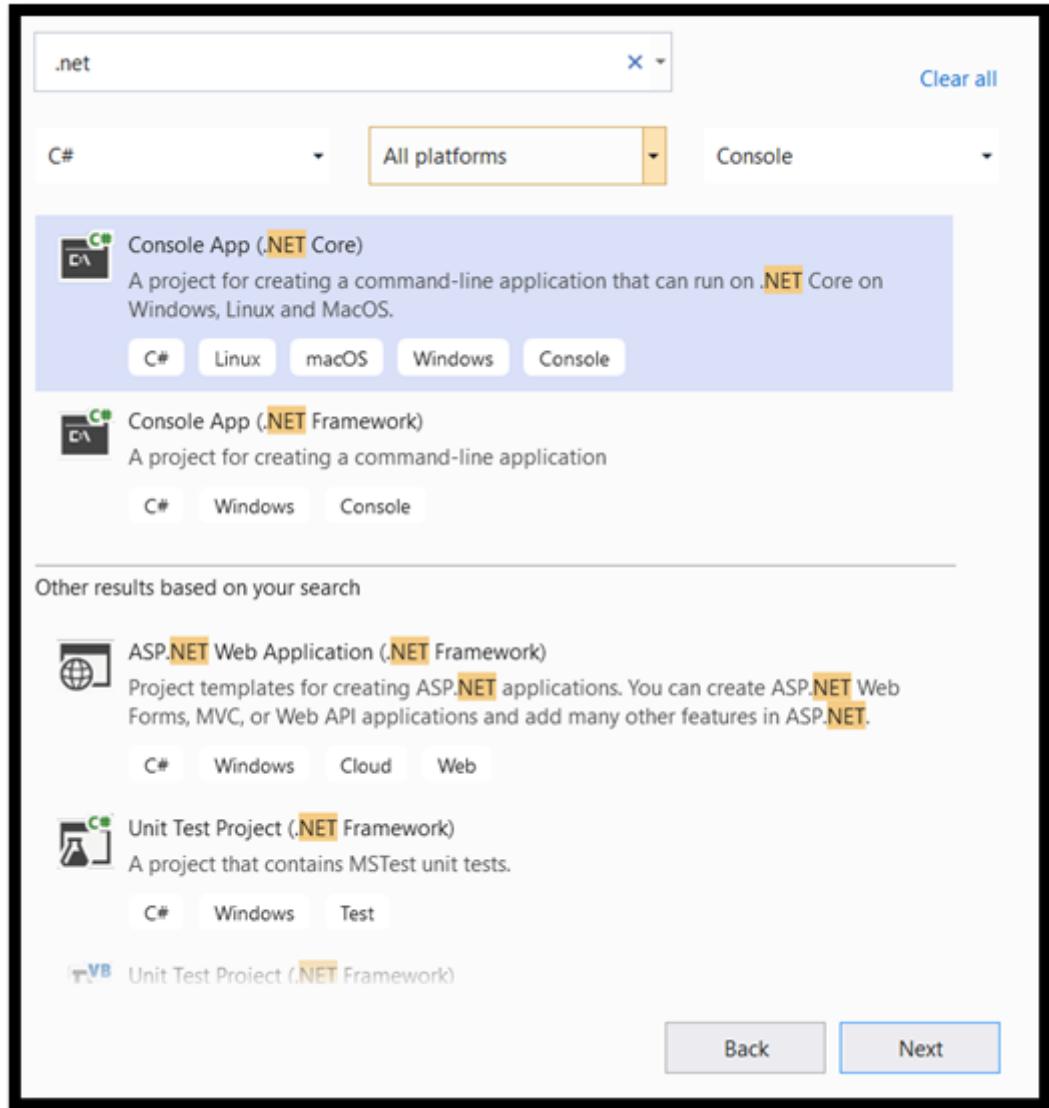
In the [previous stage of this tutorial](#), we discussed the prerequisites of creating your own Windows Machine Learning model and app, and downloaded an image set to use. In this stage, we'll learn how to use the ML.NET Model Builder to turn our image set into an image classification model.

Create your project

1. Open Visual Studio and choose "create a new project".



2. In the search bar, type .NET, select C# as your language and console as your platform and then choose the C# Console App (.NET Core) project template.



3. In the configuration window:

- Name your project. Here, we've called it **MLNETTraining**.
- Choose the location for your project.
- Make sure `Place solution and project in the same directory` is unchecked.
- Press `create` to create your project.

Configure your new project

Console App (.NET Core) C# Linux macOS Windows Console

Project name

Location



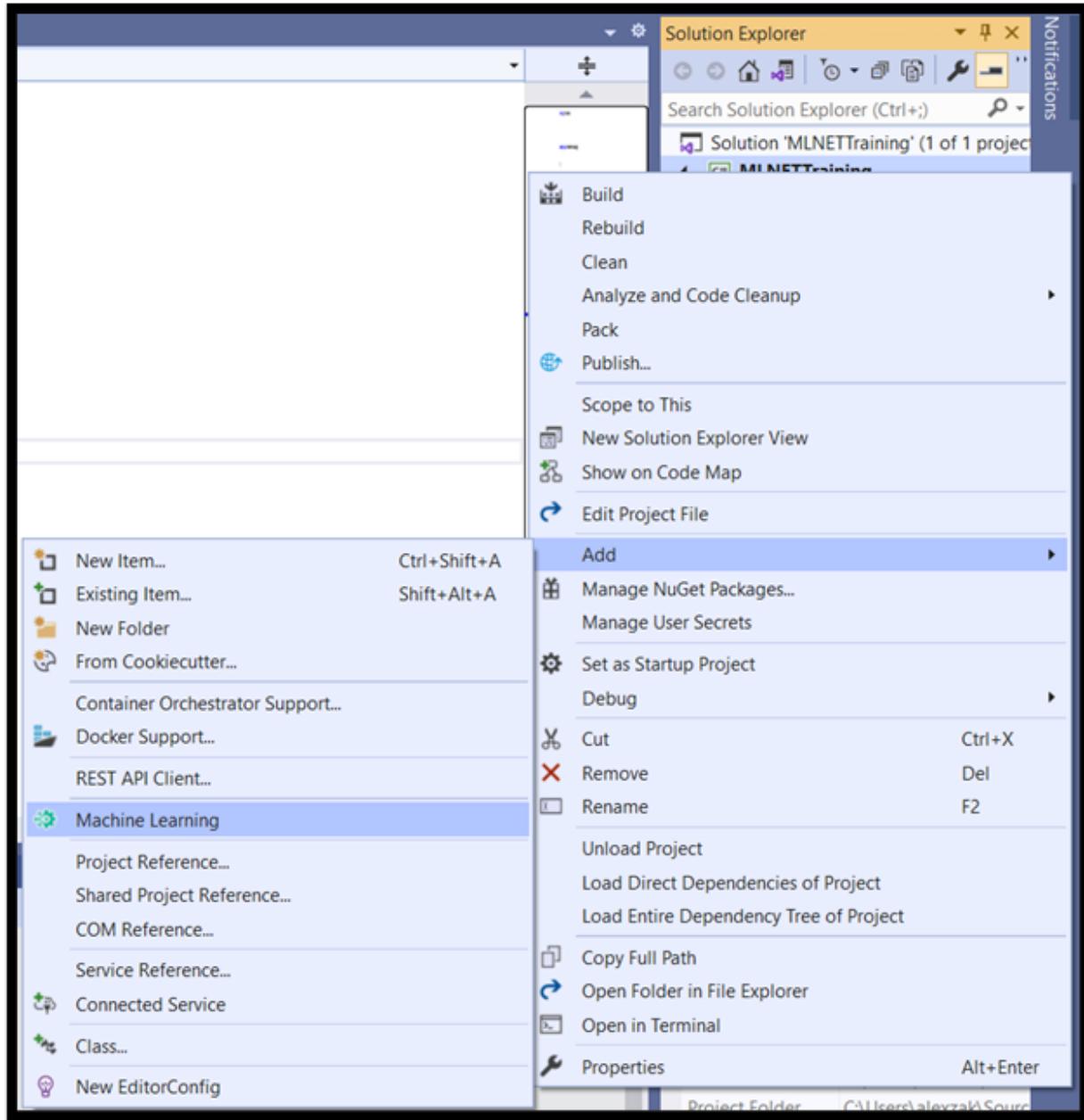
Solution name i

Place solution and project in the same directory

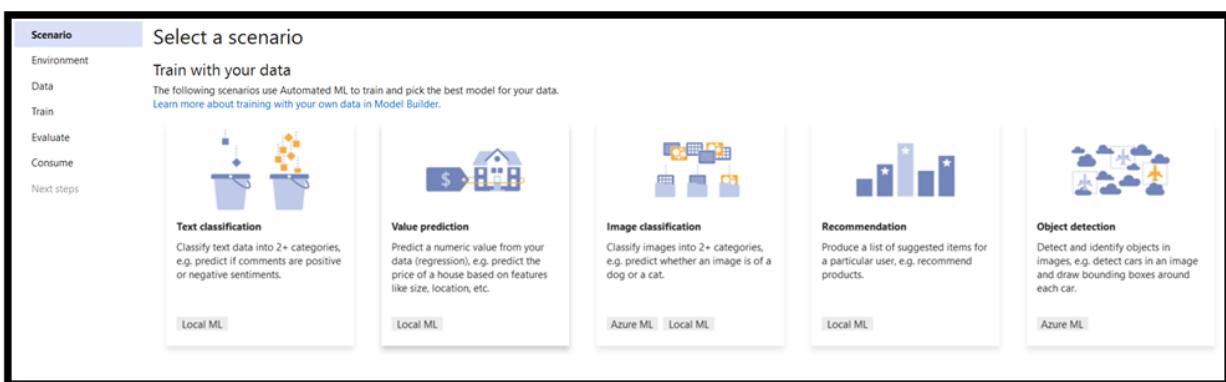
Set up Model Builder

Now, you will add Model Builder to our project.

1. Right-click on the **MLNETTraining** project in Solution Explorer and select **Add > Machine Learning**.



This way, you open ML.NET Model Builder in a new docked tool window in Visual Studio. Model Builder will guide you through the process of building a machine learning model.



The first step is to choose the relevant scenario. Not all scenarios support ONNX format.

If the training environment is Azure cloud, the generated models are in ONNX format and can be easily consumed by Windows ML app without conversion. However, if you

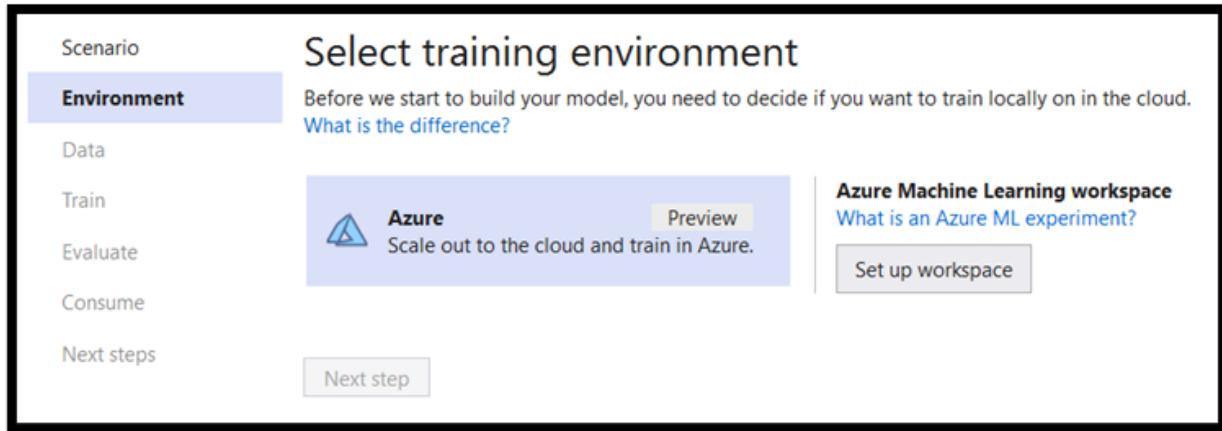
decide to train your machine learning model locally on your machine, the generated model will be in ML.NET format.

- Local CPU training is supported for all scenarios except Object Detection.
- Local GPU training is supported for Image Classification.
- Azure training is supported for Image Classification and Object Detection.

In this tutorial, you will train image classification model in Azure training environment. The output model will be in ONNX format. Azure account is required to complete the training.

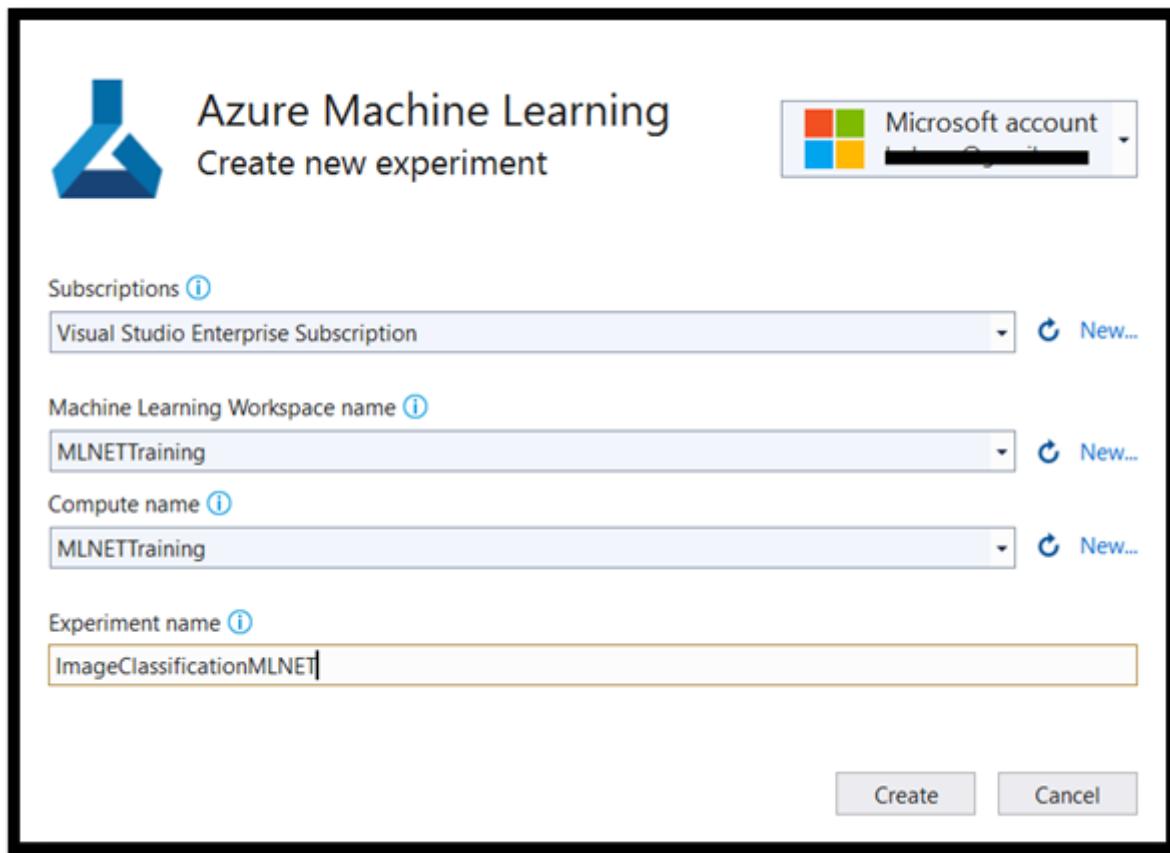
2. Choose the Image Classification Scenario.

3. Select Set up workspace to set up your Azure training environment.



On the upper right corner, sign into the account associated with your Azure subscription. In the menu below:

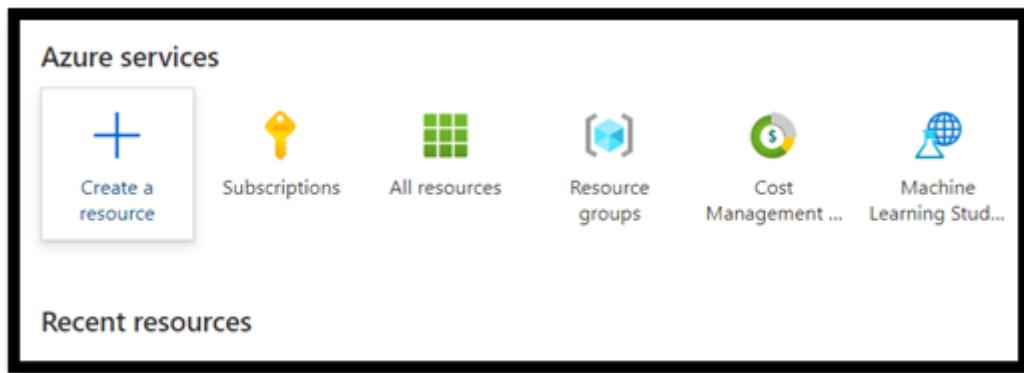
- Choose the relevant subscription.
- Select and create a new Machine Learning Workspace.
- Select or create a new Compute resource.
- Give the name to your workspace – ImageClassificationMLNET.



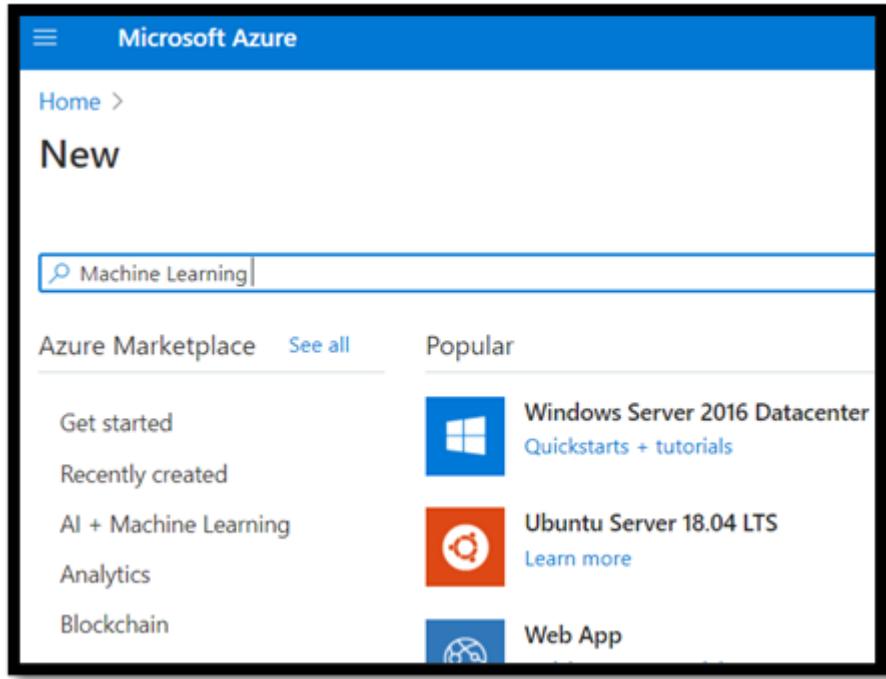
ⓘ Important

If you aren't able to create the Machine Learning workspace from Model Builder, follow these steps to create a workspace manually from your Azure Portal. Otherwise, you can skip to step 4.

At your Azure account, select Create a resource:



In the search bar, look for a Machine Learning.



Press Create to create a new Machine Learning workspace.

A screenshot of the Microsoft Azure Marketplace page for 'Machine Learning'. The page shows the following details:

- Icon**: A blue 3D geometric icon.
- Name**: Machine Learning
- Provider**: Microsoft
- Add to Favorites**: A blue 'Add to Favorites' button with a heart icon.
- Create**: A blue 'Create' button.
- Overview**: The selected tab, indicated by a blue underline.
- Plans**
- Usage Information + Support**
- Reviews**

To create a new workspace, you will need to provide your subscription name, select or create a new resource group, give a name to the workspace and define all the required parameters such as region, storage account, etc.

Microsoft Azure

Home > New > Machine Learning >

Machine Learning

Create a machine learning workspace

Basics Networking Advanced Tags Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

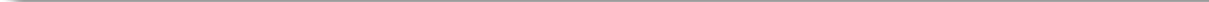
Subscription * ⓘ Visual Studio Enterprise Subscription

Resource group * ⓘ MLNETTraining
Create new

Workspace details

Specify the name and region for the workspace.

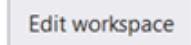
Workspace name * ⓘ MLNETTraining



Once you've established your workspace and created a new training environment in ML.NET, you can move to the next step.

Select training environment

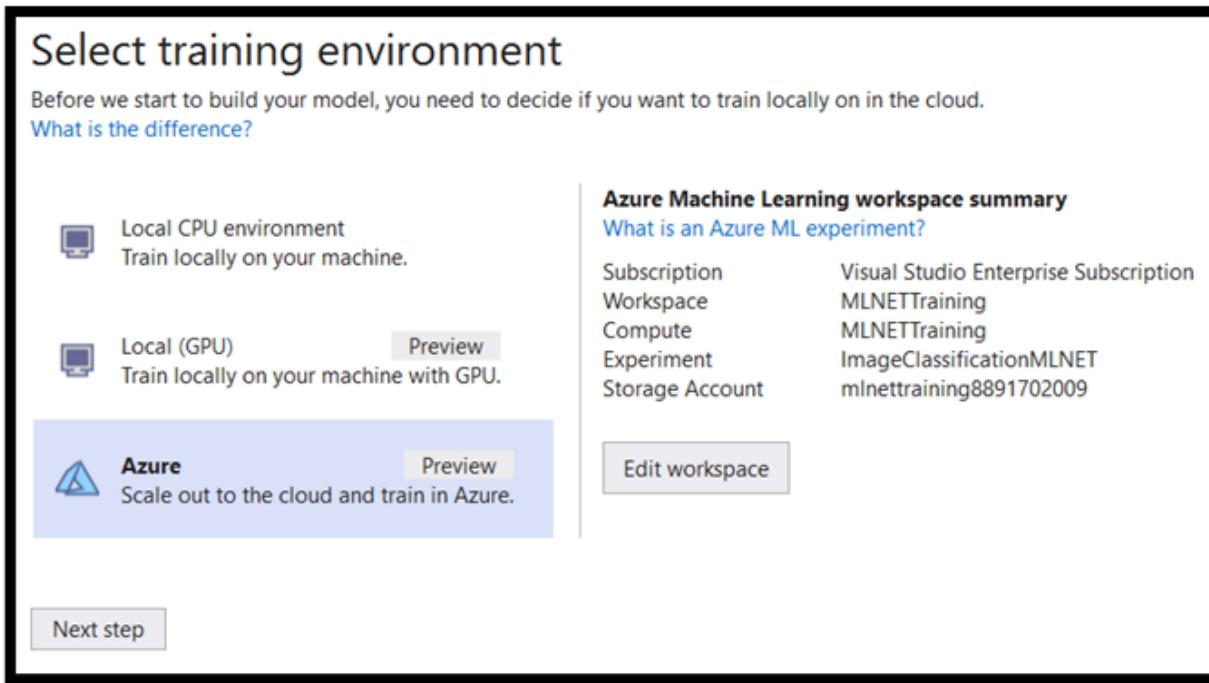
Before we start to build your model, you need to decide if you want to train locally or in the cloud.
[What is the difference?](#)

 Local CPU environment Train locally on your machine.	 Azure Preview Scale out to the cloud and train in Azure.
 Local (GPU) Preview Train locally on your machine with GPU.	 Edit workspace

Azure Machine Learning workspace summary
[What is an Azure ML experiment?](#)

Subscription	Visual Studio Enterprise Subscription
Workspace	MLNETTraining
Compute	MLNETTraining
Experiment	ImageClassificationMLNET
Storage Account	mynettraining8891702009

[Next step](#)



Wait until the deployment of Machine Learning Services is complete.

The next step is to add the data to Model Builder.

4. Navigate to the location of the image dataset and select the training folder with the relevant food categories. In this tutorial you will train the model to recognize desert, soup and fruit, so you need only these categories in our dataset folder.

Scenario

Environment

Data

Train

Evaluate

Consume

Next steps

Add data

In order to build a model, you must add image data.
[How do I get sample datasets and learn more?](#)

Input

Select the folder which contains all your images. This folder should organize your photos into separate labeled sub-folders.

Select a folder:

Supported file formats: .png, .jpg, .jpeg, .gif.

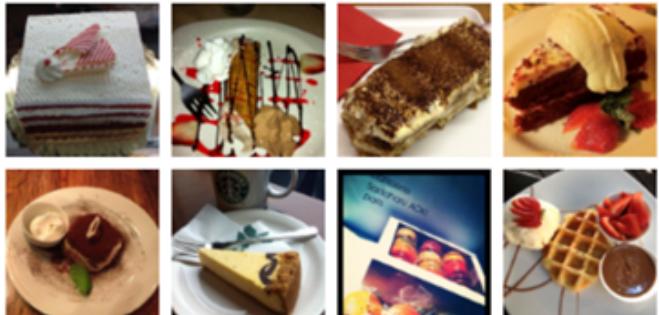
Example folder structure:

```
Images
  └── Label 1
    ├── Image 1
    └── Image 2
  └── Label 2
    └── Label 3
```

Total images 3709. Showing 8/1500.

training:

- Dessert (1500)
- Soup (1500)
- Vegetable-... (709)



[Next step](#)

Now, you are ready to move to the training part!

Train your Model

Model Builder evaluates many models with varying algorithms and settings to give you the best performing model.

1. Select next and then Start Training to start the training process. The ML.NET model builder will start by uploading data to Azure, prepare the workspace and then initiate the training process.

Scenario

Environment

Data

Train

Evaluate

Consume

Next steps

Train your model in Azure

Model Builder automatically sets the training time based on the size of your dataset.

Training setup summary 

[Cancel training](#)  Training...

[Monitor current run in Azure portal.](#)

[Next step](#)

Once training is done, you will see a summary of the training results.

The screenshot shows the Model Builder interface with a sidebar on the left containing links: Scenario, Environment, Data, Train, Evaluate, Consume, and Next steps. The 'Train' link is highlighted with a blue background. The main content area has a title 'Train your model in Azure'. Below it, a message says 'Model Builder automatically sets the training time based on the size of your dataset.' A 'Training setup summary' link is shown with a dropdown arrow. In the center, there are two buttons: 'Train again' and 'Training complete' with a green checkmark. To the right of these buttons, a 'Training results' section displays the following metrics:

- Best accuracy: 95.42%
- Best model: DNN + ResNet50
- Training time: 1882 seconds
- Models explored (total): 1

A blue link 'Monitor current run in Azure portal.' is present. At the bottom, a 'Next step' button is visible.

Best accuracy - shows you the accuracy of the best model that Model Builder found. Higher accuracy means the model predicted more correctly on test data. In our case, the model can predict the correct result with 95.42% of confidence.

Evaluate the results

1. Move the next step to evaluate the training results.
2. Select the image from the evaluation folder of the dataset and explore the prediction.

Scenario

Environment

Data

Train

Evaluate

Consume

Next steps

Evaluate

Results of training for your model can be found below.
[How do I understand my model performance?](#)

Best model:

Accuracy: 95.42%

Model: DNN + ResNet50

Try your model



[Try another image](#)

Results

Vegetable-Fruit	100%
Dessert	< 1%
Soup	< 1%

[Next step](#)

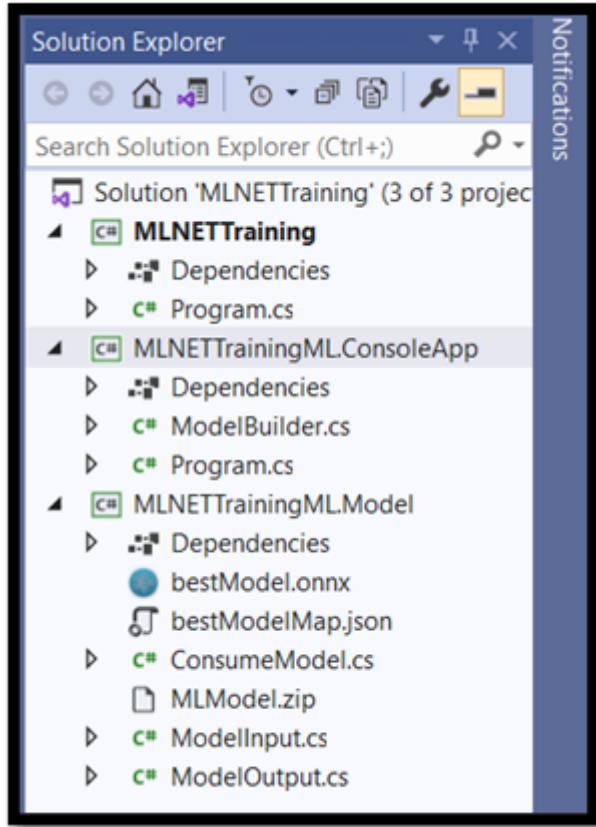
Add the model to the solution

ML.NET Model Builder can automatically add both the machine learning model and the projects for training and consuming the model to your solution.

1. Navigate to the consume part of the training process and sell Add to solution. This will add the generated model to your solution folder.

Scenario	<h2>Consume the model</h2>
Environment	Let's add the model and necessary projects and references to the solution. Once added, open MLNETTrainingML.ConsoleApp to see how to consume your model. You can then copy the code from this generated console app to the app where you want to consume your model and make predictions.
Data	
Train	Add to solution
Evaluate	
Consume	Next step
Next steps	

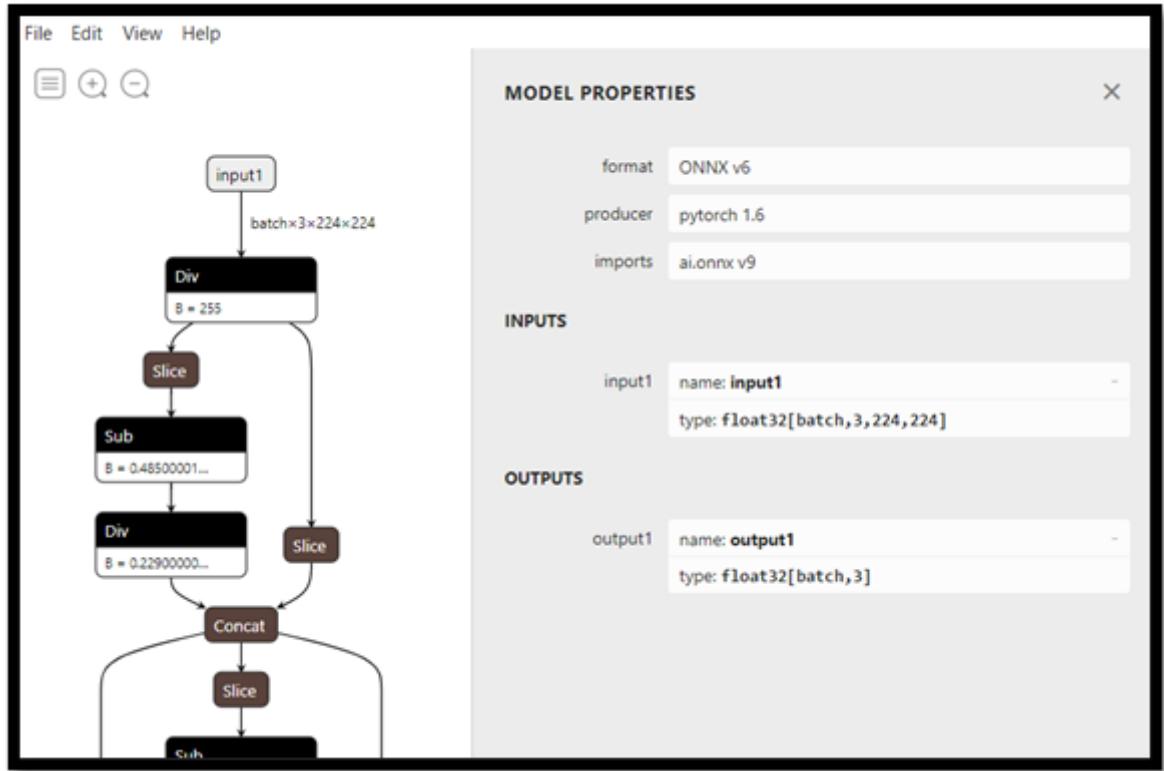
In the Solution Explorer, you should see the code files that were generated by Model Builder, including the model – bestModel.onnx in ONNX format.



The model trained in Azure cloud environment, so the generated model is in ONNX format.

Explore your model

1. Right-click on the bestModel.onnx will and select Open Containing Folder.
2. Open your model file with Netron program.
3. Press on the input1 node to open the model properties.



As you can see, the model requires 32-bit float tensor (multi-dimensional array) object as an input and returns Tensor float as an output. The way the model was built, it does not return the string value of a predicted label but an array of three numbers, each represents the relevant label of the food type. You will need to extract these values to show the correct prediction with Windows ML app.

Label 1	Label 2	Label 3
0	1	2
dessert	soup	Vegetable-Fruit

Next Steps

Now that you've trained your Machine Learning model, you're ready to [deploy it in a UWP app with Windows Machine Learning](#)

Deploy your ML.NET model in a Windows app with the Windows Machine Learning APIs

Article • 12/30/2021

In the previous part of this tutorial, you learned how to build and export a ML.NET model in ONNX format. Now that you have that model, you can embed it into a Windows application and run it locally on a device by calling WinML APIs.

Once we're done, you'll have a working Image classifier WinML UWP app (C#).

About the sample app

Using our model, we'll create an app that can classify images of food. It allows you to select an image from your local device and process it by a locally stored classification ONNX model you built and trained in the previous part. The tags returned are displayed next to the image, as well as the the confidence probability of the classification.

If you've been following this tutorial so far, you should already have the necessary prerequisites for app development in place. If you need a refresher, see the first part of this tutorial.

Note

If you prefer to download the complete sample code, you can clone the solution file [↗](#). Clone the repository, navigate to this sample, then open the `classifierMLNETModel1.sln` file with Visual Studio. Then, you can skip to the [Launch the application](#Launch the application) step.

Create a WinML UWP (C#)

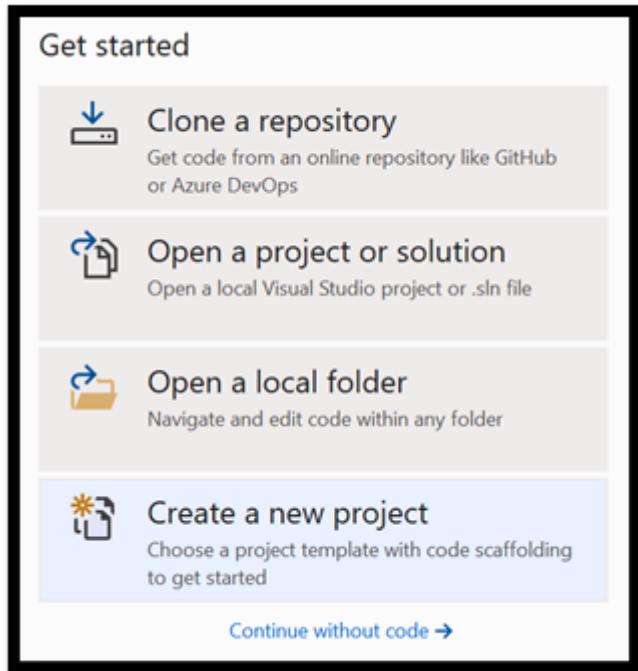
Below, we'll show you how to create your app and WinML code from scratch. You'll learn how to:

- Load a machine learning model.
- Load an image in the required format.
- Bind the model's inputs and outputs.
- Evaluate the model and display meaningful results.

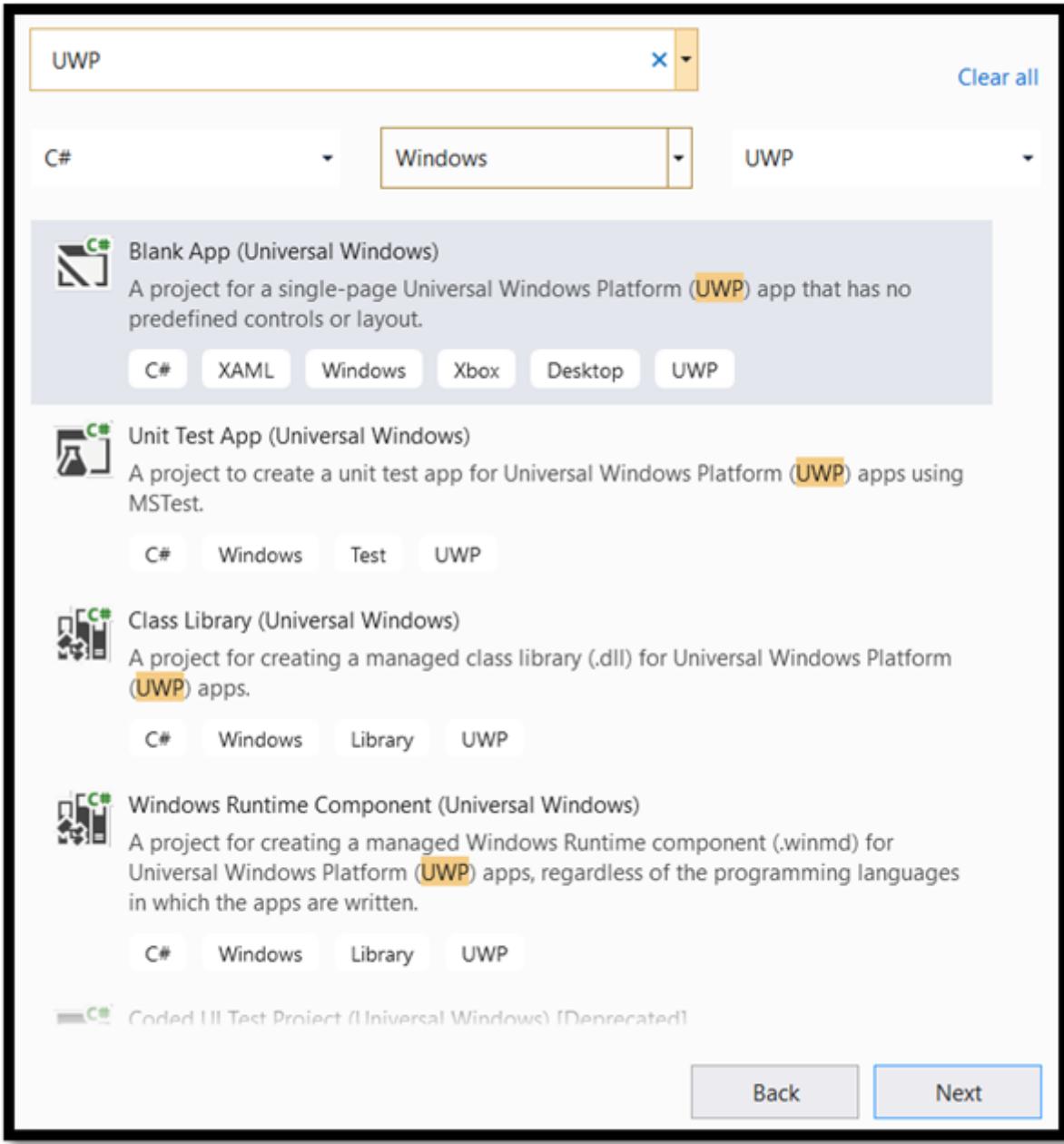
You'll also use basic XAML to create a simple GUI, so you can test the image classifier.

Create the app

1. Open Visual Studio and choose `create a new project`.



2. In the search bar, type `UWP` and then select `Blank APP (Universal Windows)`. This opens a new C# project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout. Select `Next` to open a configuration window for the project.



3. In the configuration window:

- Choose a name for your project. Here, we use **classifierMLNETModel**.
- Choose the location of your project.
- If you're using VS 2019, ensure **Place solution and project in the same directory** is unchecked.
- If you're using VS 2017, ensure **Create directory for solution** is checked.

Configure your new project

Console App (.NET Core) C# Linux macOS Windows Console

Project name

Location

Solution name i

Place solution and project in the same directory

Press **create** to create your project. The minimum target version window may pop up. Be sure your minimum version is set to **Windows 10 build 17763** or later.

To create an app and deploy a model with a WinML app, you'll need the following:

4. After the project was created, navigate to the project folder, open the assets folder [...\classifierMLNETModel\Assets], and copy your `bestModel.onnx` file to this location.

Explore project solution

Let's explore your project solution.

Visual Studio automatically created several cs-code files inside the Solution Explorer. `MainPage.xaml` contains the XAML code for your GUI, and `MainPage.xaml.cs` contains your application code. If you've created a UWP app before, these files should be very familiar to you.

Create the Application GUI

First, let's create a simple GUI for your app.

1. Double-click on the `MainPage.xaml` file. In your blank app, the XAML template for your app's GUI is empty, so we'll need to add some UI features.
2. Replace the code of `MainPage.xaml` with the following.

XAML

```
<Page
    x:Class="classifierMLNETModel.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:classifierMLNETModel"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <StackPanel Margin="1,0,-1,0">
            <TextBlock x:Name="Menu"
                FontWeight="Bold"
                TextWrapping="Wrap"
                Margin="10,0,0,0"
                Text="Image Classification"/>
            <TextBlock Name="space" />
            <Button Name="recognizeButton"
                Content="Pick Image"
                Click="OpenFileButton_Click"
                Width="110"
                Height="40"
                IsEnabled="True"
                HorizontalAlignment="Left"/>
            <TextBlock Name="space3" />
            <Button Name="Output"
                Content="Result is:"
                Width="110"
                Height="40"
                IsEnabled="True"
                HorizontalAlignment="Left"
                VerticalAlignment="Top">
                </Button>
            <!--Dispaly the Result-->
            <TextBlock Name="displayOutput"
                FontWeight="Bold"
                TextWrapping="Wrap"
                Margin="25,0,0,0"
                Text="" Width="1471" />
            <TextBlock Name="space2" />
            <!--Image preview -->
            <Image Name="UIPreviewImage" Stretch="Uniform" MaxWidth="300"
MaxHeight="300"/>
        </StackPanel>
    </Grid>
</Page>
```

Add the model to the project using Windows Machine Learning Code Generator

Windows Machine Learning Code Generator, or `mlgen`, is a Visual Studio extension to help you get started using WinML APIs on UWP apps. It generates template code when you add a trained ONNX file into the UWP project.

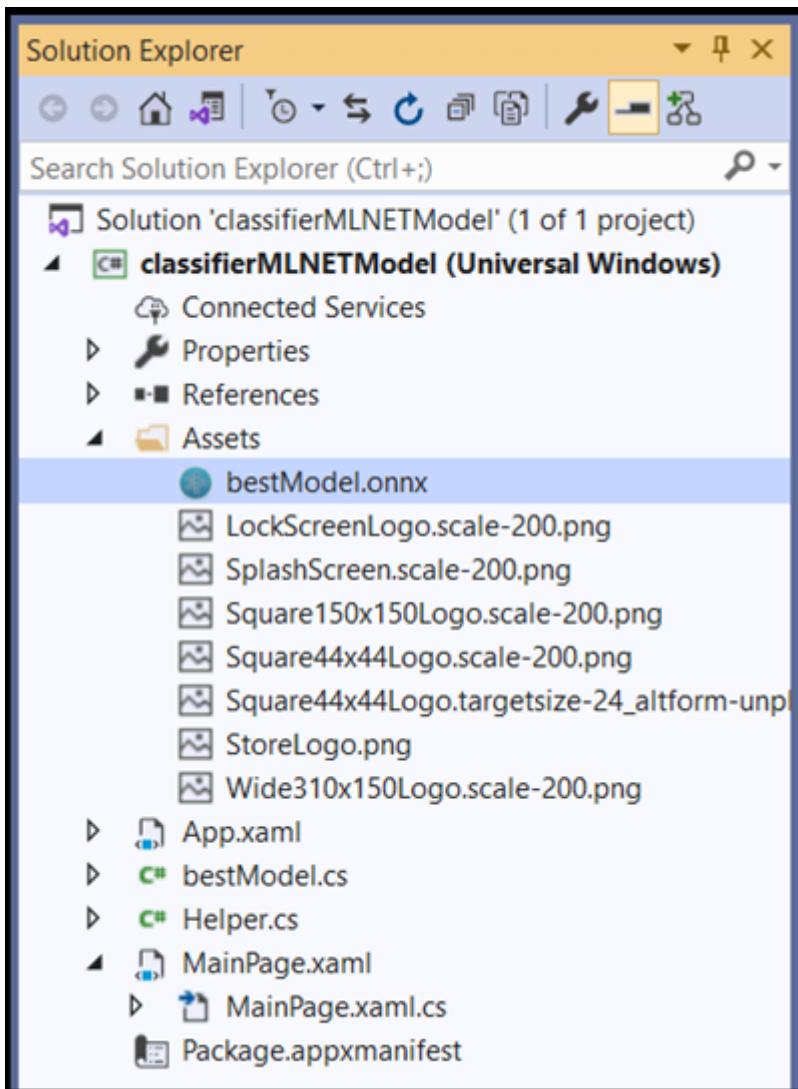
Windows Machine Learning's code generator `mlgen` creates an interface (for C#, C++/WinRT, and C++/CX) with wrapper classes that call the Windows ML API for you. This allows you to easily load, bind, and evaluate a model in your project. We'll use it in this tutorial to handle many of those functions for us.

Code generator is available for Visual Studio 2017 and later. Please be aware that 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. If you've been following this tutorial from the introduction, you will have already handled this, but if not, you should download for [VS 2019](#) or for [VS 2017](#).

ⓘ Note

To learn more about `mlgen`, please see the [mlgen documentation](#)

1. If you haven't already, install `mlgen`.
2. Right-click on the `Assets` folder in the Solution Explorer in Visual studio, and select `Add > Existing Item`.
3. Navigate to the assets folder inside `ImageClassifierAppUWP` `[... \ImageClassifierAppUWP\Assets]`, find the ONNX model you previously copied there, and select `add`.
4. After you added an ONNX model (name: "classifier") to the assets folder in solution explorer in VS, the project should now have two new files:
 - `bestModel.onnx` - this is your model in ONNX format.
 - `bestModel.cs` – automatically generated WinML code file.



5. To make sure the model builds when you compile our application, select the `bestModel.onnx` file and choose `Properties`. For `Build Action`, select `Content`.

Now, let's explore the newly generated code in the `bestModel.cs` file.

The generated code includes three classes:

- `bestModelModel`: This class includes two methods for model instantiation and model evaluation. It will help us to create the machine learning model representation, create a session on the system default device, bind the specific inputs and outputs to the model, and evaluate the model asynchronously.
- `bestModelInput`: This class initializes the input types that the model expects. The model input depends on the model requirements for input data.
- `bestModelOutput`: This class initializes the types that the model will output. The model output depends on how it is defined by the model.

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

Tensor conversion

To make it easier dealing with tensorization, change the input `TensorFloat` class to `ImageFeatureValue`.

1. Make the following changes in the `bestModel.cs` file:

The code:

C#

```
public sealed class bestModelInput
{
    public TensorFloat input; // shape(-1,3,32,32)
}
```

Will become:

C#

```
public sealed class bestModelInput
{
    public ImageFeatureValue input; // shape(-1,3,32,32)
}
```

Load the model & inputs

Load the model

1. Double-click on the `MainPage.xaml.cs` code file to open the application code.
2. Replace the “using” statements with the following, to get an access to all the APIs that you'll need.

C#

```
// Specify all the using statements which give us the access to all the APIs
// that you'll need
using System;
using System.Threading.Tasks;
using Windows.AI.MachineLearning;
using Windows.Graphics.Imaging;
using Windows.Media;
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.Storage.Streams;
using Windows.UI.Xaml;
```

```
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Imaging;
```

3. Add the following variable declarations after the using statements inside your `MainPage` class, under the namespace `classifierMLNETModel`.

C#

```
// All the required fields declaration
private bestModelModel modelGen;
private bestModelInput image = new bestModelInput();
private bestModelOutput results;
private StorageFile selectedStorageFile;
private string label = "";
private float probability = 0;
private Helper helper = new Helper();

public enum Labels
{
    desert,
    soup,
    vegetable_fruit,
}
```

Now, you'll implement the `LoadModel` method. The method will access the ONNX model and store it in memory. Then, you'll use the `CreateFromStreamAsync` method to instantiate the model as a `LearningModel` object. The `LearningModel` class represents a trained machine learning model. Once instantiated, the `LearningModel` is the initial object you use to interact with Windows ML.

To load the model, you can use several static methods in the `LearningModel` class. In this case, you'll use the `CreateFromStreamAsync` method.

The `CreateFromStreamAsync` method was automatically created with `mlgen`, so you don't need to implement this method. You can review this method by double clicking on the `bestModel.cs` file generated file by `mlgen`.

To learn more about `LearningModel` class, please review the [LearningModel Class documentation](#).

To learn more about additional ways of loading the model, please review the [Load a model documentation](#)

4. Let's define the main method.

C#

```
// The main page to initialize and execute the model.
public MainPage()
{
    this.InitializeComponent();
    loadModel();
}
```

5. Add the implementation of the `loadModel` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
private async Task loadModel()
{
    // Get an access the ONNX model and save it in memory.
    StorageFile modelFile = await
    StorageFile.GetFileFromApplicationUriAsync(new Uri($"ms-
    appx:///Assets/bestModel.onnx"));
    // Instantiate the model.
    modelGen = await bestModelModel.CreateFromStreamAsync(modelFile);
}
```

Load the Image

1. We need to define a click event to initiate the sequence of four method calls for model execution – conversion, binding and evaluation, output extraction and display the results. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
// Waiting for a click event to select a file
private async void OpenFileButton_Click(object sender,
RoutedEventArgs e)
{
    if (!await getImage())
    {
        return;
    }
    // After the click event happened and an input selected, begin
    the model execution.
    // Bind the model input
    await imageBind();
    // Model evaluation
    await evaluate();
    // Extract the results
    extractResult();
    // Display the results
```

```
        await displayResult();
    }
```

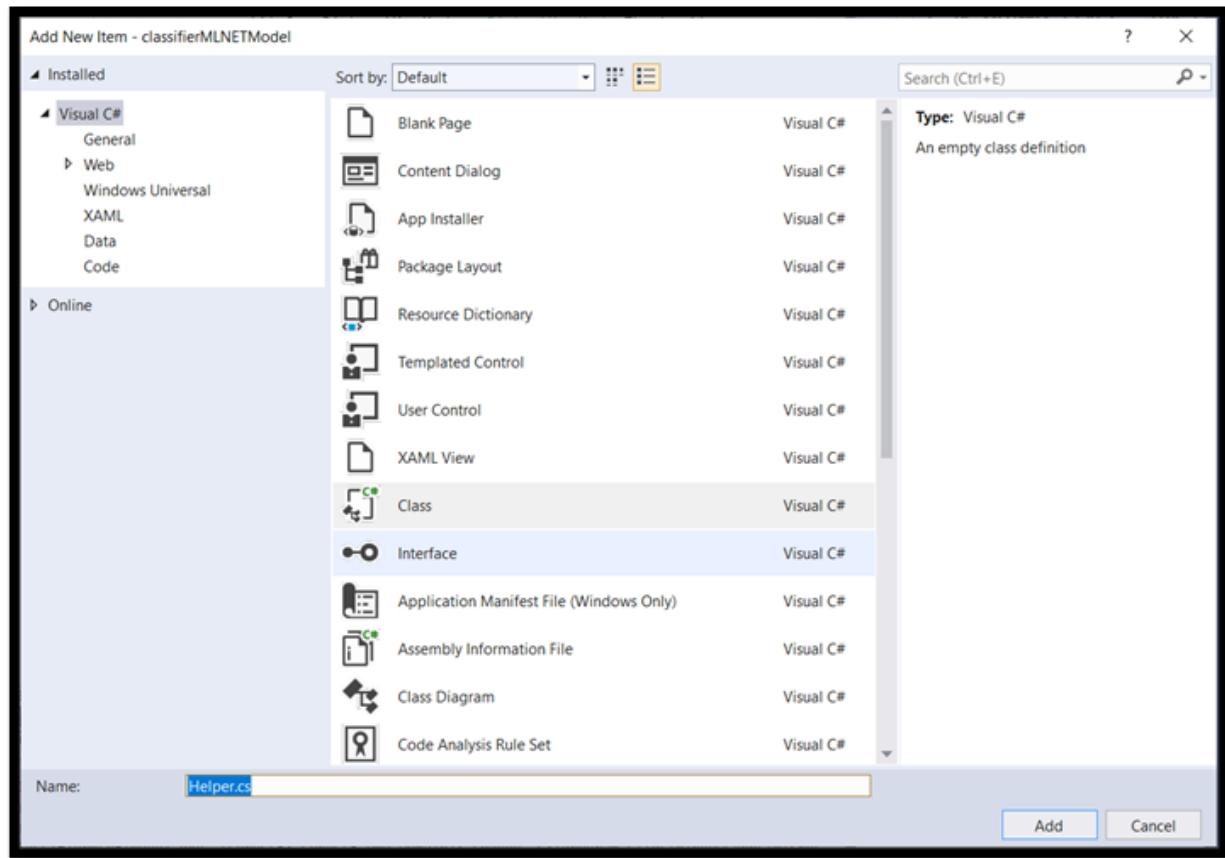
2. Now, you'll implement the `getImage()` method. This method will select an input image file and save it in memory. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
// A method to select an input image file
private async Task<bool> getImage()
{
    try
    {
        // Trigger file picker to select an image file
        FileOpenPicker fileOpenPicker = new FileOpenPicker();
        fileOpenPicker.SuggestedStartLocation =
PickerLocationId.PicturesLibrary;
        fileOpenPicker.FileTypeFilter.Add(".jpg");
        fileOpenPicker.FileTypeFilter.Add(".png");
        fileOpenPicker.ViewMode = PickerViewMode.Thumbnail;
        selectedStorageFile = await
fileOpenPicker.PickSingleFileAsync();
        if (selectedStorageFile == null)
        {
            return false;
        }
    }
    catch (Exception)
    {
        return false;
    }
    return true;
}
```

Next, you will implement an image `Bind()` method to get the representation of the file in the bitmap BGRA8 format. But first, you'll create a helper class to resize the image.

3. To create a helper file, right-click on the solution name (`ClassifierPyTorch`), then choose `Add a new item`. In the open window, select `class` and give it a name. Here, we're calling it `Helper`.



4. A new class file will appear within your project. Open this class, and add the following code:

C#

```
using System;
using System.Threading.Tasks;
using Windows.Graphics.Imaging;
using Windows.Media;

namespace classifierPyTorch
{
    public class Helper
    {
        private const int SIZE = 32;
        VideoFrame cropped_vf = null;

        public async Task<VideoFrame>
CropAndDisplayInputImageAsync(VideoFrame inputVideoFrame)
        {
            bool useDX = inputVideoFrame.SoftwareBitmap == null;

            BitmapBounds cropBounds = new BitmapBounds();
            uint h = SIZE;
            uint w = SIZE;
            var frameHeight = useDX ?
inputVideoFrame.Direct3DSurface.Description.Height :
inputVideoFrame.SoftwareBitmap.PixelHeight;
            var frameWidth = useDX ?
```

```

        inputVideoFrame.Direct3DSurface.Description.Width :
        inputVideoFrame.SoftwreBitmap.PixelWidth;

            var requiredAR = ((float)SIZE / SIZE);
            w = Math.Min((uint)(requiredAR * frameHeight),
(uint)frameWidth);
            h = Math.Min((uint)(frameWidth / requiredAR),
(uint)frameHeight);
            cropBounds.X = (uint)((frameWidth - w) / 2);
            cropBounds.Y = 0;
            cropBounds.Width = w;
            cropBounds.Height = h;

            cropped_vf = new VideoFrame(BitmapPixelFormat.Bgra8, SIZE, SIZE,
BitmapAlphaMode.Ignore);

            await inputVideoFrame.CopyToAsync(cropped_vf, cropBounds, null);
            return cropped_vf;
        }
    }
}

```

Now, let's convert the image to the appropriate format.

The `bestModelInput` class initializes the input types that the model expects. In our case, we configured our code to expect an `ImageFeatureValue`.

The `ImageFeatureValue` class describes the properties of the image used to pass into a model. To create an `ImageFeatureValue`, you use `CreateFromVideoFrame` method. For more specific details of why this is the case and how these classes and methods work, see the [ImageFeatureValue class documentation](#).

Note

In this tutorial, we use the `ImageFeatureValue` class instead of a tensor. If Window ML does not support your model's color format, this won't be an option. For an example of how to work with image conversions and tensorization, see the [Custom Tensorization Sample ↗](#).

5. Add the implementation of the `convert()` method to your `MainPage.xaml.cs` code file inside the `MainPage` class. The `convert` method will get us a representation of the input file in a BGRA8 format.

C#

```

// A method to convert and bind the input image.
private async Task imageBind()

```

```

    {
        UIPreviewImage.Source = null;
        try
        {
            SoftwareBitmap softwareBitmap;
            using (IRandomAccessStream stream = await
selectedStorageFile.OpenAsync(FileAccessMode.Read))
            {
                // Create the decoder from the stream
                BitmapDecoder decoder = await
BitmapDecoder.CreateAsync(stream);
                // Get the SoftwareBitmap representation of the file in
BGRA8 format
                softwareBitmap = await decoder.GetSoftwareBitmapAsync();
                softwareBitmap = SoftwareBitmap.Convert(softwareBitmap,
BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied);
            }
            // Display the image
            SoftwareBitmapSource imageSource = new
SoftwareBitmapSource();
            await imageSource.SetBitmapAsync(softwareBitmap);
            UIPreviewImage.Source = imageSource;

            // Encapsulate the image within a VideoFrame to be bound and
evaluated
            VideoFrame inputImage =
VideoFrame.CreateWithSoftwareBitmap(softwareBitmap);
            // Resize the image size to 224x224
            inputImage=await
helper.CropAndDisplayInputImageAsync(inputImage);
            // Bind the model input with image
            ImageFeatureValue imageTensor =
ImageFeatureValue.CreateFromVideoFrame(inputImage);
            image.input1 = imageTensor;

            // Encapsulate the image within a VideoFrame to be bound and
evaluated
            VideoFrame inputImage =
VideoFrame.CreateWithSoftwareBitmap(softwareBitmap);
            // Bind the input image
            ImageFeatureValue imageTensor =
ImageFeatureValue.CreateFromVideoFrame(inputImage);
            image.modelInput = imageTensor;

        }
        catch (Exception e)
        {
        }
    }
}

```

Bind and Evaluate the model

Next, you'll create a session based on the model, bind the input and output from the session, and evaluate the model.

Create a session to bind the model:

To create a session, you use the `LearningModelSession` class. This class is used to evaluate machine learning models, and binds the model to a device that then runs and evaluates the model. You can select a device when you create a session to execute your model on a specific device of your machine. The default device is the CPU.

ⓘ Note

To learn more about how to choose a device, please review the [Create a session](#) documentation.

Bind model inputs and outputs:

To bind input and output, you use the `LearningModelBinding` class. A machine learning model has input and output features, which pass information into and out of the model. Be aware that required features must be supported by the Window ML APIs. The `LearningModelBinding` class is applied on a `LearningModelSession` to bind values to named input and output features.

The implementation of the binding is automatically generated by mlgen, so you don't have to take care of it. The binding is implemented by calling the predefined methods of the `LearningModelBinding` class. In our case, it uses the `Bind` method to bind a value to the named feature type.

Evaluate the model:

After you create a session to bind the model and bounded values to a model's inputs and outputs, you can evaluate the model's inputs and get its predictions. To run the model execution, you should call any of the predefined evaluate methods on the `LearningModelSession`. In our case, we'll use the `EvaluateAsync` method.

Similar to `CreateFromStreamAsync`, the `EvaluateAsync` method was also automatically generated by WinML Code Generator, so you don't need to implement this method. You can review this method in the `bestModel.cs` file.

The `EvaluateAsync` method will asynchronously evaluate the machine learning model using the feature values already bound in bindings. It will create a session with `LearningModelSession`, bind the input and output with `LearningModelBinding`, execute the model evaluation, and get the output features of the model using the `LearningModelEvaluationResult` class.

ⓘ Note

To learn about other evaluation methods to run the model, please check which methods can be implemented on the `LearningModelSession` by reviewing the [LearningModelSession Class documentation](#).

1. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class to create a session, bind and evaluate the model.

C#

```
// A method to evaluate the model
private async Task evaluate()
{
    results = await modelGen.EvaluateAsync(image);
}
```

Extract and display the results

You'll now need to extract the model output and display the right result, which you'll do by implementing the `extractResult` and `displayResult` methods. You'll need to find the highest probability to return the correct label.

1. Add the `extractResult` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
// A method to extract output from the model
private void extractResult()
{
    // Retrieve the results of evaluation
    var mResult = results.modelOutput as TensorFloat;
    // convert the result to vector format
    var resultVector = mResult.GetAsVectorView();

    probability = 0;
    int index = 0;
```

```
// find the maximum probability
for(int i=0; i<resultVector.Count; i++)
{
    var elementProbability=resultVector[i];
    if (elementProbability > probability)
    {
        index = i;
    }
}
label = ((Labels)index).ToString();
}
```

2. Add the `displayResult` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
private async Task displayResult()
{
    displayOutput.Text = label;
}
```

That's it! You've successfully created the Windows machine learning app with a basic GUI to test our classification model. The next step is to launch the application and run it locally on your Windows device.

Launch the application

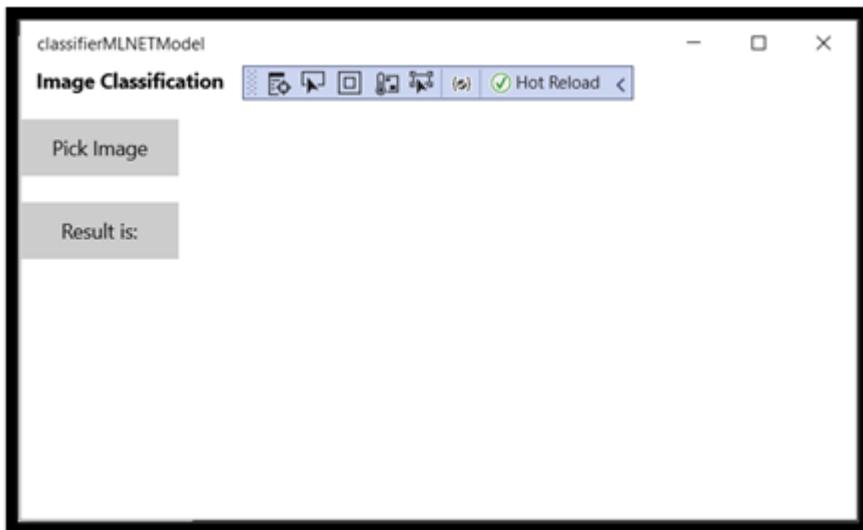
Once you completed the application interface, added the model, and generated the Windows ML code, you can test the application!

Enable developer mode and test your application from Visual Studio. Make sure the dropdown menus in the top toolbar are set to `Debug`. 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.

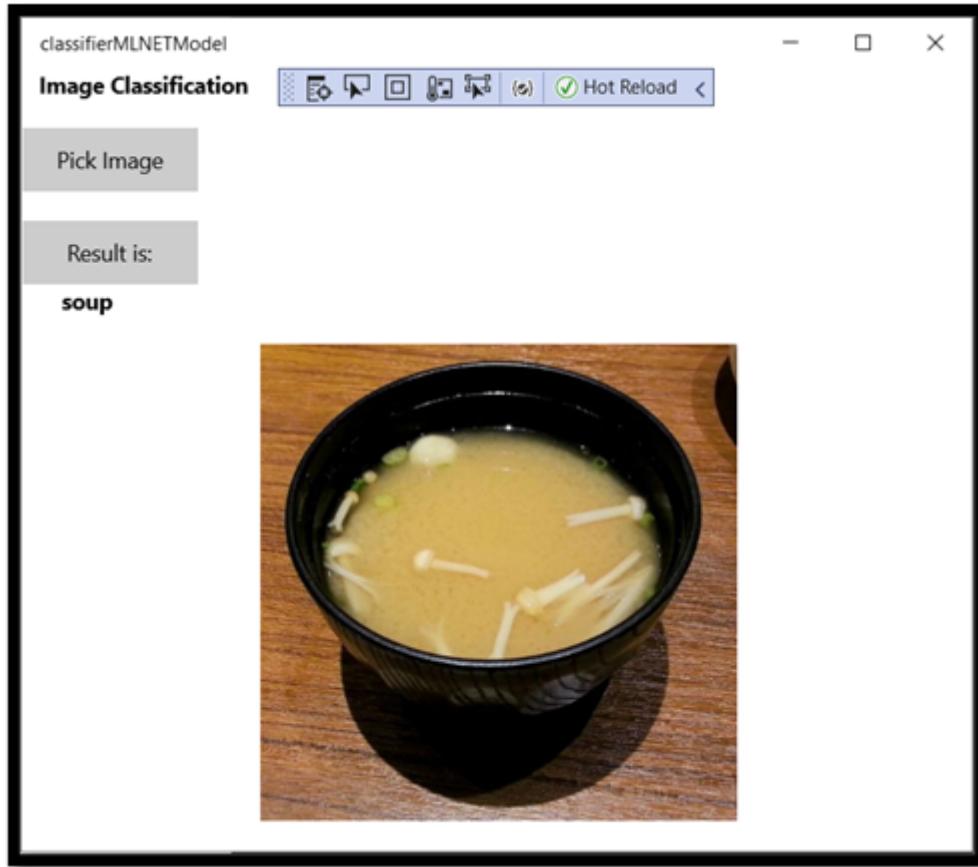
To test our app, let's use the below image of soup. Let's see how our app classifies the content of the image.



1. Save this image on your local device to test the app. Change the image format to `.jpg` if required. You can also add any other relevant image from your local device in a `.jpg` or `.png` format.
2. To run the project, select the `Start Debugging` button on the toolbar, or press `F5`.
3. When the application starts, press `Pick Image` and select the image from your local device.



The result will appear on the screen right away. As you can see, our Windows ML app successfully classified the image as a soup.



Summary

You've just made your first Windows Machine Learning app, from model creation to successful execution.

Additional Resources

To learn more about topics mentioned in this tutorial, visit the following resources:

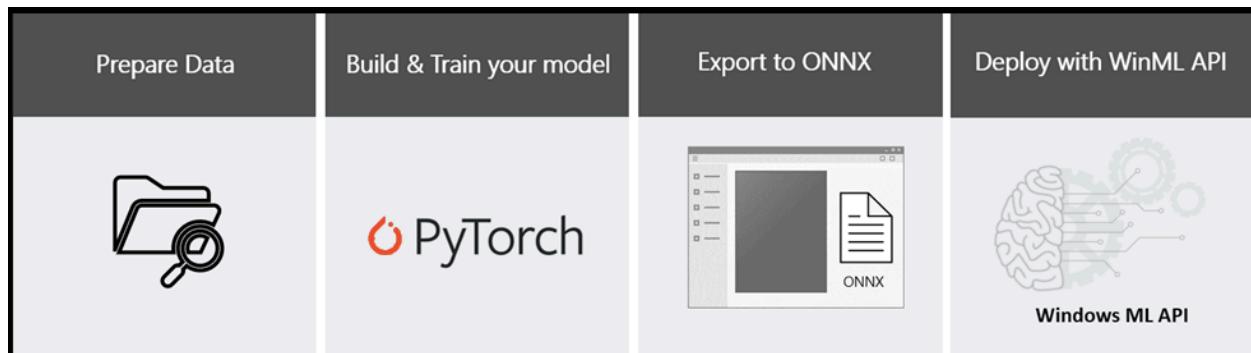
- Windows ML tools: Learn more tools like the [Windows ML Dashboard](#), [WinMLRunner](#), and the [mglgen](#) Windows ML code generator.
- ONNX model: Learn more about the ONNX format.
- [Windows ML performance and memory](#): Learn more how to manage app performance with Windows ML.
- [Windows Machine Learning API reference](#): Learn more about three areas of Windows ML APIs.

Image Classification with PyTorch and Windows ML

Article • 06/28/2024

ⓘ Note

For greater functionality, [PyTorch can also be used with DirectML on Windows](#).



This tutorial will show you how to train an image classification neural network model using PyTorch, export the model to the ONNX format, and deploy it in a Windows Machine Learning application running locally on your Windows device.

Basic knowledge in Python and C# programming languages is required. Previous experience in machine learning is preferable but not required.

If you'd like to move straight on to installation, see [Install PyTorch](#).

If you've set up PyTorch already, start the model training process by [getting the data](#).

Once you're ready to go with the data, you can start to [train your model](#), and then [convert it to the ONNX format](#).

If you have an ONNX model and want to learn how to create a WinML app from scratch, navigate to [deploy your model](#).

ⓘ Note

If you want, you can clone the Windows Machine Learning samples repo and run the completed code for this tutorial. You can find [the PyTorch training solution here](#), or [the completed Windows ML app here](#). If you're using the PyTorch file, make sure you set up the relevant PyTorch interpreter before you run it.

Scenario

In this tutorial, we'll create a machine learning image classification application that can run on any Windows device. The model will be trained to recognize types of patterns, and will classify 10 labels of images from the chosen training set.

Prerequisites for PyTorch - model training:

PyTorch is supported on the following Windows distributions:

- Windows 7 and greater. Windows 10 or greater recommended.
- Windows Server 2008 r2 and greater

To use Pytorch on Windows, you must have Python 3.x installed. Python 2.x is not supported.

Prerequisites for Windows ML app deployment

To create and deploy a WinML app, you'll need the following:

- Windows 10 version 1809 (build 17763) or higher. You can check your build version number by running `winver` via the Run command (`Windows logo key + R`).
- Windows SDK for build 17763 or higher. [You can get the SDK here](#).
- Visual Studio 2017 version 15.7 or later. We recommend using Visual Studio 2019, and some screenshots in this tutorial may be different if you use VS2017 instead. [You can get Visual Studio here](#).
- Windows ML Code Generator (mlgen) Visual Studio extension. Download for [VS 2019](#) or for [VS 2017](#).
- You'll also need to [enable Developer Mode on your PC](#)

ⓘ Note

Windows ML APIs are built into the latest versions of Windows 10 (1809 or higher) and Windows Server 2019. If your target platform is older versions of Windows, you can [port your WinML app to the redistributable NuGet package \(Windows 8.1 or higher\)](#).

Next Steps

We'll start by [installing PyTorch and configuring our environment](#)

 **Important**

PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Install and configure PyTorch on your machine.

Article • 07/01/2024

ⓘ Note

For greater functionality, [PyTorch can also be used with DirectML on Windows](#).

In the [previous stage of this tutorial](#), we discussed the basics of PyTorch and the prerequisites of using it to create a machine learning model. Here, we'll install it on your machine.

Get PyTorch

First, you'll need to setup a Python environment.

We recommend setting up a virtual Python environment inside Windows, using Anaconda as a package manager. The rest of this setup assumes you use an Anaconda environment.

1. [Download and install Anaconda here](#). Select `Anaconda 64-bit installer for Windows Python 3.8`.

ⓘ Important

Be aware to install Python 3.x. Currently, PyTorch on Windows only supports Python 3.x; Python 2.x is not supported.

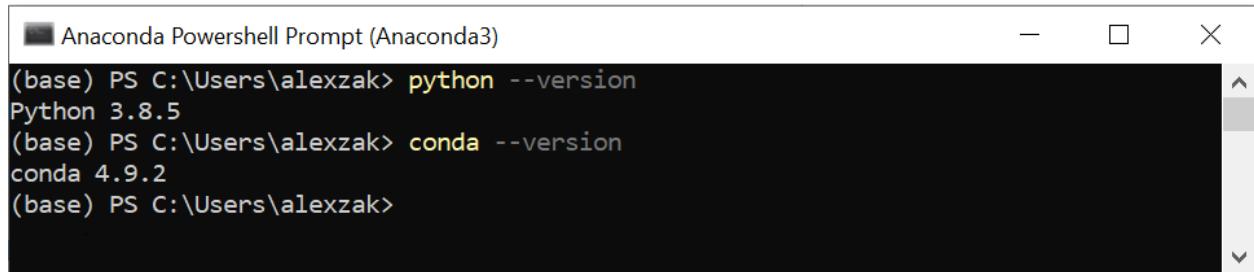


After the installation is complete, verify your Anaconda and Python versions.

2. Open Anaconda manager via Start - Anaconda3 - Anaconda PowerShell Prompt and test your versions:

You can check your Python version by running the following command: `python --version`

You can check your Anaconda version by running the following command: `conda --version`



A screenshot of a Windows command prompt window titled "Anaconda Powershell Prompt (Anaconda3)". The window shows the following output:

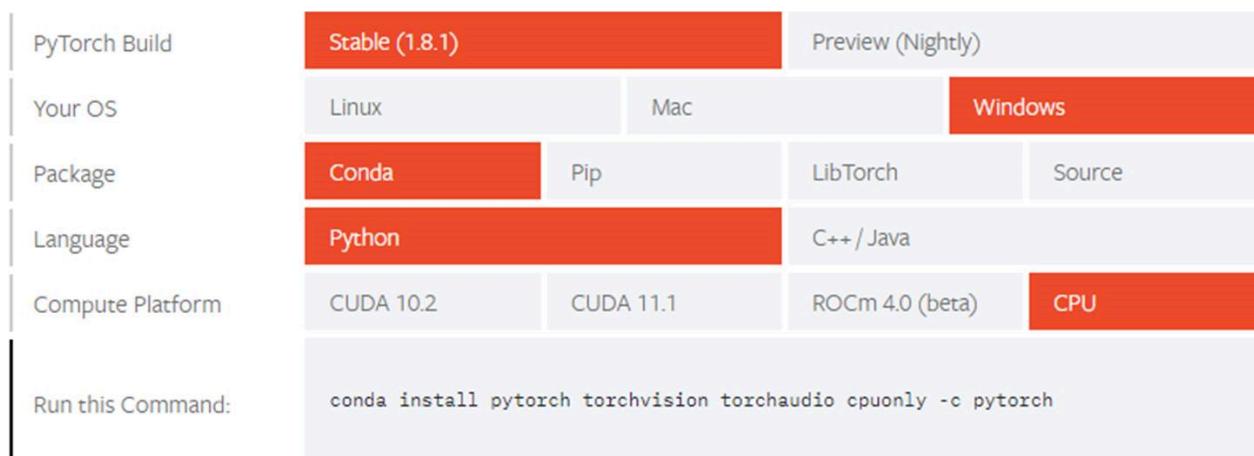
```
(base) PS C:\Users\alexzak> python --version
Python 3.8.5
(base) PS C:\Users\alexzak> conda --version
conda 4.9.2
(base) PS C:\Users\alexzak>
```

Now, you can install PyTorch package from binaries via Conda.

3. Navigate to <https://pytorch.org/>.

Select the relevant PyTorch installation details:

- PyTorch build – stable.
- Your OS – Windows
- Package – Conda
- Language – Python
- Compute Platform – CPU, or choose your version of Cuda. In this tutorial, you will train and inference model on CPU, but you could use a Nvidia GPU as well.



A screenshot of the PyTorch download page. The user has selected the following options:

- PyTorch Build: Stable (1.8.1)
- Your OS: Windows
- Package: Conda
- Language: Python
- Compute Platform: CPU

The "Run this Command:" field contains the following Conda command:

```
conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

4. Open Anaconda manager and run the command as it specified in the installation instructions.

```
conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\alexzak> conda install pytorch torchvision torchaudio cpuonly -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\Users\alexzak\Anaconda3

added / updated specs:
- cpuonly
- pytorch
- torchaudio
- torchvision

The following packages will be downloaded:

  package          build
  -----          -----
  conda-4.9.2      py37haa95532_0    2.9 MB
  cpuonly-1.0       0                 2 KB  pytorch
  libuv-1.40.0      he774522_0     255 KB
  ninja-1.10.2      py37h6d14046_0   246 KB
  pytorch-1.7.1      py3.7_cpu_0    157.0 MB  pytorch
  torchaudio-0.7.2      py37             2.7 MB  pytorch
  torchvision-0.8.2      py37_cpu        6.6 MB  pytorch
  typing_extensions-3.7.4.3      py_0           28 KB
  -----
                           Total:      169.7 MB

The following NEW packages will be INSTALLED:

cpuonly      pytorch/noarch::cpuonly-1.0-0
libuv         pkgs/main/win-64::libuv-1.40.0-he774522_0
ninja          pkgs/main/win-64::ninja-1.10.2-py37h6d14046_0
pytorch        pytorch/win-64::pytorch-1.7.1-py3.7_cpu_0
torchaudio     pytorch/win-64::torchaudio-0.7.2-py37
torchvision    pytorch/win-64::torchvision-0.8.2-py37_cpu
typing_extensions  pkgs/main/noarch::typing_extensions-3.7.4.3-py_0

The following packages will be UPDATED:

  conda          4.8.2-py37_0 --> 4.9.2-py37haa95532_0

Proceed ([y]/n)?
```

5. Confirm and complete the extraction of the required packages.

```
Anaconda Powershell Prompt (Anaconda3)
Proceed ([y]/n)? y

Downloading and Extracting Packages
pytorch-1.7.1      | 1007.0 MB | ######=-----#
##### | 100%
torchaudio-0.7.2    | 2.7 MB   | #####-----#
##### | 100%
libuv-1.40.0        | 255 KB  | #####-----#
##### | 100%
torchvision-0.8.2    | 7.3 MB   | #####-----#
##### | 100%
ninja-1.10.2        | 247 KB  | #####-----#
##### | 100%
cuda toolkit-11.0.221 | 627.0 MB | #####-----#
##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) PS C:\Users\alexzak>
```

Let's verify PyTorch installation by running sample PyTorch code to construct a randomly initialized tensor.

6. Open the Anaconda PowerShell Prompt and run the following command.

```
python
```

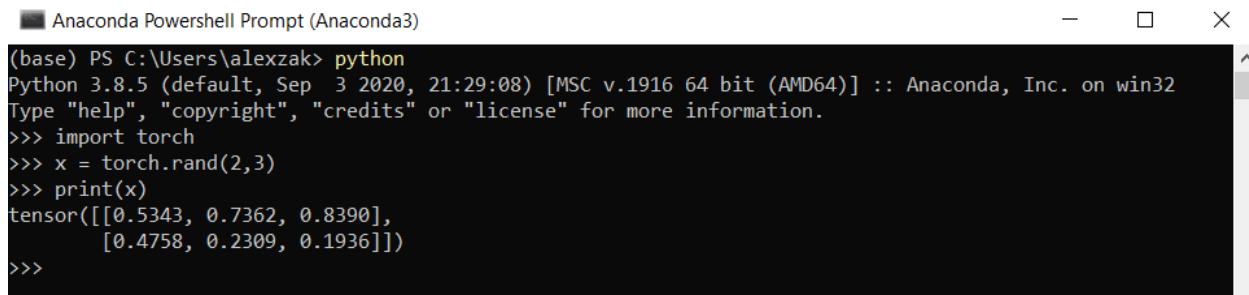
Next, enter the following code:

```
import torch

x = torch.rand(2, 3)

print(x)
```

The output should be a random 2x3 tensor. The numbers will be different, but it should look similar to the below.



A screenshot of the Anaconda Powershell Prompt window. The title bar says "Anaconda Powershell Prompt (Anaconda3)". The command line shows the following Python session:

```
(base) PS C:\Users\alexzak> python
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> x = torch.rand(2,3)
>>> print(x)
tensor([[0.5343,  0.7362,  0.8390],
       [0.4758,  0.2309,  0.1936]])
>>>
```

ⓘ Note

Interested in learning more? Visit the [PyTorch official website](#) ↗

Next Steps

Now that we've installed PyTorch, we're ready to [set up the data for our model](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) ↗ | [Get help at Microsoft Q&A](#)

Prepare the data

Article • 06/28/2024

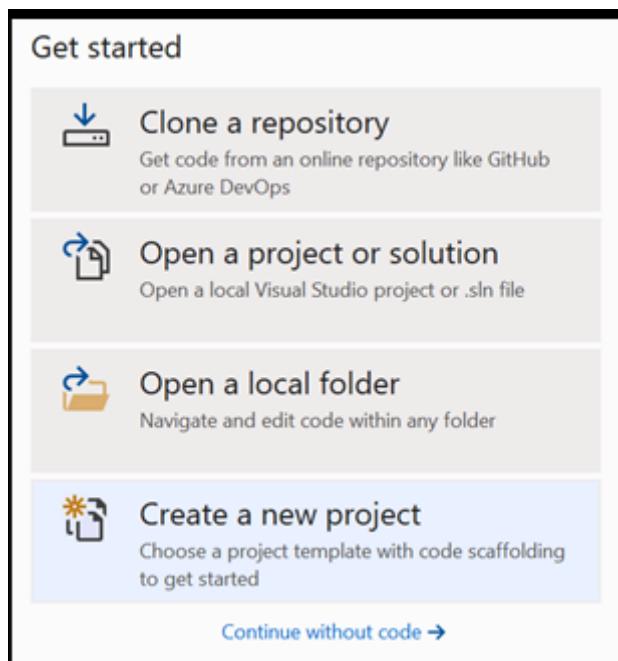
ⓘ Note

For greater functionality, [PyTorch can also be used with DirectML on Windows](#).

In the [previous stage of this tutorial](#), we installed PyTorch on your machine. Now, we'll use it to set up our code with the data we'll use to make our model.

Open a new project within Visual Studio.

1. Open Visual Studio and choose `create a new project`.



2. In the search bar, type `Python` and select `Python Application` as your project template.

The screenshot shows the PyCharm 'New Project' dialog. At the top, there is a search bar with 'python' and a 'Clear all' button. Below the search bar are three dropdown menus: 'Python', 'Windows', and 'All project types'. The 'All project types' menu is currently selected. The main area displays five project templates:

- Python Application**: A project for creating a command-line application. It includes tabs for Python, Windows, Linux, macOS, and Console.
- From Existing Python code**: Create a new project using code files that are already in a folder hierarchy. It includes tabs for Python, Linux, macOS, Windows, Console, and Web.
- Web Project**: A project for creating a generic Python web project. It includes tabs for Python, Windows, Linux, macOS, and Web.
- Django Web Project**: A project for creating an application using the Django web framework. It features sample pages that use the Twitter Bootstrap framework for responsive web design. It includes tabs for Python, Windows, Linux, macOS, and Web.
- Flask Web Project**: A project for creating an application using the Flask web framework with the Jinja template engine. It features sample pages that use the Twitter Bootstrap framework for responsive web design. It includes tabs for Python, Windows, Linux, macOS, and Web.

At the bottom right of the dialog are 'Back' and 'Next' buttons.

3. In the configuration window:

- Name your project. Here, we call it **PyTorchTraining**.
- Choose the location of your project.
- If you're using VS2019, ensure `Create directory for solution` is checked.
- If you're suing VS 2017, ensure `Place solution and project in the same directory` is unchecked.

Configure your new project

Python Application Python Windows Linux macOS Console

Project name

PyTorchTraining

Location

C:\Users\alexzak\Source\Repos



Solution name i

PyTorchTraining

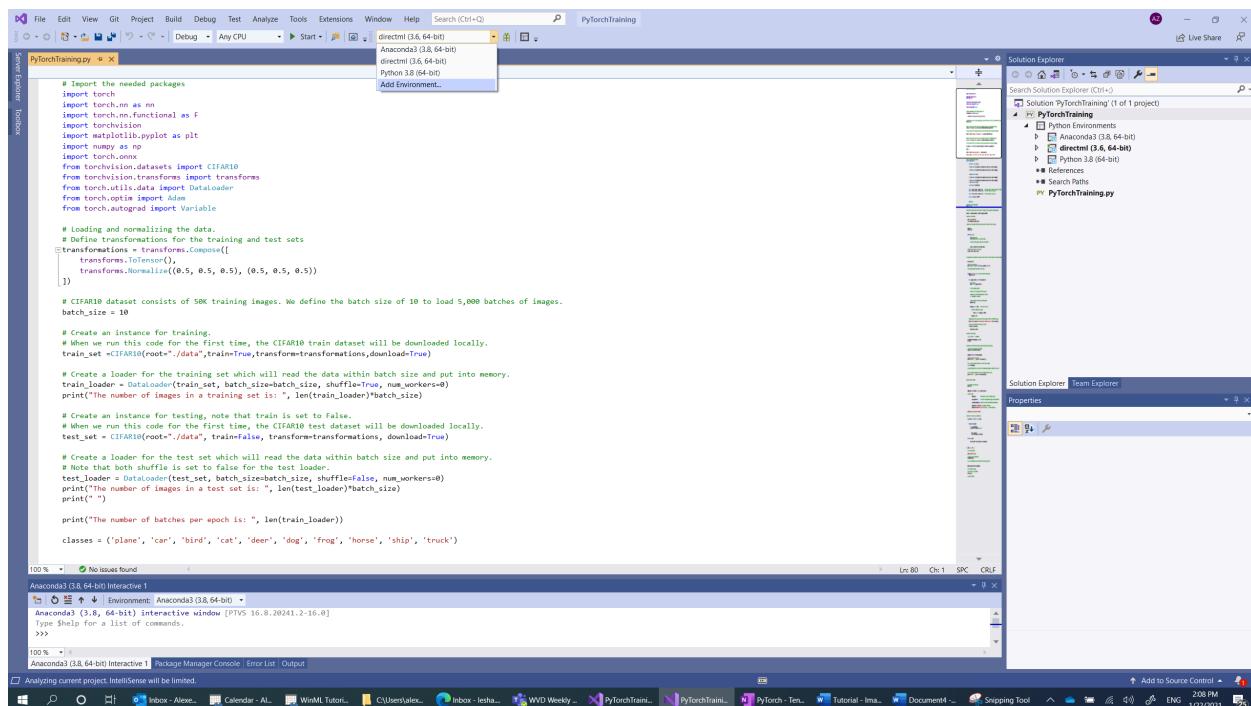
Place solution and project in the same directory

Press **Create** to create your project.

Create a Python interpreter

Now, you need to define a new Python interpreter. This must include the PyTorch package you've recently installed.

1. Navigate to interpreter selection, and select **Add environment**:



2. In the **Add environment** window, select **Existing environment**, and choose

Anaconda3 (3.6, 64-bit). This includes the PyTorch package.

Add environment

Virtual environment
Conda environment
Existing environment
Python installation

Project
PyTorchTraining

Environment
Anaconda3 (3.8, 64-bit)

To test the new Python interpreter and PyTorch package, enter the following code to the `PyTorchTraining.py` file:

```
from __future__ import print_function

import torch

x=torch.rand(2, 3)

print(x)
```

The output should be a random 5x3 tensor similar to the below.

```
tensor([[0.1563, 0.6212, 0.2504],
        [0.4116, 0.2615, 0.6808]])
Press any key to continue . . .
```

ⓘ Note

Interested in learning more? Visit the [PyTorch official website](#) ↗.

Load the dataset

You'll use the PyTorch `torchvision` class to load the data.

The Torchvision library includes several popular datasets such as Imagenet, CIFAR10, MNIST, etc, model architectures, and common image transformations for computer vision. That makes data loading in PyTorch quite an easy process.

CIFAR10

Here, we'll use the CIFAR10 dataset to build and train the image classification model. CIFAR10 is a widely used dataset for machine learning research. It consists of 50,000 training images and 10,000 test images. All of them are of size 3x32x32, which means 3-channel color images of 32x32 pixels in size.

The images are divided to 10 classes: 'airplane' (0), 'automobile' (1), 'bird' (2), 'cat' (3) , 'deer' (4), 'dog' (5), 'frog' (6), 'horse' (7), 'ship' (8), 'truck' (9).

You will follow three steps to load and read the CIFAR10 dataset in PyTorch:

- Define transformations to be applied to the image: To train the model, you need to transform the images to Tensors of normalized range [-1,1].
- Create an instance of the available dataset and load the dataset: To load the data, you'll use the `torch.utils.data.Dataset` class - an abstract class for representing a dataset. The dataset will be downloaded locally only the first time you run the code.
- Access the data using the DataLoader. To get the access to the data and put the data into memory, you'll use the `torch.utils.data.DataLoader` class. DataLoader in PyTorch wraps a dataset and provides access to the underlying data. This wrapper will hold batches of images per defined batch size.

You'll repeat these three steps for both training and testing sets.

1. Open the `PyTorchTraining.py` file in Visual Studio, and add the following code. This handles the three above steps for the training and test data sets from the CIFAR10 dataset.

```
py

from torchvision.datasets import CIFAR10
from torchvision.transforms import transforms
from torch.utils.data import DataLoader

# Loading and normalizing the data.
# Define transformations for the training and test sets
transformations = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# CIFAR10 dataset consists of 50K training images. We define the batch size
# of 10 to load 5,000 batches of images.
batch_size = 10
number_of_labels = 10

# Create an instance for training.
# When we run this code for the first time, the CIFAR10 train dataset will
# be downloaded locally.
```

```

train_set
=CIFAR10(root="../data", train=True, transform=transformations, download=True)

# Create a loader for the training set which will read the data within batch
size and put into memory.
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True,
num_workers=0)
print("The number of images in a training set is: ",
len(train_loader)*batch_size)

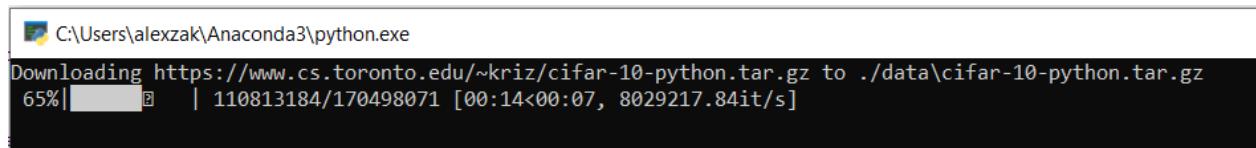
# Create an instance for testing, note that train is set to False.
# When we run this code for the first time, the CIFAR10 test dataset will be
downloaded locally.
test_set = CIFAR10(root="../data", train=False, transform=transformations,
download=True)

# Create a loader for the test set which will read the data within batch
size and put into memory.
# Note that each shuffle is set to false for the test loader.
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False,
num_workers=0)
print("The number of images in a test set is: ",
len(test_loader)*batch_size)

print("The number of batches per epoch is: ", len(train_loader))
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
'ship', 'truck')

```

The first time you run this code, the CIFAR10 dataset will be downloaded to your device.



C:\Users\alexzak\Anaconda3\python.exe
 Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data\cifar-10-python.tar.gz
 65%|██████████| 110813184/170498071 [00:14<00:07, 8029217.84it/s]

Next Steps

With the data ready to go, it's time to [train our PyTorch model](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Train your image classifier model with PyTorch

Article • 06/28/2024

ⓘ Note

For greater functionality, [PyTorch can also be used with DirectML on Windows](#).

In the [previous stage of this tutorial](#), we acquired the dataset we'll use to train our image classifier with PyTorch. Now, it's time to put that data to use.

To train the image classifier with PyTorch, you need to complete the following steps:

1. Load the data. If you've done the previous step of this tutorial, you've handled this already.
2. Define a Convolution Neural Network.
3. Define a loss function.
4. Train the model on the training data.
5. Test the network on the test data.

Define a Convolution Neural Network.

To build a neural network with PyTorch, you'll use the `torch.nn` package. This package contains modules, extensible classes and all the required components to build neural networks.

Here, you'll build a basic **convolution neural network** (CNN) to classify the images from the CIFAR10 dataset.

A CNN is a class of neural networks, defined as multilayered neural networks designed to detect complex features in data. They're most commonly used in computer vision applications.

Our network will be structured with the following 14 layers:

```
Conv -> BatchNorm -> ReLU -> Conv -> BatchNorm -> ReLU -> MaxPool -> Conv ->  
BatchNorm -> ReLU -> Conv -> BatchNorm -> ReLU -> Linear.
```

The convolution layer

The convolution layer is a main layer of CNN which helps us to detect features in images. Each of the layers has number of channels to detect specific features in images, and a number of kernels to define the size of the detected feature. Therefore, a convolution layer with 64 channels and kernel size of 3×3 would detect 64 distinct features, each of size 3×3 . When you define a convolution layer, you provide the number of in-channels, the number of out-channels, and the kernel size. The number of out-channels in the layer serves as the number of in-channels to the next layer.

For example: A Convolution layer with in-channels=3, out-channels=10, and kernel-size=6 will get the RGB image (3 channels) as an input, and it will apply 10 feature detectors to the images with the kernel size of 6×6 . Smaller kernel sizes will reduce computational time and weight sharing.

Other layers

The following other layers are involved in our network:

- The `ReLU` layer is an activation function to define all incoming features to be 0 or greater. When you apply this layer, any number less than 0 is changed to zero, while others are kept the same.
- the `BatchNorm2d` layer applies normalization on the inputs to have zero mean and unit variance and increase the network accuracy.
- The `MaxPool` layer will help us to ensure that the location of an object in an image will not affect the ability of the neural network to detect its specific features.
- The `Linear` layer is final layers in our network, which computes the scores of each of the classes. In the CIFAR10 dataset, there are ten classes of labels. The label with the highest score will be the one model predicts. In the linear layer, you have to specify the number of input features and the number of output features which should correspond to the number of classes.

How does a Neural Network work?

The CNN is a feed-forward network. During the training process, the network will process the input through all the layers, compute the loss to understand how far the predicted label of the image is falling from the correct one, and propagate the gradients back into the network to update the weights of the layers. By iterating over a huge dataset of inputs, the network will “learn” to set its weights to achieve the best results.

A forward function computes the value of the loss function, and the backward function computes the gradients of the learnable parameters. When you create our neural

network with PyTorch, you only need to define the forward function. The backward function will be automatically defined.

1. Copy the following code into the `PyTorchTraining.py` file in Visual Studio to define the CCN.

```
py

import torch
import torch.nn as nn
import torchvision
import torch.nn.functional as F

# Define a convolution neural network
class Network(nn.Module):
    def __init__(self):
        super(Network, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=12,
kernel_size=5, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(12)
        self.conv2 = nn.Conv2d(in_channels=12, out_channels=12,
kernel_size=5, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(12)
        self.pool = nn.MaxPool2d(2,2)
        self.conv4 = nn.Conv2d(in_channels=12, out_channels=24,
kernel_size=5, stride=1, padding=1)
        self.bn4 = nn.BatchNorm2d(24)
        self.conv5 = nn.Conv2d(in_channels=24, out_channels=24,
kernel_size=5, stride=1, padding=1)
        self.bn5 = nn.BatchNorm2d(24)
        self.fc1 = nn.Linear(24*10*10, 10)

    def forward(self, input):
        output = F.relu(self.bn1(self.conv1(input)))
        output = F.relu(self.bn2(self.conv2(output)))
        output = self.pool(output)
        output = F.relu(self.bn4(self.conv4(output)))
        output = F.relu(self.bn5(self.conv5(output)))
        output = output.view(-1, 24*10*10)
        output = self.fc1(output)

    return output

# Instantiate a neural network model
model = Network()
```

 **Note**

Interested in learning more about neural network with PyTorch? Check out the [PyTorch documentation](#)

Define a loss function

A loss function computes a value that estimates how far away the output is from the target. The main objective is to reduce the loss function's value by changing the weight vector values through backpropagation in neural networks.

Loss value is different from model accuracy. Loss function gives us the understanding of how well a model behaves after each iteration of optimization on the training set. The accuracy of the model is calculated on the test data and shows the percentage of the right prediction.

In PyTorch, the neural network package contains various loss functions that form the building blocks of deep neural networks. In this tutorial, you will use a Classification loss function based on Define the loss function with Classification Cross-Entropy loss and an Adam Optimizer. Learning rate (lr) sets the control of how much you are adjusting the weights of our network with respect the loss gradient. You will set it as 0.001. The lower it is, the slower the training will be.

1. Copy the following code into the `PyTorchTraining.py` file in Visual Studio to define the loss function and an optimizer.

```
py
```

```
from torch.optim import Adam

# Define the loss function with Classification Cross-Entropy loss and an
optimizer with Adam optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
```

Train the model on the training data.

To train the model, you have to loop over our data iterator, feed the inputs to the network, and optimize. PyTorch doesn't have a dedicated library for GPU use, but you can manually define the execution device. The device will be an Nvidia GPU if exists on your machine, or your CPU if it does not.

1. Add the following code to the `PyTorchTraining.py` file

py

```
from torch.autograd import Variable

# Function to save the model
def saveModel():
    path = "./myFirstModel.pth"
    torch.save(model.state_dict(), path)

# Function to test the model with the test dataset and print the accuracy
# for the test images
def testAccuracy():

    model.eval()
    accuracy = 0.0
    total = 0.0
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            # run the model on the test set to predict labels
            outputs = model(images.to(device))
            # the label with the highest energy will be our prediction
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            accuracy += (predicted == labels.to(device)).sum().item()

    # compute the accuracy over all test images
    accuracy = (100 * accuracy / total)
    return(accuracy)

# Training function. We simply have to loop over our data iterator and feed
# the inputs to the network and optimize.
def train(num_epochs):

    best_accuracy = 0.0

    # Define your execution device
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print("The model will be running on", device, "device")
    # Convert model parameters and buffers to CPU or Cuda
    model.to(device)

    for epoch in range(num_epochs): # loop over the dataset multiple times
        running_loss = 0.0
        running_acc = 0.0

        for i, (images, labels) in enumerate(train_loader, 0):

            # get the inputs
            images = Variable(images.to(device))
            labels = Variable(labels.to(device))
```

```

# zero the parameter gradients
optimizer.zero_grad()
# predict classes using images from the training set
outputs = model(images)
# compute the loss based on model output and real labels
loss = loss_fn(outputs, labels)
# backpropagate the loss
loss.backward()
# adjust parameters based on the calculated gradients
optimizer.step()

# Let's print statistics for every 1,000 images
running_loss += loss.item()      # extract the loss value
if i % 1000 == 999:
    # print every 1000 (twice per epoch)
    print('[%d, %5d] loss: %.3f' %
          (epoch + 1, i + 1, running_loss / 1000))
    # zero the loss
    running_loss = 0.0

# Compute and print the average accuracy fo this epoch when tested
over all 10000 test images
accuracy = testAccuracy()
print('For epoch', epoch+1, 'the test accuracy over the whole test
set is %d %%' % (accuracy))

# we want to save the model if the accuracy is the best
if accuracy > best_accuracy:
    saveModel()
    best_accuracy = accuracy

```

Test the model on the test data.

Now, you can test the model with batch of images from our test set.

1. Add the following code to the `PyTorchTraining.py` file.

```

py

import matplotlib.pyplot as plt
import numpy as np

# Function to show the images
def imagedshow(img):
    img = img / 2 + 0.5      # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

```

```

# Function to test the model with a batch of images and show the labels
predictions
def testBatch():
    # get batch of images from the test DataLoader
    images, labels = next(iter(test_loader))

    # show all images as one image grid
    imageshow(torchvision.utils.make_grid(images))

    # Show the real labels on the screen
    print('Real labels: ', ''.join('%5s' % classes[labels[j]]
                                   for j in range(batch_size)))

    # Let's see what if the model identifiers the labels of those example
    outputs = model(images)

    # We got the probability for every 10 labels. The highest (max)
    probability should be correct label
    _, predicted = torch.max(outputs, 1)

    # Let's show the predicted labels on the screen to compare with the real
    ones
    print('Predicted: ', ''.join('%5s' % classes[predicted[j]]
                                 for j in range(batch_size)))

```

Finally, let's add the main code. This will initiate model training, save the model, and display the results on the screen. We'll run only two iterations [`train(2)`] over the training set, so the training process won't take too long.

2. Add the following code to the `PyTorchTraining.py` file.

```

py

if __name__ == "__main__":
    # Let's build our model
    train(5)
    print('Finished Training')

    # Test which classes performed well
    testAccuracy()

    # Let's load the model we just created and test the accuracy per label
    model = Network()
    path = "myFirstModel.pth"
    model.load_state_dict(torch.load(path))

    # Test with batch of images
    testBatch()

```

Let's run the test! Make sure the dropdown menus in the top toolbar are set to Debug. 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.

Choosing the epoch number (the number of complete passes through the training dataset) equal to two (`[train(2)]`) will result in iterating twice through the entire test dataset of 10,000 images. It will take around 20 minutes to complete the training on 8th Generation Intel CPU, and the model should achieve more or less 65% of success rate in the classification of ten labels.

3. To run the project, click the Start Debugging button on the toolbar, or press F5.

The console window will pop up and will be able to see the process of training.

As you defined, the loss value will be printed every 1,000 batches of images or five times for every iteration over the training set. You expect the loss value to decrease with every loop.

You'll also see the accuracy of the model after each iteration. Model accuracy is different from the loss value. Loss function gives us the understanding of how well a model behaves after each iteration of optimization on the training set. The accuracy of the model is calculated on the test data and shows the percentage of the right prediction. In our case it will tell us how many images from the 10,000-image test set our model was able to classify correctly after each training iteration.

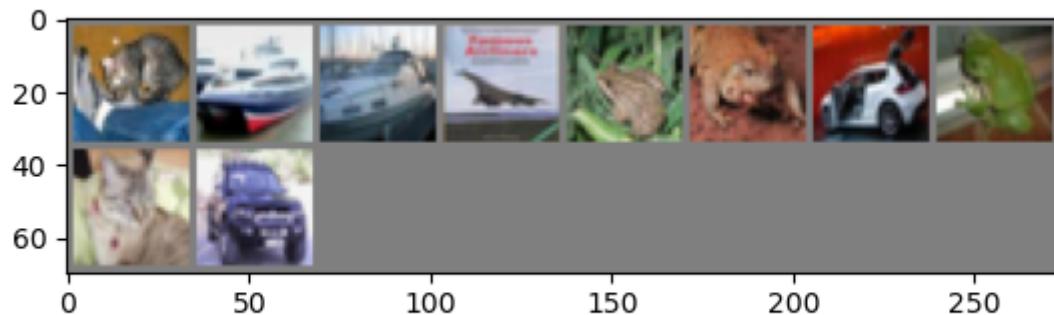
Once the training is complete, you should expect to see the output similar to the below. Your numbers won't be exactly the same - trianing depends on many factors, and won't always return identifical results - but they should look similar.

```
C:\Users\alexzak\Anaconda3\python.exe
Files already downloaded and verified
The number of images in a training set is: 50000
Files already downloaded and verified
The number of images in a test set is: 10000

The number of batches per epoch is: 5000
The model will be running on cpu device
[1, 1000] loss: 1.782
[1, 2000] loss: 1.445
[1, 3000] loss: 1.309
[1, 4000] loss: 1.203
[1, 5000] loss: 1.115
For epoch 1 the test accuracy over the whole test set is 62 %
[2, 1000] loss: 1.073
[2, 2000] loss: 1.004
[2, 3000] loss: 1.014
[2, 4000] loss: 0.964
[2, 5000] loss: 0.960
For epoch 2 the test accuracy over the whole test set is 64 %
[3, 1000] loss: 0.863
[3, 2000] loss: 0.894
[3, 3000] loss: 0.892
[3, 4000] loss: 0.871
[3, 5000] loss: 0.862
For epoch 3 the test accuracy over the whole test set is 69 %
[4, 1000] loss: 0.780
[4, 2000] loss: 0.798
[4, 3000] loss: 0.795
[4, 4000] loss: 0.816
[4, 5000] loss: 0.797
For epoch 4 the test accuracy over the whole test set is 70 %
[5, 1000] loss: 0.696
[5, 2000] loss: 0.730
[5, 3000] loss: 0.762
[5, 4000] loss: 0.736
[5, 5000] loss: 0.758
For epoch 5 the test accuracy over the whole test set is 70 %
Finished Training
Real labels: cat ship ship plane frog frog car frog cat car
Predicted: cat ship car plane frog cat cat bird cat car
Press any key to continue . . .
```

After running just 5 epochs, the model success rate is 70%. This is a good result for a basic model trained for short period of time!

Testing with the batch of images, the model got right 7 images from the batch of 10. Not bad at all and consistent with the model success rate.



You can check which classes our model can predict the best. Simple add the run the code below:

4. **Optional** - add the following `testClassess` function into the `PyTorchTraining.py` file, add a call of this function - `testClassess()` inside the main function - `__name__ == "__main__"`.

```
py

# Function to test what classes performed well
def testClassess():
    class_correct = list(0. for i in range(number_of_labels))
    class_total = list(0. for i in range(number_of_labels))
    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            c = (predicted == labels).squeeze()
            for i in range(batch_size):
                label = labels[i]
                class_correct[label] += c[i].item()
                class_total[label] += 1

    for i in range(number_of_labels):
        print('Accuracy of %5s : %2d %%' % (
            classes[i], 100 * class_correct[i] / class_total[i]))
```

The output is as follows:

```
The number of batches per epoch is: 5000
Accuracy of plane : 76 %
Accuracy of car : 80 %
Accuracy of bird : 51 %
Accuracy of cat : 58 %
Accuracy of deer : 55 %
Accuracy of dog : 30 %
Accuracy of frog : 50 %
Accuracy of horse : 68 %
Accuracy of ship : 77 %
Accuracy of truck : 65 %
```

Next Steps

Now that we have a classification model, the next step is to [convert the model to the ONNX format](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Convert your PyTorch training model to ONNX

Article • 07/01/2024

ⓘ Note

For greater functionality, [PyTorch can also be used with DirectML on Windows](#).

In the [previous stage of this tutorial](#), we used PyTorch to create our machine learning model. However, that model is a `.pth` file. To be able to integrate it with Windows ML app, you'll need to convert the model to ONNX format.

Export the model

To export a model, you will use the `torch.onnx.export()` function. This function executes the model, and records a trace of what operators are used to compute the outputs.

1. Copy the following code into the `PyTorchTraining.py` file in Visual Studio, above your main function.

```
py

import torch.onnx

#Function to Convert to ONNX
def Convert_ONNX():

    # set the model to inference mode
    model.eval()

    # Let's create a dummy input tensor
    dummy_input = torch.randn(1, input_size, requires_grad=True)

    # Export the model
    torch.onnx.export(model,           # model being run
                      dummy_input,      # model input (or a tuple for multiple inputs)
                      "ImageClassifier.onnx",   # where to save the model
                      export_params=True, # store the trained parameter weights inside
    the model file
                      opset_version=10,   # the ONNX version to export the model to
                      do_constant_folding=True, # whether to execute constant folding
    for optimization
                      input_names = ['modelInput'], # the model's input names
                      output_names = ['modelOutput'], # the model's output names
```

```

        dynamic_axes={'modelInput' : {0 : 'batch_size'},      # variable
length axes
                           'modelOutput' : {0 : 'batch_size'}})
print(" ")
print('Model has been converted to ONNX')

```

It's important to call `model.eval()` or `model.train(False)` before exporting the model, as this sets the model to **inference mode**. This is needed since operators like `dropout` or `batchnorm` behave differently in inference and training mode.

2. To run the conversion to ONNX, add a call to the conversion function to the main function. You don't need to train the model again, so we'll comment out some functions that we no longer need to run. Your main function will be as follows.

py

```

if __name__ == "__main__":
    # Let's build our model
    #train(5)
    #print('Finished Training')

    # Test which classes performed well
    #testAccuracy()

    # Let's load the model we just created and test the accuracy per label
    model = Network()
    path = "myFirstModel.pth"
    model.load_state_dict(torch.load(path))

    # Test with batch of images
    #testBatch()
    # Test how the classes performed
    #testClassess()

    # Conversion to ONNX
    Convert_ONNX()

```

3. Run the project again by selecting the `Start Debugging` button on the toolbar, or pressing `F5`. There's no need to train the model again, just load the existing model from the project folder.

The output will be as follows.

```
C:\Users\alexzak\Anaconda3\python.exe
Files already downloaded and verified
The number of images in a training set is: 50000
Files already downloaded and verified
The number of images in a test set is: 10000

The number of batches per epoch is: 10000

Model has been converted to ONNX
Press any key to continue . . .
```

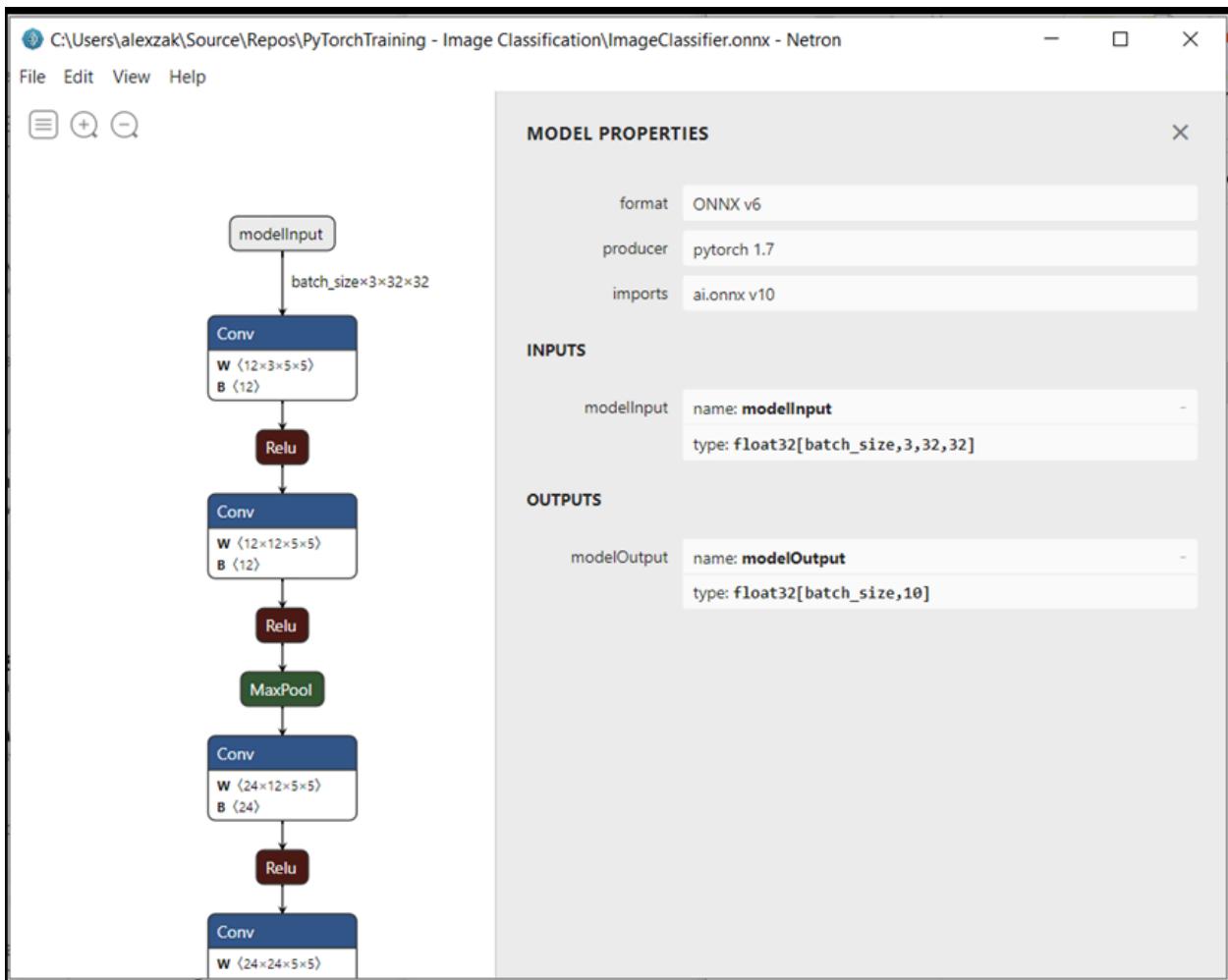
Navigate to your project location and find the ONNX model next to the `.pth` model.

 **Note**

Interested in learning more? Review the [PyTorch tutorial on exporting a model ↗](#).

Explore your model.

1. Open the `ImageClassifier.onnx` model file with Netron.
2. Select the *data* node to open the model properties.



As you can see, the model requires a 32-bit tensor (multi-dimensional array) float object as an input, and returns a Tensor float as an output. The output array will include the probability for every label. The way you built the model, the labels are represented by 10 numbers, and every number represents the ten classes of objects.

[\[+\] Expand table](#)

| Label |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| plane | car | bird | cat | deer | dog | frog | horse | ship | truck |

You'll need to extract these values to show the correct prediction with Windows ML app.

Next Steps

Our model is ready to deploy. Next, for the main event - let's [build a Windows application and run it locally on your Windows device](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Deploy model in Windows app with Windows ML API

Article • 06/28/2024

ⓘ Note

For greater functionality, [PyTorch can also be used with DirectML on Windows](#).

In the previous part of this tutorial, you learned how to build and export a model in ONNX format. Now, we'll show you how to embed your exported model into a Windows application, and run it locally on a device by calling WinML APIs.

By the time we're done, you'll have a working image classification app.

About the sample app

In this step of the tutorial, you'll create an app that can classify images using your ML model. Its basic UI allows you to select an image from your local device, and uses the classification ONNX model you built and trained in the previous part to classify it. The tags returned by the model are then displayed next to the image.

Here, we'll walk you through that process.

ⓘ Note

If you choose to use the predefined code sample, you can clone [the solution file](#). Clone the repository, navigate to this sample, and open the `classifierPyTorch.sln` file with Visual Studio. Skip to the **Launch the Application** part of this page to see it in use.

Below, we'll guide you how to create your app and add Windows ML code.

Create a Windows ML UWP (C#)

To make a working Windows ML app, you'll need to do the following:

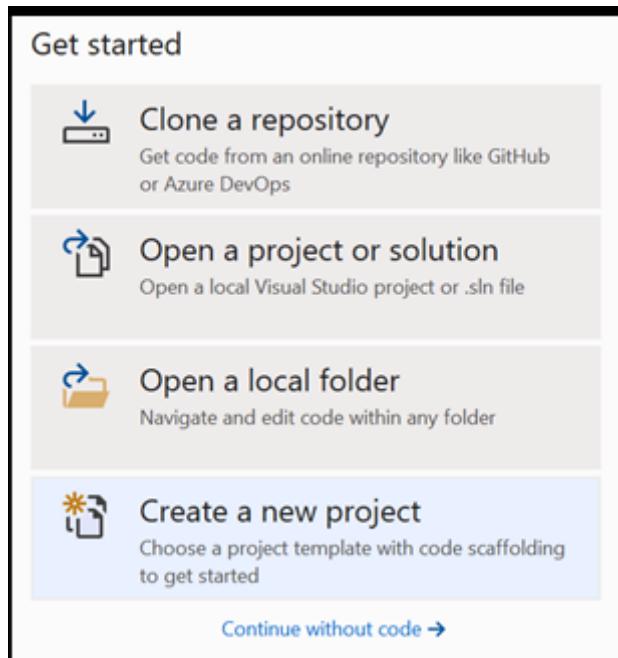
- Load a machine learning model.
- Load an image in a required format.

- Bind the model's inputs and outputs.
- Evaluate the model and display meaningful results.

You'll also need to create a basic UI, as it's hard to create a satisfactory image-based app in the command line.

Open a new project within Visual Studio

1. Let's get started. Open Visual Studio and choose **create a new project**.



2. In the search bar, type `UWP`, then select `Blank APP (Universal Windows)`. This opens a C# project for a single-page Universal Windows Platform (UWP) app with predefined controls or layout. Select `next` to open a configuration window for the project.

UWP

Clear all

C# Windows UWP

 Blank App (Universal Windows)
A project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout.

C# XAML Windows Xbox Desktop UWP

 Unit Test App (Universal Windows)
A project to create a unit test app for Universal Windows Platform (UWP) apps using MSTest.

C# Windows Test UWP

 Class Library (Universal Windows)
A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps.

C# Windows Library UWP

 Windows Runtime Component (Universal Windows)
A project for creating a managed Windows Runtime component (.winmd) for Universal Windows Platform (UWP) apps, regardless of the programming languages in which the apps are written.

C# Windows Library UWP

 Coded UI Test Project (Universal Windows) [Deprecated]

Back Next

3. In the configuration window, do the following:

- Name your project. Here, we call it **classifierPyTorch**.
- Choose the location of your project.
- If you're using VS2019, ensure `Create directory for solution` is checked.
- If you're using VS2017, ensure `Place solution and project in the same directory` is unchecked.

Configure your new project

Blank App (Universal Windows) C# XAML Windows Xbox UWP Desktop

Project name

classifierPyTorch

Location

C:\Users\alexzak\Source\Repos



Solution name !

classifierPyTorch

Place solution and project in the same directory

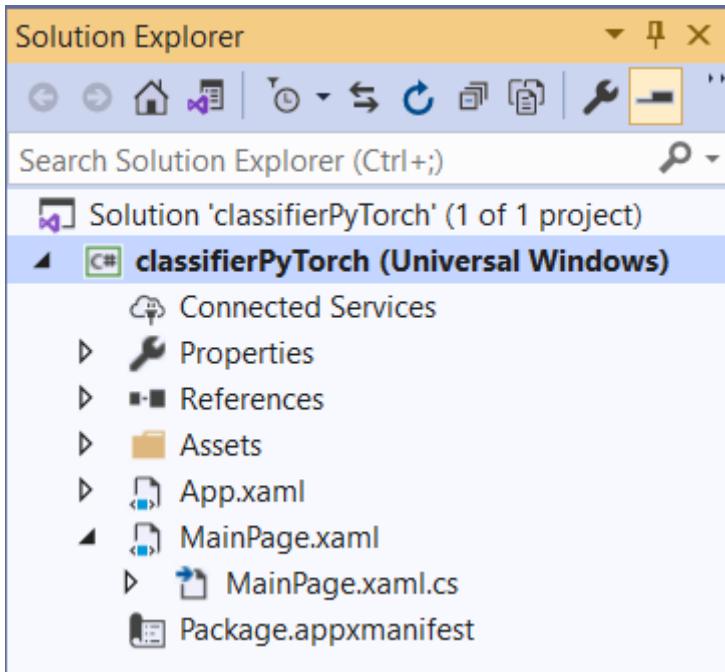
Press `create` to create your project. The minimum target version window may pop up. Be sure your minimum version is set to **Windows 10, version 1809 (10.0; build 17763)** or higher.

4. After the project is created, navigate to the project folder, open the **assets** folder [`....\classifierPyTorch\Assets`], and copy your `ImageClassifier.onnx` file to this location.

Explore project solution

Let's explore your project solution.

Visual Studio automatically created several cs-code files inside the Solution Explorer. `MainPage.xaml` contains the XAML code for your GUI, and `MainPage.xaml.cs` contains your application code, also known as the code-behind. If you've created a UWP app before, these files should be very familiar to you.



Create the Application GUI

First, let's create a simple GUI for your app.

1. Double-click on the `MainPage.xaml` code file. In your blank app, the XAML template for your app's GUI is empty, so we'll need to add some UI features.
2. Add the below code to `MainPage.xaml`, replacing the `<Grid>` and `</Grid>` tags.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <StackPanel Margin="1,0,-1,0">
        <TextBlock x:Name="Menu"
            FontWeight="Bold"
            TextWrapping="Wrap"
            Margin="10,0,0,0"
            Text="Image Classification"/>
        <TextBlock Name="space" />
        <Button Name="recognizeButton"
            Content="Pick Image"
            Click="OpenFileButton_Click"
            Width="110"
            Height="40"
            IsEnabled="True"
            HorizontalAlignment="Left"/>
        <TextBlock Name="space3" />
        <Button Name="Output"
            Content="Result is:"
            Width="110"
            Height="40"
            IsEnabled="True"/>
    </StackPanel>
</Grid>
```

```
        HorizontalAlignment="Left"
        VerticalAlignment="Top">
    </Button>
    <!--Display the Result-->
    <TextBlock Name="displayOutput"
        FontWeight="Bold"
        TextWrapping="Wrap"
        Margin="25,0,0,0"
        Text="" Width="1471" />
    <TextBlock Name="space2" />
    <!--Image preview -->
    <Image Name="UIPreviewImage" Stretch="Uniform" MaxWidth="300"
MaxHeight="300"/>
</StackPanel>
</Grid>
```

Add the model to the project using Windows ML Code Generator (mlgen)

Windows Machine Learning Code Generator, or **mlgen**, is a Visual Studio extension to help you get started using WinML APIs on UWP apps. It generates template code when you add a trained ONNX file into the UWP project.

Windows Machine Learning's code generator **mlgen** creates an interface (for C#, C++/WinRT, and C++/CX) with wrapper classes that call the Windows ML API for you. This allows you to easily load, bind, and evaluate a model in your project. We'll use it in this tutorial to handle many of those functions for us.

Code generator is available for Visual Studio 2017 and later. We recommend using Visual Studio. Please be aware that 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. If you've been following this tutorial from the introduction, you will have already handled this, but if not, you should download for [VS 2019](#) or for [VS 2017](#).

ⓘ Note

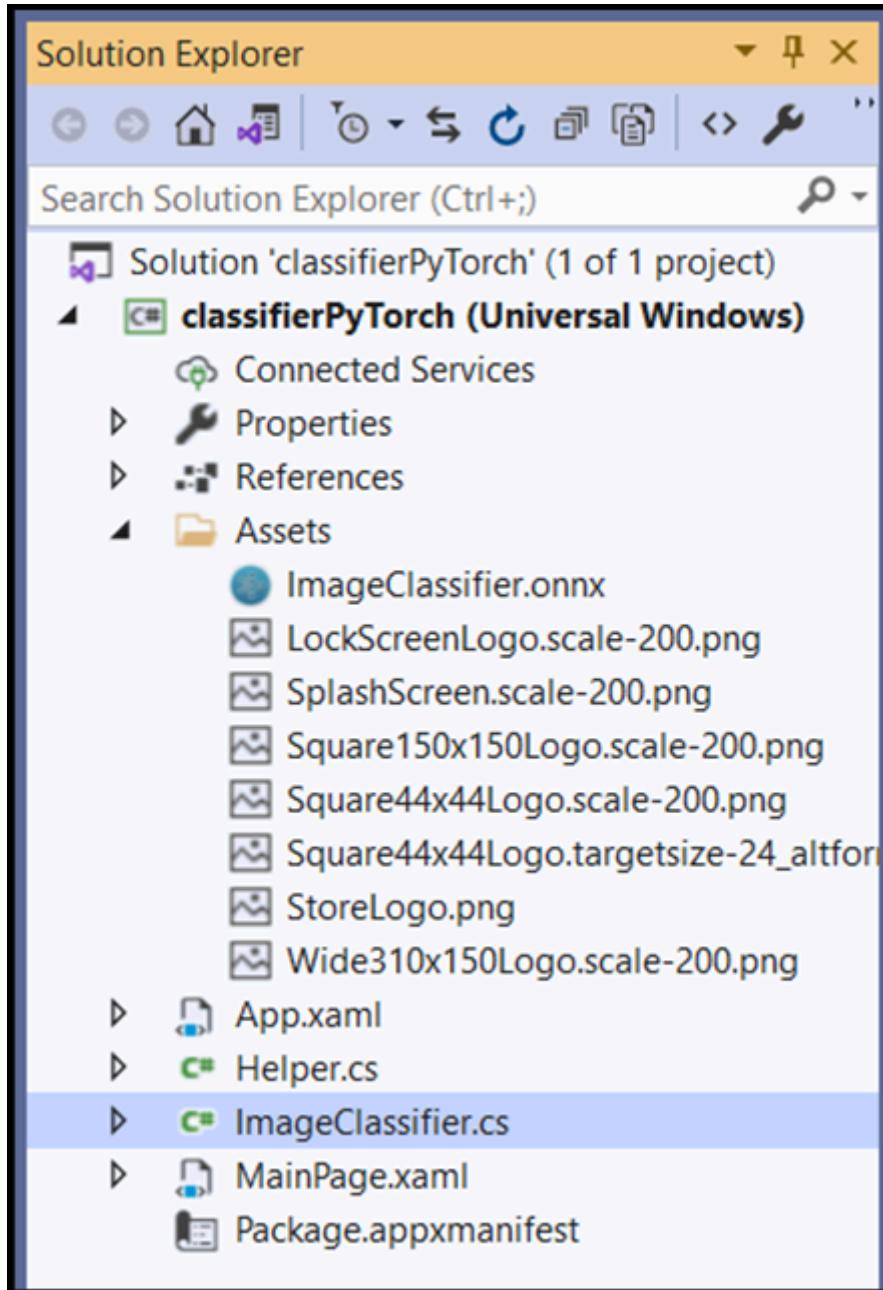
To learn more about **mlgen**, please see the [mlgen documentation](#)

1. If you haven't already, install **mlgen**.
2. Right-click on the **Assets** folder in the Solution Explorer in Visual studio, and select **Add > Existing Item**.

3. Navigate to the assets folder inside `classifierPyTorch` [....\classifierPyTorch \Assets], find the ONNX model you previously copied there, and select `add`.

4. After you added an ONNX model to the assets folder in solution explorer in VS, the project should now have two new files:

- `ImageClassifier.onnx` - this is your model in ONNX format.
- `ImageClassifier.cs` – automatically generated WinML code file.



5. To make sure the model builds when you compile our application, select the `ImageClassifier.onnx` file and choose `Properties`. For `Build Action`, select `Content`.

ONNX file code

Now, let's explore a newly generated code in the `ImageClassifier.cs` file.

The generated code includes three classes:

- `ImageClassifierModel`: This class includes two methods for model instantiation and model evaluation. It will help us to create the machine learning model representation, create a session on the system default device, bind the specific inputs and outputs to the model, and evaluate the model asynchronously.
- `ImageClassifierInput`: This class initializes the input types that the model expects. The model input depends on the model requirements for input data.
- `ImageClassifierOutput`: This class initializes the types that the model will output. The model output depends on how it is defined by the model.

In this tutorial, we don't want to deal with tensorization. We'll make a slight change to the `ImageClassifierInput` class, to change the input data type and make our life easier.

1. Make the following changes in the `ImageClassifier.cs` file:

Change the `input` variable from a `TensorFloat` to an `ImageFeatureValue`.

C#

```
public sealed class ImageClassifierInput
{
    public ImageFeatureValue input; // shape(-1,3,32,32)
}
```

Load the Model

1. Double-click on the `MainPage.xaml.cs` file to open the code-behind for the app.
2. Replace the "using" statements by following, to get an access to all the APIs that you'll need:

C#

```
// Specify all the using statements which give us the access to all the APIs
// that we'll need
using System;
using System.Threading.Tasks;
using Windows.AI.MachineLearning;
using Windows.Graphics.Imaging;
using Windows.Media;
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.Storage.Streams;
```

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Imaging;
```

3. Add the following variable declarations inside your `MainPage` class, above the `public MainPage()` function.

C#

```
// All the required fields declaration
private ImageClassifierModel modelGen;
private ImageClassifierInput image = new ImageClassifierInput();
private ImageClassifierOutput results;
private StorageFile selectedStorageFile;
private string label = "";
private float probability = 0;
private Helper helper = new Helper();

public enum Labels
{
    plane,
    car,
    bird,
    cat,
    deer,
    dog,
    frog,
    horse,
    ship,
    truck
}
```

Now, you'll implement the `LoadModel` method. The method will access the ONNX model and store it in memory. Then, you'll use the `CreateFromStreamAsync` method to instantiate the model as a `LearningModel` object. The `LearningModel` class represents a trained machine learning model. Once instantiated, the `LearningModel` is the initial object you use to interact with Windows ML.

To load the model, you can use several static methods in the `LearningModel` class. In this case, you'll use the `CreateFromStreamAsync` method.

The `CreateFromStreamAsync` method was automatically created with mlgen, so you don't need to implement this method. You can review this method by double clicking on the `classifier.cs` file generated file by mlgen.

 **Note**

To learn more about `LearningModel` class, please review the [LearningModel Class documentation](#). To learn more about additional ways of loading the model, please review the [Load a model documentation](#)

4. Add a call to a `loadModel` method to the constructor of the main class.

C#

```
// The main page to initialize and execute the model.  
public MainPage()  
{  
    this.InitializeComponent();  
    loadModel();  
}
```

5. Add the implementation of the `loadModel` method, inside that `MainPage` class.

C#

```
private async Task loadModel()  
{  
    // Get an access the ONNX model and save it in memory.  
    StorageFile modelFile = await  
StorageFile.GetFileFromApplicationUriAsync(new Uri($"ms-  
appx:///Assets/ImageClassifier.onnx"));  
    // Instantiate the model.  
    modelGen = await  
ImageClassifierModel.CreateFromStreamAsync(modelFile);  
}
```

Load the Image

1. We need to define a click event to initiate the sequence of four method calls for model execution – conversion, binding and evaluation, output extraction and display the results. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
// Waiting for a click event to select a file  
private async void OpenFileButton_Click(object sender,  
RoutedEventArgs e)  
{  
    if (!await getImage())  
    {  
        return;  
    }
```

```

        }
        // After the click event happened and an input selected, begin
        // the model execution.
        // Bind the model input
        await imageBind();
        // Model evaluation
        await evaluate();
        // Extract the results
        extractResult();
        // Display the results
        await displayResult();
    }
}

```

- Now, you'll implement the `getImage()` method. This method will select an input image file and save it in memory. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```

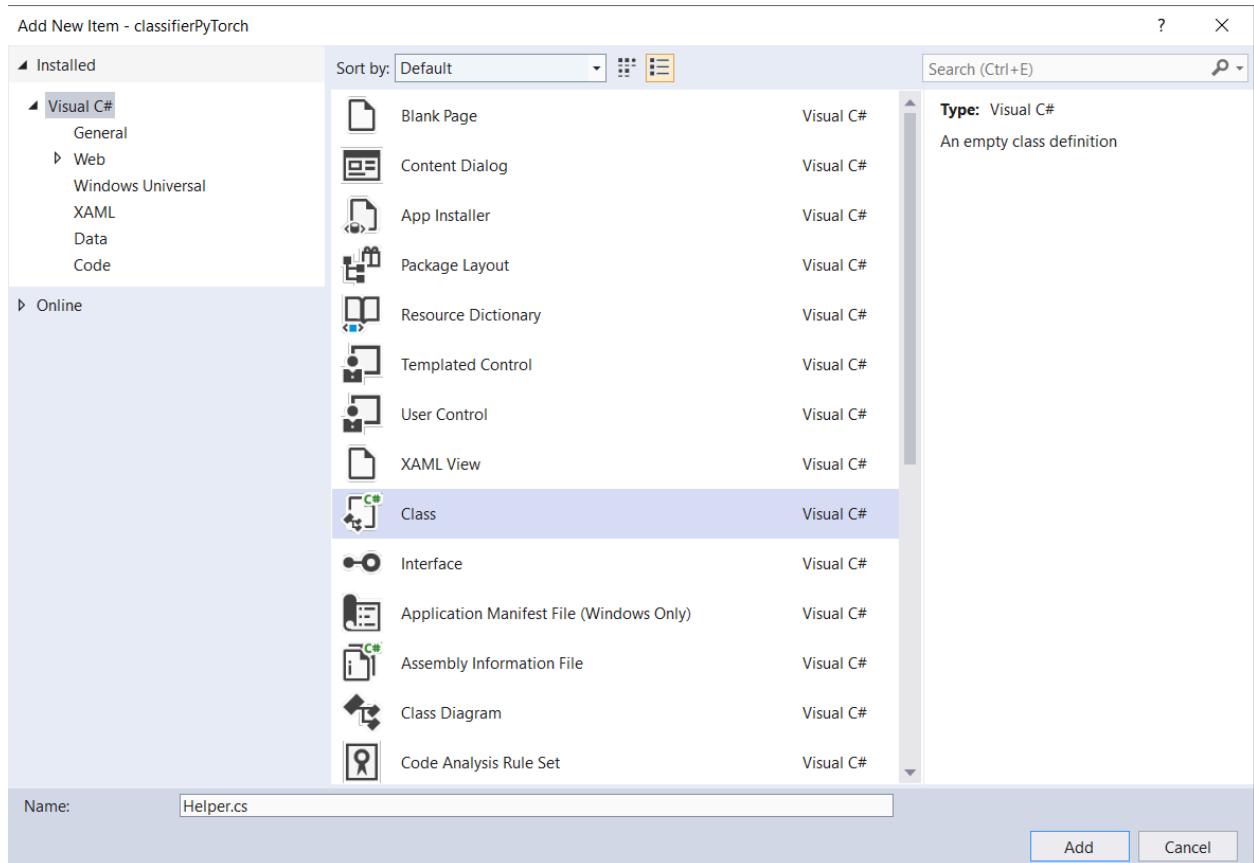
// A method to select an input image file
private async Task<bool> getImage()
{
    try
    {
        // Trigger file picker to select an image file
        FileOpenPicker fileOpenPicker = new FileOpenPicker();
        fileOpenPicker.SuggestedStartLocation =
PickerLocationId.PicturesLibrary;
        fileOpenPicker.FileTypeFilter.Add(".jpg");
        fileOpenPicker.FileTypeFilter.Add(".png");
        fileOpenPicker.ViewMode = PickerViewMode.Thumbnail;
        selectedStorageFile = await
fileOpenPicker.PickSingleFileAsync();
        if (selectedStorageFile == null)
        {
            return false;
        }
    }
    catch (Exception)
    {
        return false;
    }
    return true;
}

```

Next, you will implement an image `Bind()` method to get the representation of the file in the bitmap BGRA8 format. But first, you'll create a helper class to resize the image.

- To create a helper file, right-click on the solution name (`ClassifierPyTorch`), then choose `Add a new item`. In the open window, select `Class` and give it a name.

Here, we're calling it `Helper`.



4. A new class file will appear within your project. Open this class, and add the following code:

```
C#  
  
using System;  
using System.Threading.Tasks;  
using Windows.Graphics.Imaging;  
using Windows.Media;  
  
namespace classifierPyTorch  
{  
    public class Helper  
    {  
        private const int SIZE = 32;  
        VideoFrame cropped_vf = null;  
  
        public async Task<VideoFrame>  
CropAndDisplayInputImageAsync(VideoFrame inputVideoFrame)  
        {  
            bool useDX = inputVideoFrame.SoftwareBitmap == null;  
  
            BitmapBounds cropBounds = new BitmapBounds();  
            uint h = SIZE;  
            uint w = SIZE;  
            var frameHeight = useDX ?  
inputVideoFrame.Direct3DSurface.Description.Height :  
                inputVideoFrame.SoftwareBitmap.Bounds.Height;  
            cropBounds.Width = w;  
            cropBounds.Height = h;  
            cropBounds.Top = (frameHeight - h) / 2;  
            cropBounds.Left = (w - w) / 2;  
            if (useDX)  
                inputVideoFrame.Crop(cropBounds);  
            else  
                inputVideoFrame.SoftwareBitmap.Crop(cropBounds);  
            cropped_vf = inputVideoFrame;
```

```

        inputVideoFrame.SoftwareBitmap.PixelHeight;
        var frameWidth = useDX ?
inputVideoFrame.Direct3DSurface.Description.Width :
inputVideoFrame.SoftwareBitmap.PixelWidth;

        var requiredAR = ((float)SIZE / SIZE);
        w = Math.Min((uint)(requiredAR * frameHeight),
(uint)frameWidth);
        h = Math.Min((uint)(frameWidth / requiredAR),
(uint)frameHeight);
        cropBounds.X = (uint)((frameWidth - w) / 2);
        cropBounds.Y = 0;
        cropBounds.Width = w;
        cropBounds.Height = h;

        cropped_vf = new VideoFrame(BitmapPixelFormat.Bgra8, SIZE, SIZE,
BitmapAlphaMode.Ignore);

        await inputVideoFrame.CopyToAsync(cropped_vf, cropBounds, null);
        return cropped_vf;
    }
}
}

```

Now, let's convert the image to the appropriate format.

The `ImageClassifierInput` class initializes the input types that the model expects. In our case, we configured our code to expect an `ImageFeatureValue`.

The `ImageFeatureValue` class describes the properties of the image used to pass into a model. To create an `ImageFeatureValue`, you use `CreateFromVideoFrame` method. For more specific details of why this is the case and how these classes and methods work, see the [ImageFeatureValue class documentation](#)

ⓘ Note

In this tutorial, we use the `ImageFeatureValue` class instead of a tensor. If Window ML does not support your model's color format, this won't be an option. For an example of how to work with image conversions and tensorization, see the [Custom Tensorization Sample ↗](#).

5. Add the implementation of the `convert()` method to your `MainPage.xaml.cs` code file inside the `MainPage` class. The `convert` method will get us a representation of the input file in a BGRA8 format.

```

// A method to convert and bide the input image.
private async Task imageBind ()
{
    UIPreviewImage.Source = null;
    try
    {
        SoftwareBitmap softwareBitmap;
        using (IRandomAccessStream stream = await
selectedStorageFile.OpenAsync(FileAccessMode.Read))
        {
            // Create the decoder from the stream
            BitmapDecoder decoder = await BitmapDecoder.CreateAsync(stream);
            // Get the SoftwareBitmap representation of the file in BGRA8
format
            softwareBitmap = await decoder.GetSoftwareBitmapAsync();
            softwareBitmap = SoftwareBitmap.Convert(softwareBitmap,
BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied);
        }
        // Display the image
        SoftwareBitmapSource imageSource = new SoftwareBitmapSource();
        await imageSource.SetBitmapAsync(softwareBitmap);
        UIPreviewImage.Source = imageSource;

        // Encapsulate the image within a VideoFrame to be bound and
evaluated
        VideoFrame inputImage =
VideoFrame.CreateWithSoftwareBitmap(softwareBitmap);
        // Resize the image size to 32x32
        inputImage=await helper.CropAndDisplayInputImageAsync(inputImage);
        // Bind the model input with image
        ImageFeatureValue imageTensor =
ImageFeatureValue.CreateFromVideoFrame(inputImage);
        image.modelInput = imageTensor;

        // Encapsulate the image within a VideoFrame to be bound and
evaluated
        VideoFrame inputImage =
VideoFrame.CreateWithSoftwareBitmap(softwareBitmap);
        // bind the input image
        ImageFeatureValue imageTensor =
ImageFeatureValue.CreateFromVideoFrame(inputImage);
        image.modelInput = imageTensor;
    }
    catch (Exception e)
    {
    }
}

```

Bind and Evaluate the model

Next, you'll create a session based on the model, bind the input and output from the session, and evaluate the model.

Create a session to bind the model:

To create a session, you use the `LearningModelSession` class. This class is used to evaluate machine learning models, and binds the model to a device that then runs and evaluates the model. You can select a device when you create a session to execute your model on a specific device of your machine. The default device is the CPU.

ⓘ Note

To learn more about how to choose a device, please review the [Create a session](#) documentation.

Bind model inputs and outputs:

To bind input and output, you use the `LearningModelBinding` class. A machine learning model has input and output features, which pass information into and out of the model. Be aware that required features must be supported by the Window ML APIs. The `LearningModelBinding` class is applied on a `LearningModelSession` to bind values to named input and output features.

The implementation of the binding is automatically generated by mlgen, so you don't have to take care of it. The binding is implemented by calling the predefined methods of the `LearningModelBinding` class. In our case, it uses the `Bind` method to bind a value to the named feature type.

Evaluate the model:

After you create a session to bind the model and bounded values to a model's inputs and outputs, you can evaluate the model's inputs and get its predictions. To run the model execution, you should call any of the predefined evaluate methods on the `LearningModelSession`. In our case, we'll use the `EvaluateAsync` method.

Similar to `CreateFromStreamAsync`, the `EvaluateAsync` method was also automatically generated by WinML Code Generator, so you don't need to implement this method. You can review this method in the `ImageClassifier.cs` file.

The `EvaluateAsync` method will asynchronously evaluate the machine learning model using the feature values already bound in bindings. It will create a session with `LearningModelSession`, bind the input and output with `LearningModelBinding`, execute the model evaluation, and get the output features of the model using the `LearningModelEvaluationResult` class.

ⓘ Note

To learn about another evaluate methods to run the model, please check which methods can be implemented on the `LearningModelSession` by reviewing the [LearningModelSession Class documentation](#).

1. Add the following method to your `MainPage.xaml.cs` code file inside the `MainPage` class to create a session, bind and evaluate the model.

C#

```
// A method to evaluate the model
private async Task evaluate()
{
    results = await modelGen.EvaluateAsync(image);
}
```

Extract and display the results

You'll now need to extract the model output and display the right result, which you'll do by implementing the `extractResult` and `displayResult` methods. You'll need to find the highest probability to return the correct label.

1. Add the `extractResult` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
// A method to extract output from the model
private void extractResult()
{
    // Retrieve the results of evaluation
    var mResult = results.modelOutput as TensorFloat;
    // convert the result to vector format
    var resultVector = mResult.GetAsVectorView();

    probability = 0;
    int index = 0;
```

```
// find the maximum probability
for(int i=0; i<resultVector.Count; i++)
{
    var elementProbability=resultVector[i];
    if (elementProbability > probability)
    {
        index = i;
    }
}
label = ((Labels)index).ToString();
}
```

2. Add the `displayResult` method to your `MainPage.xaml.cs` code file inside the `MainPage` class.

C#

```
private async Task displayResult()
{
    displayOutput.Text = label;
}
```

That's it! You've successfully created the Windows machine learning app with a basic GUI to test our classification model. The next step is to launch the application and run it locally on your Windows device.

Launch the application

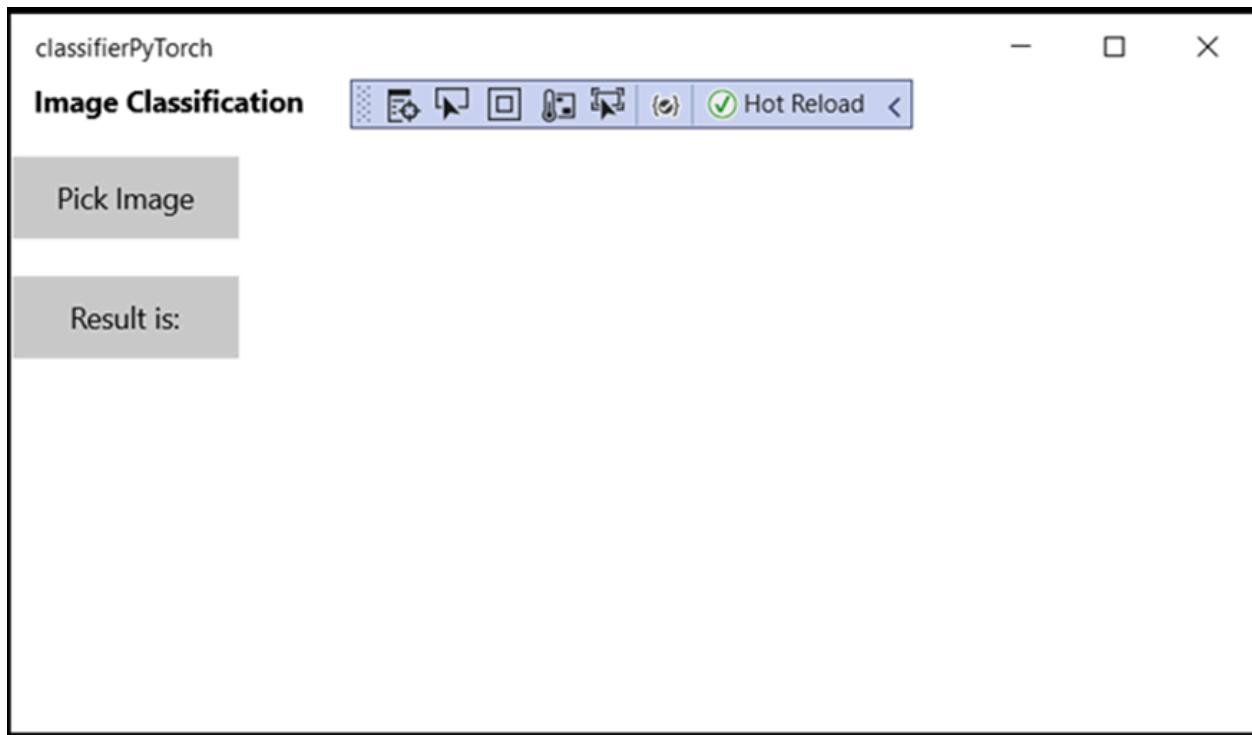
Once you completed the application interface, added the model, and generated the Windows ML code, you can test the application!

Enable developer mode and test your application from Visual Studio. Make sure the dropdown menus in the top toolbar are set to `Debug`. 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.

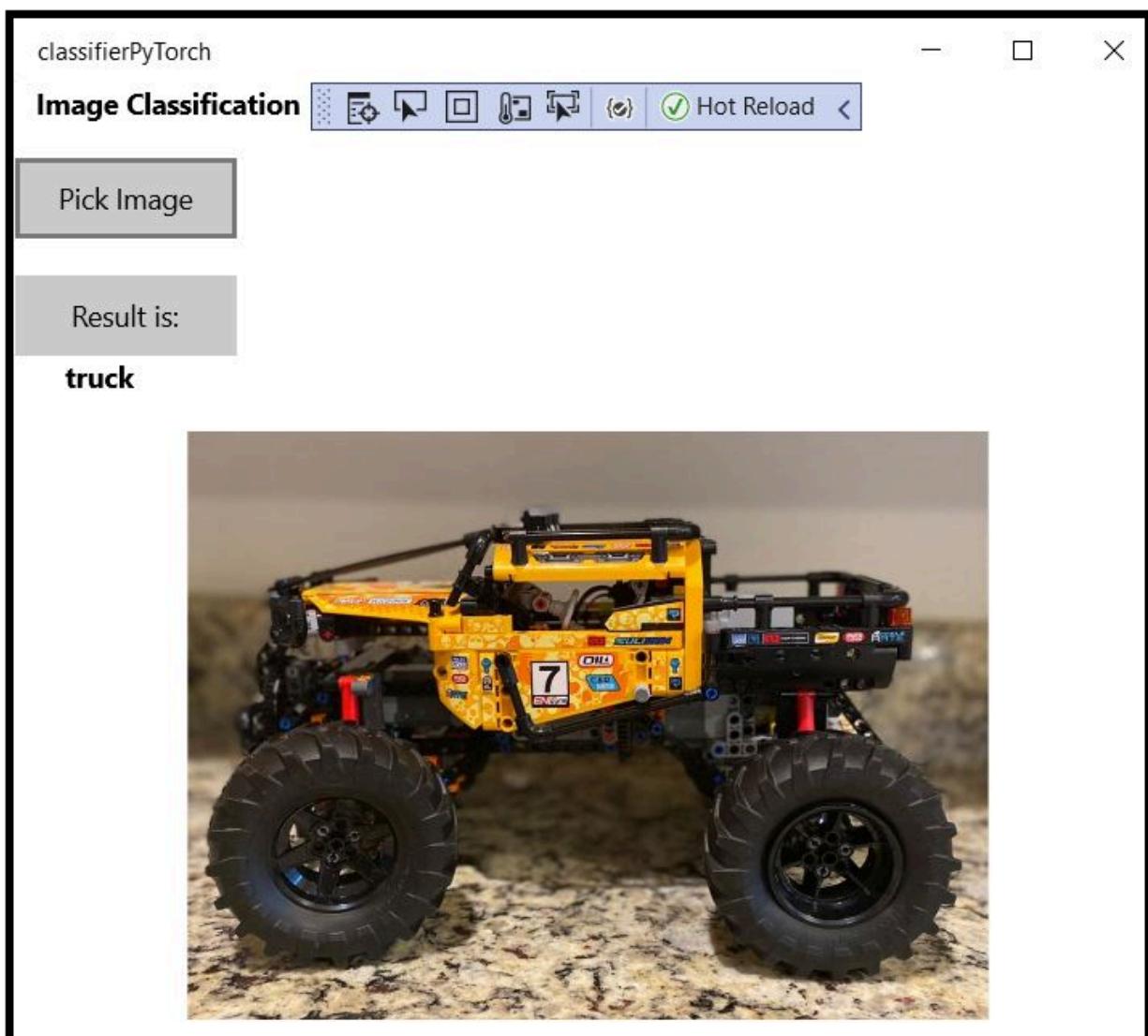
Our model was trained to classify the following images: plane, car, bird, cat, deer, dog, frog, horse, ship, truck. To test our app, you will use the image of the Lego car built for this project. Let's see how our app classifies the content of the image.



1. Save this image on your local device to test the app. Change the image format to `.jpg` if required. You can also any other relevant image from your local device in a `.jpg` or `.png` format.
2. To run the project, select the `Start Debugging` button on the toolbar, or press `F5`.
3. When the application starts, press `Pick Image` and select the image from your local device.



The result will appear on the screen right away. As you can see, our Windows ML app successfully classified the image as a car.



Summary

You've just made your first Windows Machine Learning app, from model creation to successful execution.

Additional Resources

To learn more about topics mentioned in this tutorial, visit the following resources:

- Windows ML tools: Learn more tools like the [Windows ML Dashboard](#), [WinMLRunner](#), and the [mglens](#) Windows ML code generator.
- ONNX model: Learn more about the ONNX format.
- [Windows ML performance and memory](#): Learn more how to manage app performance with Windows ML.
- [Windows Machine Learning API reference](#): Learn more about three areas of Windows ML APIs.

Feedback

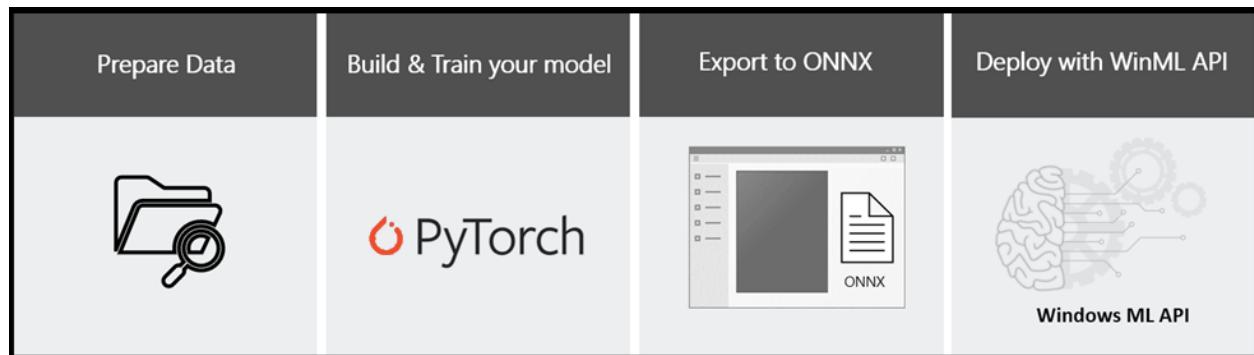
Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Data analysis with PyTorch and Windows ML

Article • 12/30/2021



Windows Machine Learning can be used to run predictions on tabular datasets, predicting numeric values based on independent input variables. This guide uses a specific dataset in Excel format, but the procedures outlined will work for any related task using a tabular dataset of your choice.

This guide will show you how to solve a classification task with a neural network using the PyTorch library, export the model to the ONNX format, and deploy it in a Windows Machine Learning application running locally on your Windows device.

Basic knowledge in Python and C# programming languages is required. Previous experience in machine learning is preferable but not required.

If you'd like to move straight on to installation, see [Install PyTorch](#).

If you've set up PyTorch already, start the model training process by [getting the data](#).

Once you're ready to go with the data, you can start to [train your model](#), and then [convert it to the ONNX format](#).

If you have an ONNX model and want to learn how to create a WinML app from scratch, navigate to [deploy your model](#).

ⓘ Note

If you want, you can clone the Windows Machine Learning samples repo and run the completed code for this tutorial. You can find the [PyTorch training solution here](#), or the [completed Windows ML app here](#). If you're using the PyTorch file, make sure you set up the relevant PyTorch interpreter before you run it.

Scenario

In this tutorial, we'll create a machine learning data analysis application to predict the type of iris flowers. For this purpose, you will use Fisher's iris flower dataset. The model will be trained to recognize certain types of iris patterns and predict the correct type.

Prerequisites for PyTorch - model training:

PyTorch is supported on the following Windows distributions:

- Windows 7 and greater. Windows 10 or greater recommended.
- Windows Server 2008 r2 and greater

To use Pytorch on Windows, you must have Python 3.x installed. Python 2.x is not supported.

Prerequisites for Windows ML app deployment

To create and deploy a WinML app, you'll need the following:

- Windows 10 version 1809 (build 17763) or higher. You can check your build version number by running `winver` via the Run command (`Windows logo key + R`).
- Windows SDK for build 17763 or higher. [You can get the SDK here](#).
- Visual Studio 2017 version 15.7 or later. We recommend using Visual Studio 2019, and some screenshots in this tutorial may be different if you use VS2017 instead. [You can get Visual Studio here](#).
- You'll also need to [enable Developer Mode on your PC](#)

ⓘ Note

Windows ML APIs are built into the latest versions of Windows 10 (1809 or higher) and Windows Server 2019. If your target platform is older versions of Windows, you can port your WinML app to the redistributable NuGet package (Windows 8.1 or higher).

Next Steps

We'll start by [installing PyTorch and configuring our environment](#)

ⓘ Important

PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.

Install and configure PyTorch

Article • 12/30/2021

In the [previous stage of this tutorial](#), we discussed the basics of PyTorch and the prerequisites of using it to create a machine learning model. Here, we'll install it on your machine.

Get PyTorch

First, you'll need to setup a Python environment.

We recommend setting up a virtual Python environment inside Windows, using Anaconda as a package manager. The rest of this setup assumes you use an Anaconda environment.

1. [Download and install Anaconda here](#). Select `Anaconda 64-bit installer for Windows Python 3.8`.

ⓘ Important

Be aware to install Python 3.x. Currently, PyTorch on Windows only supports Python 3.x; Python 2.x is not supported.

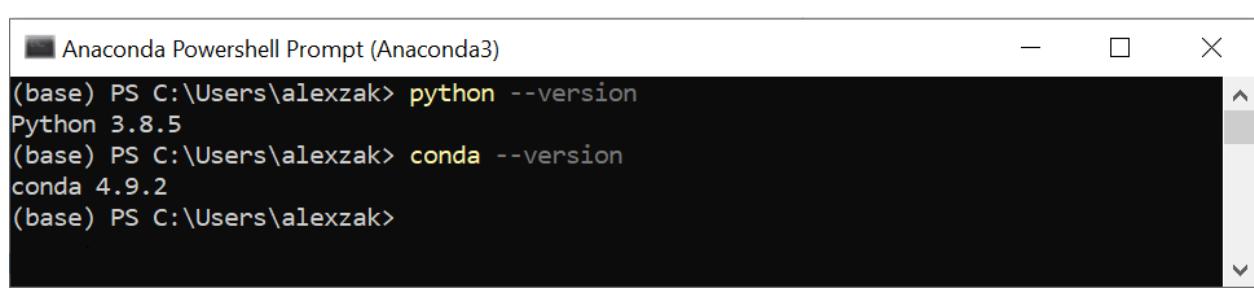


After the installation is complete, verify your Anaconda and Python versions.

2. Open Anaconda manager via Start - Anaconda3 - Anaconda PowerShell Prompt and test your versions:

You can check your Python version by running the following command: `python --version`

You can check your Anaconda version by running the following command: `conda --version`



```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\alexzak> python --version
Python 3.8.5
(base) PS C:\Users\alexzak> conda --version
conda 4.9.2
(base) PS C:\Users\alexzak>
```

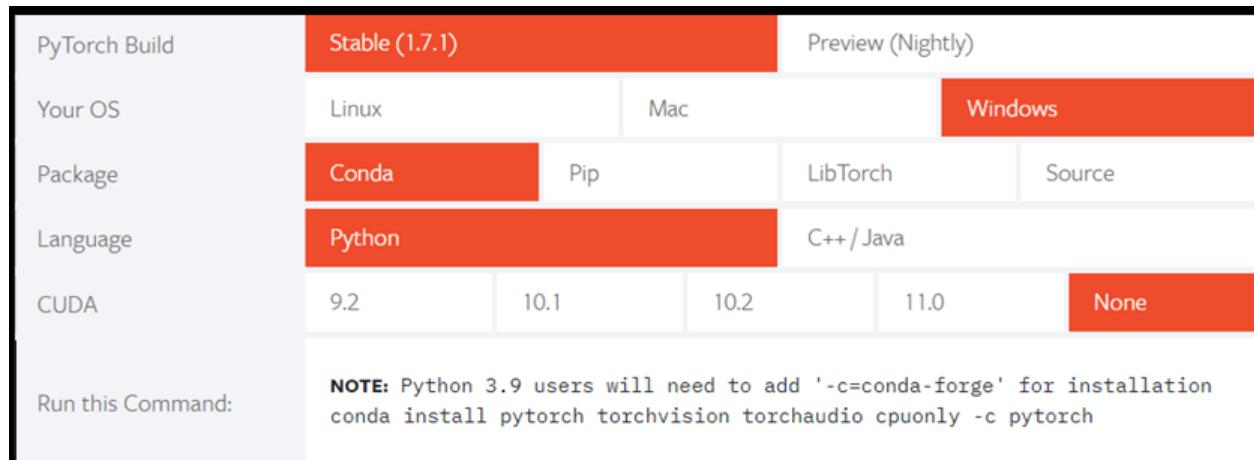
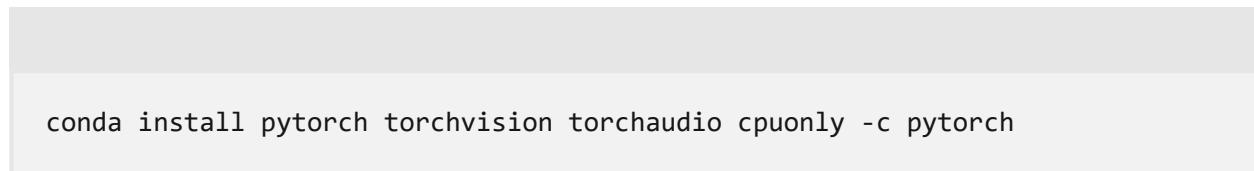
Now, you can install PyTorch package from binaries via Conda.

3. Navigate to <https://pytorch.org/>.

Select the relevant PyTorch installation details:

- PyTorch build – stable.
- Your OS – Windows
- Package – Conda
- Language – Python
- Compute Platform – CPU, or choose your version of Cuda. In this tutorial, you will train and inference model on CPU, but you could use a Nvidia GPU as well.

4. Open Anaconda manager and run the command as it specified in the installation instructions.



5. Confirm and complete the extraction of the required packages.

```
■ Anaconda Powershell Prompt (Anaconda3)
Proceed ([y]/n)? y

Downloading and Extracting Packages
pytorch-1.7.1      | 1007.0 MB | ######| 100%
#####
torchaudio-0.7.2   | 2.7 MB    | #####| 100%
#####
libuv-1.40.0       | 255 KB   | #####| 100%
#####
torchvision-0.8.2  | 7.3 MB   | #####| 100%
#####
ninja-1.10.2       | 247 KB   | #####| 100%
#####
cudatoolkit-11.0.221 | 627.0 MB | #####| 100%
#####
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) PS C:\Users\alexzak>
```

Let's verify PyTorch installation by running sample PyTorch code to construct a randomly initialized tensor.

6. Open the Anaconda PowerShell Prompt and run the following command.

```
python
```

Next, enter the following code:

```
import torch

x = torch.rand(2, 3)

print(x)
```

The output should be a random 5x3 tensor. The numbers will be different, but it should look similar to the below.

```
■ Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\alexzak> python
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> x = torch.rand(2,3)
>>> print(x)
tensor([[0.5343, 0.7362, 0.8390],
       [0.4758, 0.2309, 0.1936]])
>>>
```

ⓘ Note

Interested in learning more? Visit the [PyTorch official website](#) ↗

Next Steps

Now that we've installed PyTorch, we're ready to [set up the data for our model](#).

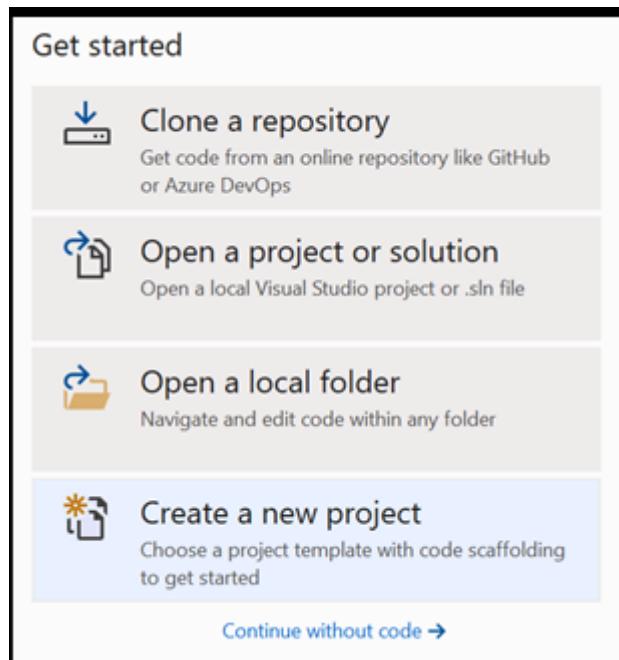
Prepare the data for analysis

Article • 12/30/2021

In the [previous stage of this tutorial](#), we installed PyTorch on your machine. Now, we'll use it to set up our code with the data we'll use to make our model.

Open a new project within Visual Studio.

1. Open Visual Studio and choose `create a new project`.



2. In the search bar, type `Python` and select `Python Application` as your project template.

python X ▾

Clear all

Python Windows All project types

Python Application
A project for creating a command-line application
Python Windows Linux macOS Console

From Existing Python code
Create a new project using code files that are already in a folder hierarchy
Python Linux macOS Windows Console Web

Web Project
A project for creating a generic Python web project
Python Windows Linux macOS Web

Django Web Project
A project for creating an application using the Django web framework. It features sample pages that use the Twitter Bootstrap framework for responsive web design.
Python Windows Linux macOS Web

Flask Web Project
A project for creating an application using the Flask web framework with the Jinja template engine. It features sample pages that use the Twitter Bootstrap framework for responsive web design.

Back Next

3. In the configuration window:

- Name your project. Here, we call it **DataClassifier**.
- Choose the location of your project.
- If you're using VS2019, ensure `Create directory for solution` is checked.
- If you're using VS2017, ensure `Place solution and project in the same directory` is unchecked.

Configure your new project

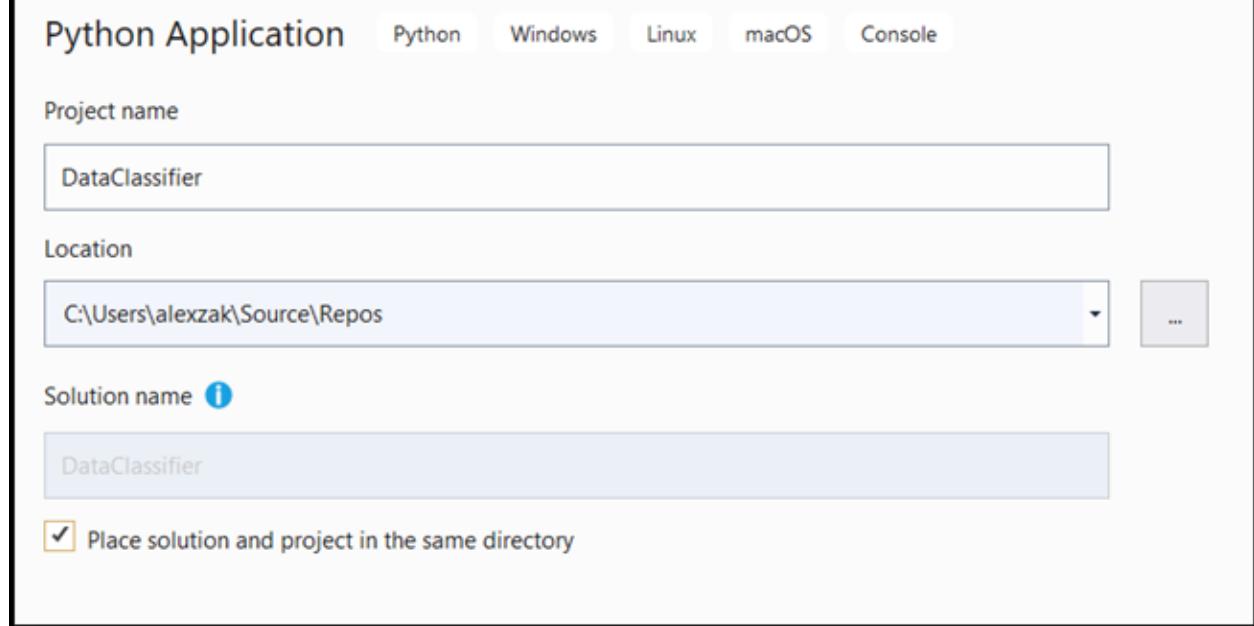
Python Application Python Windows Linux macOS Console

Project name
DataClassifier

Location
C:\Users\alexzak\Source\Repos

Solution name ⓘ
DataClassifier

Place solution and project in the same directory

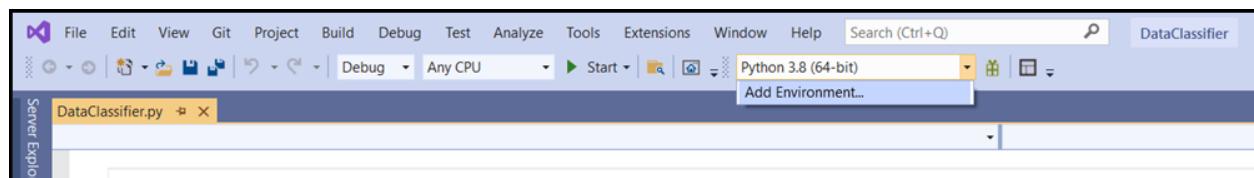


Press `create` to create your project.

Create a Python interpreter

Now, you need to define a new Python interpreter. This must include the PyTorch package you've recently installed.

1. Navigate to interpreter selection, and select `Add Environment`:



2. In the `Add Environment` window, select `Existing environment`, and choose `Anaconda3 (3.6, 64-bit)`. This includes the PyTorch package.

Add environment

Virtual environment	Project
Conda environment	<input type="text" value="DataClassifier"/>
Existing environment	Environment
Python installation	<input type="text" value="Anaconda3 (3.8, 64-bit)"/>

To test the new Python interpreter and PyTorch package, enter the following code to the `DataClassifier.py` file:

```
from __future__ import print_function

import torch

x=torch.rand(2, 3)

print(x)
```

The output should be a random 5x3 tensor similar to the below.

```
tensor([[0.1563, 0.6212, 0.2504],
        [0.4116, 0.2615, 0.6808]])
Press any key to continue . . .
```

ⓘ Note

Interested in learning more? Visit the [PyTorch official website](#).

Understanding the data

We will train the model on the Fisher's Iris flower dataset. This famous dataset includes 50 records for each of three Iris species: Iris setosa, Iris virginica, and Iris versicolor.

Several versions of the dataset have been published. You can find Iris dataset at the [UCI Machine Learning Repository](#), import the dataset directly from the [Python Scikit-learn](#)

library [🔗](#) or use any other version previously published. To learn about Iris flower data set, visit [its Wikipedia page](#) [🔗](#).

In this tutorial, to showcase how to train the model with the tabular type of input, you will use the Iris dataset exported to the Excel file.

Each line of the excel table will show four features of Irises: sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. These features will serve as your input. The last column includes the Iris type related to these parameters and will represent the regression output. In total, the dataset includes 150 inputs of four features, each of them matched to the relevant Iris type.

#	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	Iris_Type
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

Regression analysis looks at the relationship between input variables and the outcome. Based on the input, the model will learn to predict the correct type of output - one of the three Iris types: Iris-setosa, Iris-versicolor, Iris-virginica.

ⓘ Important

If you decide to use any other dataset to create your own model, you will need to specify your model input variables and output according to your scenario.

Load the dataset.

1. Download the Iris dataset in Excel format. [You can find it here](#) [🔗](#).
2. In the `DataClassifier.py` file in the **Solution Explorer Files** folder, add the following import statement to get access to all the packages that we'll need.

```
py

import torch
import pandas as pd
import torch.nn as nn
from torch.utils.data import random_split, DataLoader, TensorDataset
import torch.nn.functional as F
import numpy as np
```

```
import torch.optim as optim
from torch.optim import Adam
```

As you can see, you will be using pandas (Python data analysis) package to load and manipulate data and torch.nn package that contains modules and extensible classes for building neural networks.

3. Load the data into memory and verify the number of classes. We expect to see 50 items of each Iris type. Be sure to specify the location of the dataset on your PC.

Add the following code to the `DataClassifier.py` file.

```
py

# Loading the Data
df = pd.read_excel(r'C:\Iris_dataset.xlsx')
print('Take a look at sample from the dataset:')
print(df.head())

# Let's verify if our data is balanced and what types of species we have
print('\nOur dataset is balanced and has the following values to predict:')
print(df['Iris_Type'].value_counts())
```

When we run this code, the expected output is as follows:

```
Take a look at sample from the dataset:
   #  sepal length in cm  ...  petal width in cm  Iris_Type
0    1                  5.1  ...                  0.2  Iris-setosa
1    2                  4.9  ...                  0.2  Iris-setosa
2    3                  4.7  ...                  0.2  Iris-setosa
3    4                  4.6  ...                  0.2  Iris-setosa
4    5                  5.0  ...                  0.2  Iris-setosa

[5 rows x 6 columns]

Our dataset is balanced and has the following values to predict:
Iris-setosa      50
Iris-virginica  50
Iris-versicolor 50
Name: Iris_Type, dtype: int64
```

To be able to use the dataset and train the model, we need to define input and output. The input includes 150 lines of features, and the output is the Iris type column. The neural network we'll use requires numeric variables, so you'll convert the output variable to a numeric format.

4. Create a new column in the dataset which will represent the output in a numeric format and define a regression input and output.

Add the following code to the `DataClassifier.py` file.

```
py

# Convert Iris species into numeric types: Iris-setosa=0, Iris-versicolor=1,
Iris-virginica=2.
labels = {'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica':2}
df['IrisType_num'] = df['Iris_Type'] # Create a new column "IrisType_num"
df.IrisType_num = [labels[item] for item in df.IrisType_num] # Convert the
values to numeric ones

# Define input and output datasets
input = df.iloc[:, 1:-2] # We drop the first column and the two
last ones.
print('\nInput values are:')
print(input.head())
output = df.loc[:, 'IrisType_num'] # Output Y is the last column
print('\nThe output value is:')
print(output.head())
```

When we run this code, the expected output is as follows:

```
Input values are:
   sepal length in cm  sepal width in cm  petal length in cm  petal width in cm
0              5.1          3.5            1.4           0.2
1              6.3          3.3            6.0           2.5
2              7.2          3.6            6.1           2.5
3              6.7          3.3            5.7           2.5
4              5.8          2.8            5.1           2.4

The output value is:
0    0
1    2
2    2
3    2
4    2
```

To train the model, we need to convert the model input and output to the Tensor format:

5. Convert to Tensor:

Add the following code to the `DataClassifier.py` file.

```
py

# Convert Input and Output data to Tensors and create a TensorDataset
input = torch.Tensor(input.to_numpy()) # Create tensor of type
torch.float32
```

```

print('\nInput format: ', input.shape, input.dtype)      # Input format:
torch.Size([150, 4]) torch.float32
output = torch.tensor(output.to_numpy())                 # Create tensor type
torch.int64
print('Output format: ', output.shape, output.dtype)   # Output format:
torch.Size([150]) torch.int64
data = TensorDataset(input, output)        # Create a
torch.utils.data.TensorDataset object for further data manipulation

```

If we run the code, the expected output will show the input and output format, as follow:

```

Input format: torch.Size([150, 4]) torch.float32
Output format: torch.Size([150]) torch.int64

```

There are 150 input values. About 60% will be the model training data. You will keep 20% for validation and 30% for a test.

In this tutorial, the batch size for a training dataset is defined as 10. There are 95 items in the training set, which means that on average, there are 9 full batches to iterate through the training set once (one epoch). You will keep the batch size of the validation and test sets as 1.

6. Split the data to train, validate and test sets:

Add the following code to the `DataClassifier.py` file.

```

py

# Split to Train, Validate and Test sets using random_split
train_batch_size = 10
number_rows = len(input)      # The size of our dataset or the number of rows
in excel table.
test_split = int(number_rows*0.3)
validate_split = int(number_rows*0.2)
train_split = number_rows - test_split - validate_split
train_set, validate_set, test_set = random_split(
    data, [train_split, validate_split, test_split])

# Create Dataloader to read the data within batch sizes and put into memory.
train_loader = DataLoader(train_set, batch_size = train_batch_size, shuffle
= True)
validate_loader = DataLoader(validate_set, batch_size = 1)
test_loader = DataLoader(test_set, batch_size = 1)

```

Next Steps

With the data ready to go, it's time to [train our PyTorch model](#)

Train your data analysis model with PyTorch

Article • 06/22/2022

In the [previous stage of this tutorial](#), we acquired the dataset we'll use to train our data analysis model with PyTorch. Now, it's time to put that data to use.

To train the data analysis model with PyTorch, you need to complete the following steps:

1. Load the data. If you've done the previous step of this tutorial, you've handled this already.
2. Define a neural network.
3. Define a loss function.
4. Train the model on the training data.
5. Test the network on the test data.

Define a neural network

In this tutorial, you'll build a basic neural network model with three linear layers. The structure of the model is as follows:

Linear -> ReLU -> Linear -> ReLU -> Linear

A Linear layer applies a linear transformation to the incoming data. You have to specify the number of input features and the number of output features which should correspond to the number of classes.

A ReLU layer is an activation function to define all incoming features to be 0 or greater. Thus, when a ReLU layer is applied, any number less than 0 is changed to zero, while others are kept the same. We'll apply the activation layer on the two hidden layers, and no activation on the last linear layer.

Model parameters

Model parameters depend on our goal and the training data. The input size depends on the number of features we feed the model – four in our case. The output size is three since there are three possible types of Irises.

Having three linear layers, $(4, 24) \rightarrow (24, 24) \rightarrow (24, 3)$, the network will have 744 weights ($96+576+72$).

The learning rate (lr) sets the control of how much you are adjusting the weights of our network with respect to the loss gradient. The lower it is, the slower the training will be. You'll set lr to 0.01 in this tutorial.

How does the Network work?

Here, we're building a feed-forward network. During the training process, the network will process the input through all the layers, compute the loss to understand how far the predicted label of the image is falling from the correct one, and propagate the gradients back into the network to update the weights of the layers. By iterating over a huge dataset of inputs, the network will "learn" to set its weights to achieve the best results.

A *forward* function computes the value of the loss function, and a *backward* function computes the gradients of the learnable parameters. When you create our neural network with PyTorch, you only need to define the forward function. The backward function will be automatically defined.

1. Copy the following code into the `DataClassifier.py` file in Visual Studio to define the model parameters and the neural network.

```
py
```

```
# Define model parameters
input_size = list(input.shape)[1]    # = 4. The input depends on how many
features we initially feed the model. In our case, there are 4 features for
every predict value
learning_rate = 0.01
output_size = len(labels)           # The output is prediction results for
three types of Irises.

# Define neural network
class Network(nn.Module):
    def __init__(self, input_size, output_size):
        super(Network, self).__init__()

        self.layer1 = nn.Linear(input_size, 24)
        self.layer2 = nn.Linear(24, 24)
        self.layer3 = nn.Linear(24, output_size)

    def forward(self, x):
        x1 = F.relu(self.layer1(x))
        x2 = F.relu(self.layer2(x1))
        x3 = self.layer3(x2)
        return x3
```

```
# Instantiate the model
model = Network(input_size, output_size)
```

You'll also need to define the execution device based on the available one on your PC. PyTorch does not have a dedicated library for GPU, but you can manually define the execution device. The device will be an Nvidia GPU if exists on your machine, or your CPU if one does not.

2. Copy the following code to define the execution device:

```
py

# Define your execution device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("The model will be running on", device, "device\n")
model.to(device)      # Convert model parameters and buffers to CPU or Cuda
```

3. As the last step, define a function to save the model:

```
py

# Function to save the model
def saveModel():
    path = "./NetModel.pth"
    torch.save(model.state_dict(), path)
```

ⓘ Note

Interested in learning more about neural network with PyTorch? Check out the [PyTorch documentation ↗](#).

Define a loss function

A loss function computes a value that estimates how far away the output is from the target. The main objective is to reduce the loss function's value by changing the weight vector values through backpropagation in neural networks.

Loss value is different from model accuracy. The loss function represents how well our model behaves after each iteration of optimization on the training set. The accuracy of the model is calculated on the test data, and shows the percentage of predictions that are correct.

In PyTorch, the neural network package contains various loss functions that form the building blocks of deep neural networks. If you want to learn more of these specifics, get started with the above note. Here, we'll use the existing functions optimized for classification like this, and use a Classification Cross-Entropy loss function and an Adam optimizer. In the optimizer, learning rate (lr) sets the control of how much you are adjusting the weights of our network with respect the loss gradient. You'll set it as 0.001 here - the lower it is, the slower the training will be.

1. Copy the following code into the `DataClassifier.py` file in Visual Studio to define the loss function and an optimizer.

```
py
```

```
# Define the loss function with Classification Cross-Entropy loss and an
# optimizer with Adam optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
```

Train the model on the training data.

To train the model, you have to loop over our data iterator, feed the inputs to the network, and optimize. To validate the results, you simply compare the predicted labels to the actual labels in the validation dataset after every training epoch.

The program will display the training loss, validation loss and the accuracy of the model for every epoch or for every complete iteration over the training set. It will save the model with the highest accuracy, and after 10 epochs, the program will display the final accuracy.

1. Add the following code to the `DataClassifier.py` file

```
py
```

```
# Training Function
def train(num_epochs):
    best_accuracy = 0.0

    print("Begin training...")
    for epoch in range(1, num_epochs+1):
        running_train_loss = 0.0
        running_accuracy = 0.0
        running_vall_loss = 0.0
        total = 0

        # Training Loop
        for data in train_loader:
```

```

        #for data in enumerate(train_loader, 0):
            inputs, outputs = data # get the input and real species as
outputs; data is a list of [inputs, outputs]
            optimizer.zero_grad() # zero the parameter gradients
            predicted_outputs = model(inputs) # predict output from the
model
            train_loss = loss_fn(predicted_outputs, outputs) # calculate
loss for the predicted output
            train_loss.backward() # backpropagate the loss
            optimizer.step() # adjust parameters based on the
calculated gradients
            running_train_loss +=train_loss.item() # track the loss value

        # Calculate training loss value
        train_loss_value = running_train_loss/len(train_loader)

        # Validation Loop
        with torch.no_grad():
            model.eval()
            for data in validate_loader:
                inputs, outputs = data
                predicted_outputs = model(inputs)
                val_loss = loss_fn(predicted_outputs, outputs)

                # The label with the highest value will be our prediction
                _, predicted = torch.max(predicted_outputs, 1)
                running_val_loss += val_loss.item()
                total += outputs.size(0)
                running_accuracy += (predicted == outputs).sum().item()

        # Calculate validation loss value
        val_loss_value = running_val_loss/len(validate_loader)

        # Calculate accuracy as the number of correct predictions in the
validation batch divided by the total number of predictions done.
        accuracy = (100 * running_accuracy / total)

        # Save the model if the accuracy is the best
        if accuracy > best_accuracy:
            saveModel()
            best_accuracy = accuracy

        # Print the statistics of the epoch
        print('Completed training batch', epoch, 'Training Loss is: %.4f'
%train_loss_value, 'Validation Loss is: %.4f' %val_loss_value, 'Accuracy is
%d %%' % (accuracy))

```

Test the model on the test data.

Now that we've trained the model, we can test the model with the test dataset.

We'll add two test functions. The first tests the model you saved in the previous part. It will test the model with the test data set of 45 items, and print the accuracy of the model. The second is an optional function to test the model's confidence in predicting each of the three iris species, represented by the probability of successful classification of each species.

1. Add the following code to the `DataClassifier.py` file.

```
py

# Function to test the model
def test():
    # Load the model that we saved at the end of the training loop
    model = Network(input_size, output_size)
    path = "NetModel.pth"
    model.load_state_dict(torch.load(path))

    running_accuracy = 0
    total = 0

    with torch.no_grad():
        for data in test_loader:
            inputs, outputs = data
            outputs = outputs.to(torch.float32)
            predicted_outputs = model(inputs)
            _, predicted = torch.max(predicted_outputs, 1)
            total += outputs.size(0)
            running_accuracy += (predicted == outputs).sum().item()

    print('Accuracy of the model based on the test set of', test_split
, 'inputs is: %d %%' % (100 * running_accuracy / total))

# Optional: Function to test which species were easier to predict
def test_species():
    # Load the model that we saved at the end of the training loop
    model = Network(input_size, output_size)
    path = "NetModel.pth"
    model.load_state_dict(torch.load(path))

    labels_length = len(labels) # how many labels of Irises we have. = 3 in
our database.
    labels_correct = list(0. for i in range(labels_length)) # list to
calculate correct labels [how many correct setosa, how many correct
versicolor, how many correct virginica]
    labels_total = list(0. for i in range(labels_length)) # list to keep
the total # of labels per type [total setosa, total versicolor, total
virginica]

    with torch.no_grad():
        for data in test_loader:
            inputs, outputs = data
```

```

predicted_outputs = model(inputs)
_, predicted = torch.max(predicted_outputs, 1)

label_correct_running = (predicted == outputs).squeeze()
label = outputs[0]
if label_correct_running.item():
    labels_correct[label] += 1
labels_total[label] += 1

label_list = list(labels.keys())
for i in range(output_size):
    print('Accuracy to predict %s : %2d %%' % (label_list[i], 100 *
labels_correct[i] / labels_total[i]))

```

Finally, let's add the main code. This will initiate model training, save the model, and display the results on the screen. We'll run only two iterations [`num_epochs = 25`] over the training set, so the training process won't take too long.

2. Add the following code to the `DataClassifier.py` file.

```

py

if __name__ == "__main__":
    num_epochs = 10
    train(num_epochs)
    print('Finished Training\n')
    test()
    test_species()

```

Let's run the test! Make sure the dropdown menus in the top toolbar are set to `Debug`. 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.

3. To run the project, click the `Start Debugging` button on the toolbar, or press `F5`.

The console window will pop up and you'll see the process of training. As you defined, the loss value will be printed every epoch. The expectation is that the loss value decreases with every loop.

Once the training is complete, you should expect to see the output similar to the below. Your numbers won't be exactly the same - training depends on many factors, and won't always return identical results - but they should look similar.

```

C:\Users\alexzak\Anaconda3\python.exe
Take a look at sample from the dataset:
   # sepal length in cm  ...  petal width in cm      Iris_Type
0    5.1  ...          0.2    Iris-setosa
1  101  ...          2.5  Iris-virginica
2  110  ...          2.5  Iris-virginica
3  145  ...          2.5  Iris-virginica
4  115  ...          2.4  Iris-virginica

[5 rows x 6 columns]

Our dataset is balanced and has the following values to predict:
Iris-versicolor    50
Iris-virginica    50
Iris-setosa        50
Name: Iris_Type, dtype: int64

Input values are:
   sepal length in cm  sepal width in cm  petal length in cm  petal width in cm
0            5.1           3.5             1.4            0.2
1            6.3           3.3             6.0            2.5
2            7.2           3.6             6.1            2.5
3            6.7           3.3             5.7            2.5
4            5.8           2.8             5.1            2.4

The output value is:
0    0
1    2
2    2
3    2
4    2
Name: IrisType_num, dtype: int64

Input format: torch.Size([150, 4]) torch.float32
Output format: torch.Size([150]) torch.int64
The model will be running on cpu device

Begin training...
Completed training batch 1 Training Loss is: 1.0394 Validation Loss is: 1.0383 Accuracy is 60 %
Completed training batch 2 Training Loss is: 0.7123 Validation Loss is: 0.6312 Accuracy is 60 %
Completed training batch 3 Training Loss is: 0.4540 Validation Loss is: 0.4675 Accuracy is 70 %
Completed training batch 4 Training Loss is: 0.3307 Validation Loss is: 0.3576 Accuracy is 90 %
Completed training batch 5 Training Loss is: 0.2595 Validation Loss is: 0.2619 Accuracy is 96 %
Completed training batch 6 Training Loss is: 0.1667 Validation Loss is: 0.2804 Accuracy is 86 %
Completed training batch 7 Training Loss is: 0.1559 Validation Loss is: 0.1678 Accuracy is 93 %
Completed training batch 8 Training Loss is: 0.1074 Validation Loss is: 0.2720 Accuracy is 83 %
Completed training batch 9 Training Loss is: 0.1463 Validation Loss is: 0.1253 Accuracy is 96 %
Completed training batch 10 Training Loss is: 0.0755 Validation Loss is: 0.0999 Accuracy is 100 %
Finished Training

Accuracy of the model based on the test set of 45 inputs is: 95 %
Accuracy to predict Iris-setosa : 100 %
Accuracy to predict Iris-versicolor : 94 %
Accuracy to predict Iris-virginica : 93 %

Model has been converted to ONNX
Press any key to continue . .

```

Next Steps

Now that we have a classification model, the next step is to [convert the model to the ONNX format](#).

Convert your PyTorch model to ONNX format

Article • 06/22/2022

In the [previous stage of this tutorial](#), we used PyTorch to create our machine learning model. However, that model is a `.pth` file. To be able to integrate it with Windows ML app, you'll need to convert the model to ONNX format.

Export the model

To export a model, you will use the `torch.onnx.export()` function. This function executes the model, and records a trace of what operators are used to compute the outputs.

1. Copy the following code into the `DataClassifier.py` file in Visual Studio, above your main function.

```
py

#Function to Convert to ONNX
def convert():

    # set the model to inference mode
    model.eval()

    # Let's create a dummy input tensor
    dummy_input = torch.randn(1, 3, 32, 32, requires_grad=True)

    # Export the model
    torch.onnx.export(model,           # model being run
                      dummy_input,      # model input (or a tuple for multiple inputs)
                      "Network.onnx",    # where to save the model
                      export_params=True, # store the trained parameter weights inside
    the model file
                      opset_version=11,   # the ONNX version to export the model to
                      do_constant_folding=True, # whether to execute constant folding
    for optimization
                      input_names = ['input'],    # the model's input names
                      output_names = ['output'], # the model's output names
                      dynamic_axes={'input' : {0 : 'batch_size'},     # variable length
    axes
                                         'output' : {0 : 'batch_size'}})
    print(" ")
    print('Model has been converted to ONNX')
```

It's important to call `model.eval()` or `model.train(False)` before exporting the model, as this sets the model to **inference mode**. This is needed since operators like `dropout` or `batchnorm` behave differently in inference and training mode.

2. To run the conversion to ONNX, add a call to the conversion function to the main function. You don't need to train the model again, so we'll comment out some functions that we no longer need to run. Your main function will be as follows.

```
py
```

```
if __name__ == "__main__":
    num_epochs = 10
    train(num_epochs)
    print('Finished Training\n')
    test()
    test_species()
    convert()
```

3. Run the project again by selecting the `Start Debugging` button on the toolbar, or pressing `F5`. There's no need to train the model again, just load the existing model from the project folder.

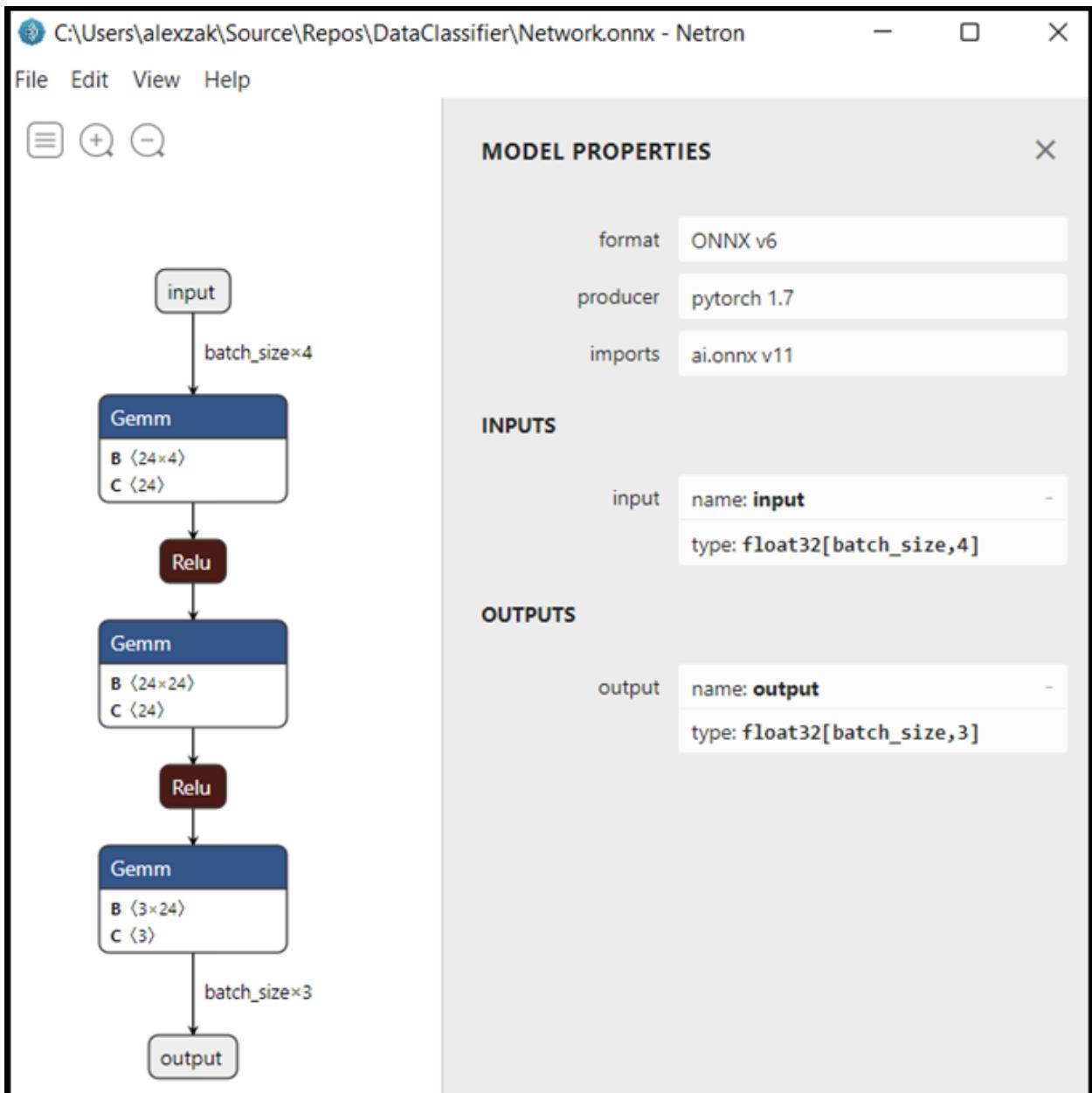
Navigate to your project location and find the ONNX model next to the `.pth` model.

ⓘ Note

Interested in learning more? Review the [PyTorch tutorial on exporting a mdoel ↗](#).

Explore your model.

1. Open the `Network.onnx` model file with Neutron.
2. Select the *data* node to open the model properties.



As you can see, the model requires a 32-bit tensor (multi-dimensional array) float object as an input, and returns a Tensor float as an output. The output array will include the probability for every label. The way you built the model, the labels are represented by 3 numbers, each one associated with a specific type of iris flower.

Label 1	label 2	Label 3
0	1	2
Iris-setosa	Iris-versicolor	Iris-virginica

You'll need to extract these values to show the correct prediction with Windows ML app.

Next Steps

Our model is ready to deploy. Next, for the main event - let's [build a Windows application and run it locally on your Windows device](#).

Deploy your data analysis model in Windows app with Windows ML APIs

Article • 12/02/2022

In the previous part of this tutorial, you learned how to build and export a model in ONNX format. Now, we'll show you how to embed your exported model into a Windows application, and run it locally on a device by calling Windows ML APIs.

By the time we're done, you'll have a working data analysis app.

About the sample app

In this step of the tutorial, you'll create an app that can analyze tabular data of Irises. The app will allow you to add the excel file with the required input information or manually enter the input parameters – the length and width of Iris's sepal and petal in cm. These features will be processed by a locally stored neural network ONNX model you built and trained in the previous part. Based on the model output, the app will display the correct Iris type.

Here, we'll walk you through that process.

Note

If you choose to use the predefined code sample, you can clone [the solution file ↗](#). Clone the repository, navigate to this sample, and open the `Iris Data Analysis.csproj` file with Visual Studio. Skip to the **Launch the Application** part of this page to see it in use.

Below, we'll guide you how to create your app and add Windows ML code.

Create a Windows ML Desktop (C#) app

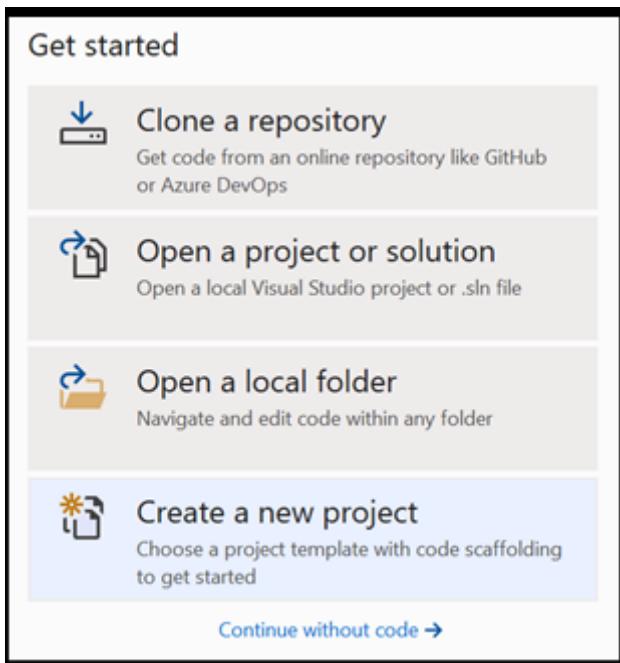
To make a working Windows ML app, you'll need to do the following:

- Load a machine learning model.
- Bind the model's inputs and outputs.
- Evaluate the model and display meaningful results.

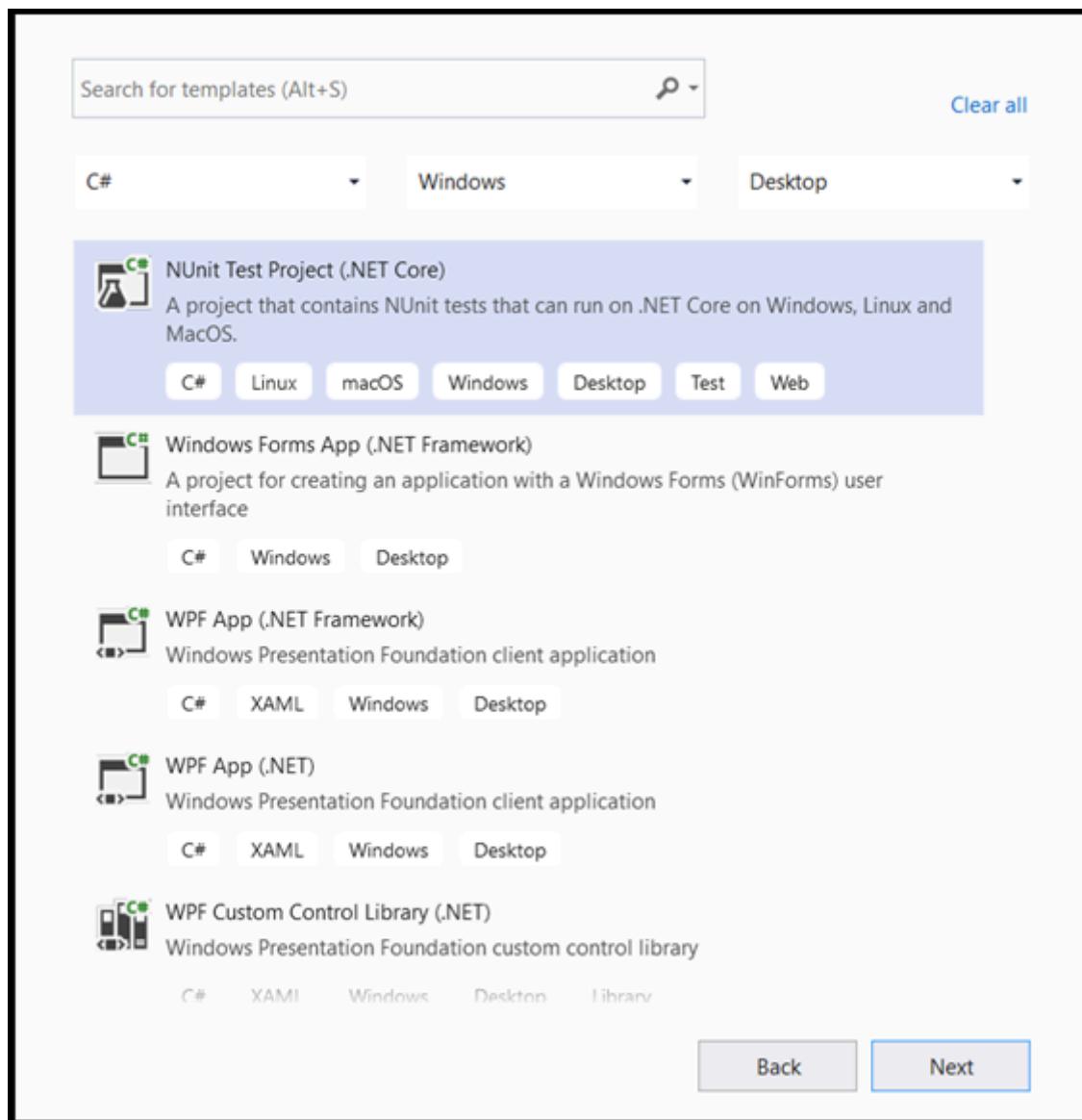
You'll also need to create a basic UI, to provide a better user experience.

Open a new project within Visual Studio

1. Let's get started. Open Visual Studio and choose `Create a new project`.



2. In the search bar, choose `C#` as your language, `Windows` as your target platform, and `Desktop` as your project type. Select `NUnit Test Project (.NET Core)` as your project type, and select `next` to open a configuration window for the project.



3. In the configuration window, do the following:

- Name your project. Here, we call it **Iris Data Analysis**.
- Choose the location of your project.
- If you're using VS2019, ensure `Create directory for solution` is checked.
- If you're using VS2017, ensure `Place solution and project in the same directory` is unchecked.

Press `create` to create your project. The minimum target version window may pop up.

Be sure your minimum version is set to **Windows 10, version 1809 (10.0; build 17763)** or higher.

4. After the project is created, navigate to the project folder, open the **assets** folder `[...\\DataClassifier\\Assets]`, and copy your `Network.onnx` file to this location.

Explore project solution

Let's explore your project solution.

Visual Studio automatically created several cs-code files inside the Solution Explorer.

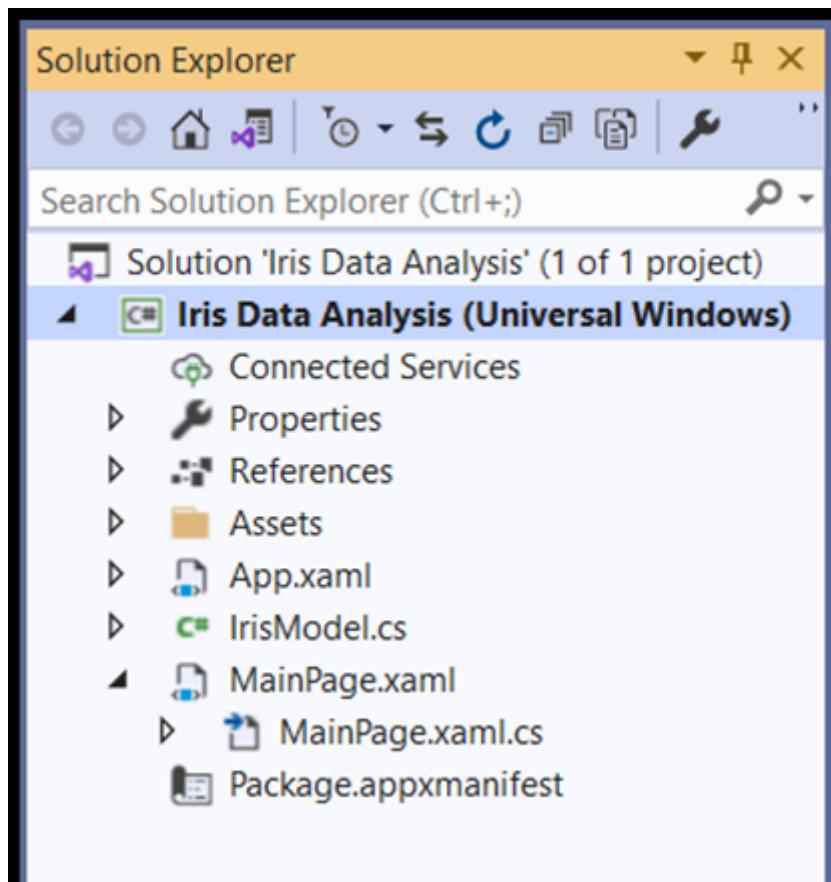
`MainPage.xaml` contains the XAML code for your GUI, and `MainPage.xaml.cs` contains your application code. If you've created a UWP app before, these files should be very familiar to you.

While we've added our `Network.onnx` file to the assets folder, we need to properly add it to this project.

1. Right-click on the Assets folder in the Solution Explorer and, select `Add > Existing Item`.
2. Navigate to the Assets folder inside `Iris Data Analysis [....\Iris Data Analysis \Assets]`, find the `Network.onnx` model you previously copied there, and select `Add`.
3. To ensure the model builds when you compile your application, right-click on the `Network.onnx` file and select `Properties`. Set the `Build Action` to `Content`.

You'll also need to create a new cs-code class file to accommodate some extra machine learning code, which includes classes and methods that will call Windows ML APIs.

4. Right-click on the solution name in Visual Studio, and choose `add` and `new item`. In the open window, select `Class` and give it a name - here, we use `IrisModel.cs`. A new class file will appear under your project.



Create Machine Learning code

In this step, we'll create all the classes and methods that will call the Windows Machine Learning APIs. These will let you load, bind and evaluate an ONNX machine learning model in your project.

1. Double click on the `IrisModel.cs` file.
2. Replace the `using` statements with the following, to get access to all the APIs that you'll need.

```
C#  
  
using System;  
using System.Linq;  
using System.Threading.Tasks;  
using Windows.AI.MachineLearning;  
using Windows.Storage;
```

Initialize machine learning classes

We'll need to add several classes to `IrisModel.cs` to help you interact with Windows Machine Learning APIs.

To get access to the trained machine learning model, we'll use the `LearningModel` class. This class is part of the `Windows.AI.MachineLearning` namespace, and represents a trained machine learning model. Once instantiated, the `LearningModel` is the main object you use to interact with Windows ML APIs.

To evaluate the learning model, you'll have to create an evaluation session. To do so, you use the `LearningModelSession` class. This class is used to evaluate machine learning models, and binds the model to a device that then runs and evaluates the model. When you create a session with this API, you can also select a device to execute your model (the default is your CPU).

Additionally, you'll need to specify the labels of the output of your machine learning models. You can connect those labels to the model's predicted output later.

 **Note**

To learn more about `LearningModel` and `LearningModelSession` classes, please review the [LearningModel class documentation](#) and the [LearningModelSession class documentation](#).

3. Copy the following code to the `IrisModel.cs` file.

C#

```
class IrisModel
{
    private LearningModel _learning_model;
    private LearningModelSession _session;
    private String[] _labels = { "Iris-setosa", "Iris-versicolor",
    "Iris-virginica"};
```

Load the Model

Next, you'll need to load the machine learning model and create a session, which you'll do with the classes you just defined. To load the model, you'll use several static methods of the `LearningModel` class - in our case, we'll use `LoadFromStorageFileAsync`, which lets you load an ONNX model from an `ISorageFile` asynchronously.

ⓘ Note

To learn more about additional ways of loading the model, please review the [Load a model documentation](#).

1. Copy the following code to the `IrisModel.cs` file.

C#

```
public async Task Initialize()
{
    // Load and create the model and session
    var modelFile = await StorageFile.GetFileFromApplicationUriAsync(new
    Uri($"ms-appx://Assets//Network.onnx"));
    _learning_model = await
    LearningModel.LoadFromStorageFileAsync(modelFile);
    _session = new LearningModelSession(_learning_model);
}
```

Define the model input tensor

Now, we'll define the correct input based on your model requirements. The network model you built in the previous part has four input values. Every input value represents the possible sizes of four features of irises: sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. Based on this input, the model will return the iris type that best fits those parameters. You'll need to limit the size of input values to the valid logical values - for this tutorial, we'll use the following:

- sepal length - 1cm to 100cm
- sepal width – 1cm to 8cm
- petal length – 0.5cm to 10cm
- petal width – 0.1cm to 5cm

1. Copy the following code to the `IrisModel.cs` file.

C#

```
private float _sepal_length = 1.0f;
public float Sepal_Length
{
    get
    {
        return _sepal_length;
    }
    set
    {
        // validate range [1,10]
        if (value >= 1 && value <= 10)
        {
            _sepal_length = value;
        }
    }
}

private float _sepal_width = 1.0f;
public float Sepal_Width
{
    get
    {
        return _sepal_width;
    }
    set
    {
        // validate range [1, 8]
        if (value >= 1 && value <= 8)
        {
            _sepal_width = value;
        }
    }
}

private float _petal_length = 0.5f;
```

```

public float Petal_Length
{
    get
    {
        return _petal_length;
    }
    set
    {
        // validate range [0.5, 10]
        if (value >= 0.5 && value <= 10)
        {
            _petal_length = value;
        }
    }
}

private float _petal_width = 0.1f;
public float Petal_Width
{
    get
    {
        return _petal_width;
    }
    set
    {
        // validate range [0.1, 5]
        if (value >= 0.1 && value <= 5)
        {
            _petal_width = value;
        }
    }
}

```

Windows ML APIs accept input values of the four descriptive classes supported by ONNX models: tensors, sequences, maps, and images. In this case, the model requires a 32-bit tensor float object in a shape of float32[batch_size,4]. Since the batch size is 1, the input tensor shape is [1x4].

To create a tensor input, you'll use the [TensorFloat](#) class.

The [TensorFloat](#) class is part of the [Windows.AI.MachineLearning](#) namespace, and is used to define a 32-bit float tensor object - a tensor of 32-bit floating point values. This class contains several useful methods to build a tensor. In your case, you'll use the [CreateFromArray](#) method to build a tensor input in the exact size your model requires. We'll add that call within the evaluation method.

Bind and Evaluate the model

Now that you've defined the model input tensor and instantiated the trained model and session, it's time to create a method to bind and evaluate the trained machine learning model.

This method is the key part of a machine learning app. It includes the tensorization of the input values and binding of the model input. **You'll use this model later in your application code to evaluate your model.**

To bind input and output, you use the `LearningModelBinding` class. A machine learning model has input and output features, which pass information into and out of the model. Be aware that required features must be supported by the Windows ML APIs. The `LearningModelBinding` class is applied on a `LearningModelSession` to bind values to named input and output features.

The `LearningModelBinding` class has several predefined methods you can use to bind values to those named features. Here, you'll use the `Bind` method to bind values to your model.

To evaluate your model and receive results from it, you call the relevant predefined evaluation methods from `LearningModelSession` - in your case, the `Evaluate` method. This method will provide the functionality you need, evaluating the machine learning model using the feature values supplied by the `LearningModelBinding` class.

① Note

To learn about another evaluate methods to run the model, please check which methods can be implemented on the `LearningModelSession` by reviewing the [LearningModelSession Class documentation](#).

Extract and display the results

The model returns the predicted values in tensor format as a Tensor float output. You'll now need to extract the model output and display the right results. To do this, you'll convert the tensor format to a vector by running the `GetAsVectorView()` function on the predicated output.

The model returns three probability values, each representing one specific iris type. You'll need to return the label with the highest probability.

1. Copy the following code to the `IrisModel.cs` file.

C#

```

internal String Evaluate()
{
    // input tensor shape is [1x4]
    long[] shape = new long[2];
    shape[0] = 1;
    shape[1] = 4;

    // set up the input tensor
    float[] input_data = new float[4];
    input_data[0] = _sepal_length;
    input_data[1] = _sepal_width;
    input_data[2] = _petal_length;
    input_data[3] = _petal_width;
    TensorFloat tensor_float = TensorFloat.CreateFromArray(shape,
    input_data);

    // bind the tensor to "input"
    var binding = new LearningModelBinding(_session);
    binding.Bind("input", tensor_float);

    // evaluate
    var results = _session.Evaluate(binding, "");

    // get the results
    TensorFloat prediction = (TensorFloat)results.Outputs.First().Value;
    var prediction_data = prediction.GetAsVectorView();

    // find the highest predicted value
    int max_index = 0;
    float max_value = 0;
    for (int i = 0; i < prediction_data.Count; i++)
    {
        var val = prediction_data.ElementAt(i);
        if (val > max_value)
        {
            max_value = val;
            max_index = i;
        }
    }

    // return the label corresponding to the highest predicted value
    return _labels.ElementAt(max_index);
}

```

You've now completed the machine learning part of your code. Now, you can easily integrate your model with Windows application. In the last part of this tutorial, we've provided a basic Windows GUI and control code to test the model, using the methods you've already created.

Create the application GUI

1. To create a GUI app code for your app, double-click on the `MainPage.xaml` code file and open a predefined template for your GUI.
2. Copy-paste the below code to `MainPage.xaml`, under the `“Background= {ThemeResource ApplicationPageBackgroundThemeBrush}”` line.

XAML

```

<Grid Margin="30,30,30,30">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <TextBlock x:Name="title" HorizontalAlignment="Left" Text="Data
Analysis App - Windows ML" TextWrapping="Wrap" VerticalAlignment="Top"
FontSize="32" TextDecorations="Underline" FontWeight="Bold"/>
    <TextBlock x:Name="subtitle" HorizontalAlignment="Left"
Text="Provide the input :" TextWrapping="Wrap" VerticalAlignment="Top"
FontSize="20" Grid.Row="1" FontWeight="Bold"/>
    <Grid Grid.Row="2">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <TextBlock x:Name="sepal_length" Text="sepal length in mm [range
of 10 - 100]" VerticalAlignment="Center"/>
            <TextBlock x:Name="sepal_width" Text="sepal width in mm [range
of 10 - 80]" VerticalAlignment="Center" Grid.Row="1"/>
            <TextBlock x:Name="petal_length" Text="petal length in mm [range
of 5 - 100]" VerticalAlignment="Center" Grid.Row="2"/>
            <TextBlock x:Name="petal_width" Text="sepal width in mm [range
of 1 - 50]" VerticalAlignment="Center" Grid.Row="3"/>

            <Slider x:Name="sepal_length_input" Minimum="10" Maximum="100"
Orientation="Horizontal" Grid.Column="1" Width="200"
ValueChanged="sepal_length_input_ValueChanged"/>
            <Slider x:Name="sepal_width_input" Minimum="10" Maximum="80"
Orientation="Horizontal" Grid.Row="1" Grid.Column="1" Width="200"
ValueChanged="sepal_width_input_ValueChanged"/>
            <Slider x:Name="petal_length_input" Minimum="5" Maximum="100"
Orientation="Horizontal" Grid.Row="2" Grid.Column="1" Width="200"
ValueChanged="petal_length_input_ValueChanged"/>
            <Slider x:Name="petal_width_input" Minimum="1" Maximum="50"
Orientation="Horizontal" Grid.Row="3" Grid.Column="1" Width="200"

```

```

        ValueChanged="petal_width_input_ValueChanged"/>
    </Grid>
    <TextBlock x:Name="output" Text="Output:" FontSize="20"
    FontWeight="Bold" Grid.Row="3"/>
    <Grid Grid.Row="4">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <TextBlock x:Name="output_subtitle" Text="Based on the
    information provided, the Iris type is:"/>
        <TextBlock x:Name="model_output" Text="Model output"
    FontStyle="Italic" Grid.Column="1" Margin="10,0,0,0"/>
    </Grid>
</Grid>

```

Create the application control

The application control code, `MainPage.xaml.cs`, includes the main method to run the app, and several steps to run your model and execute the output:

1. You'll instantiate a new object of the `IrisModel` class that you had created previously in this tutorial.
2. You'll call the `Evaluate()` method you built in the previous part on model. This method will be applied four times, one on each of the input parameters: sepal length, sepal width, petal length, and petal width.

The app will display the result based on the machine learning prediction algorithm.

1. To create an application control code, double-click the `MainPage.xaml.cs` code file and add the following code.

```

C#
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
// The Blank Page item template is documented at
https://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace Iris_Data_Analysis
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a
    Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        private IrisModel _iris_model;

```

```
public MainPage()
{
    this.InitializeComponent();
    _iris_model = new IrisModel();
#pragma warning disable CS4014 // Because this call is not awaited,
execution of the current method continues before the call is completed
    _iris_model.Initialize();
#pragma warning restore CS4014 // Because this call is not awaited,
execution of the current method continues before the call is completed
}

private void sepal_length_input_ValueChanged(object sender,
RangeBaseValueChangedEventArgs e)
{
    if (_iris_model != null)
    {
        _iris_model.Sepal_Length = (float)sepal_length_input.Value /
10.0f;
        model_output.Text = _iris_model.Evaluate();
    }
}

private void sepal_width_input_ValueChanged(object sender,
RangeBaseValueChangedEventArgs e)
{
    if (_iris_model != null)
    {
        _iris_model.Sepal_Width = (float)sepal_width_input.Value /
10.0f;
        model_output.Text = _iris_model.Evaluate();
    }
}

private void petal_length_input_ValueChanged(object sender,
RangeBaseValueChangedEventArgs e)
{
    if (_iris_model != null)
    {
        _iris_model.Petal_Length = (float)petal_length_input.Value /
10.0f;
        model_output.Text = _iris_model.Evaluate();
    }
}

private void petal_width_input_ValueChanged(object sender,
RangeBaseValueChangedEventArgs e)
{
    if (_iris_model != null)
    {
        _iris_model.Petal_Width = (float)petal_width_input.Value /
10.0f;
        model_output.Text = _iris_model.Evaluate();
    }
}
```

```
}
```

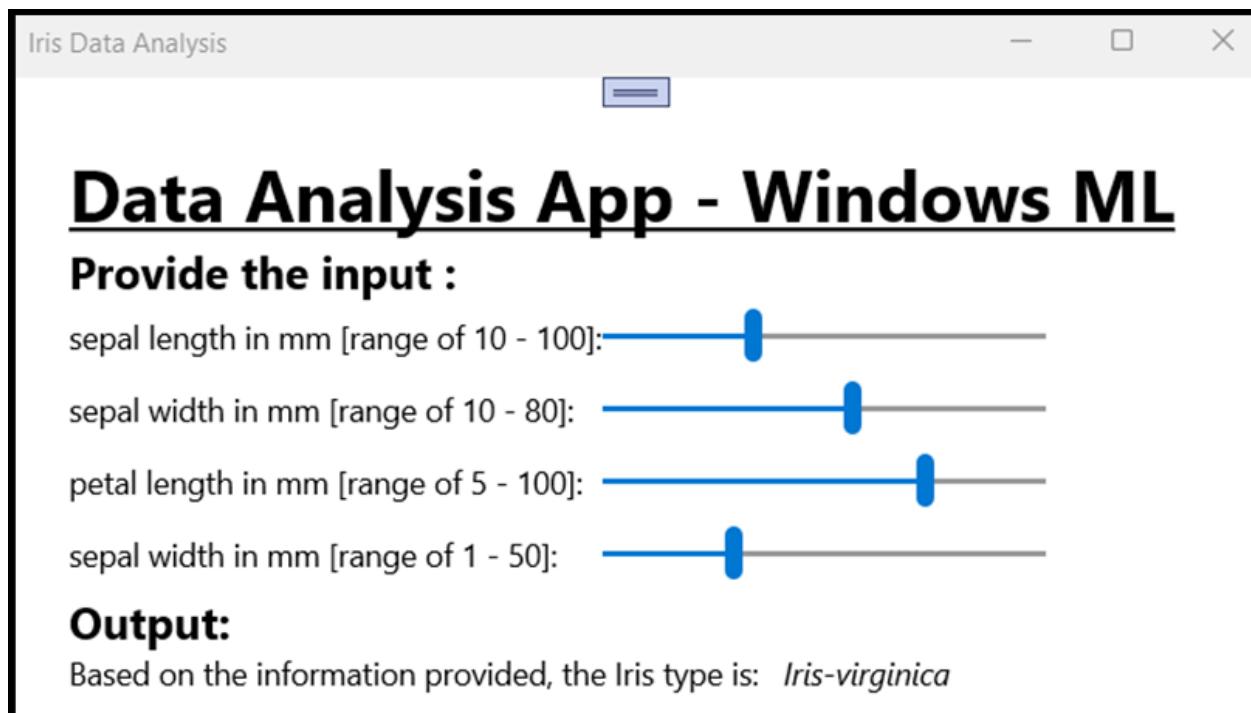
Launch the application

Now you're ready to launch your application and see the results.

Enable developer mode and test your application from Visual Studio. Make sure the dropdown menus in the top toolbar are set to `Debug`. 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.

The app GUI includes four sliders to change the input of the required parameters. Any change in the input will generate a new output based on the prediction algorithm. The output is displayed below the input sliders.

You can see that the given the input of sepal length = 40mm, sepal width = 50, petal length = 75, and petal width = 15, the app generated input of Iris-versicolor type!



Summary

You've just made your first Windows Machine Learning app, from model creation to successful execution.

Additional Resources

To learn more about topics mentioned in this tutorial, visit the following resources:

- Windows ML tools: Learn more tools like the [Windows ML Dashboard](#), [WinMLRunner](#), and the [mglens](#) Windows ML code generator.
 - ONNX model: Learn more about the ONNX format.
 - [Windows ML performance and memory](#): Learn more about how to manage app performance with Windows ML.
 - [Windows Machine Learning API reference](#): Learn more about three areas of Windows ML APIs.
-

Feedback

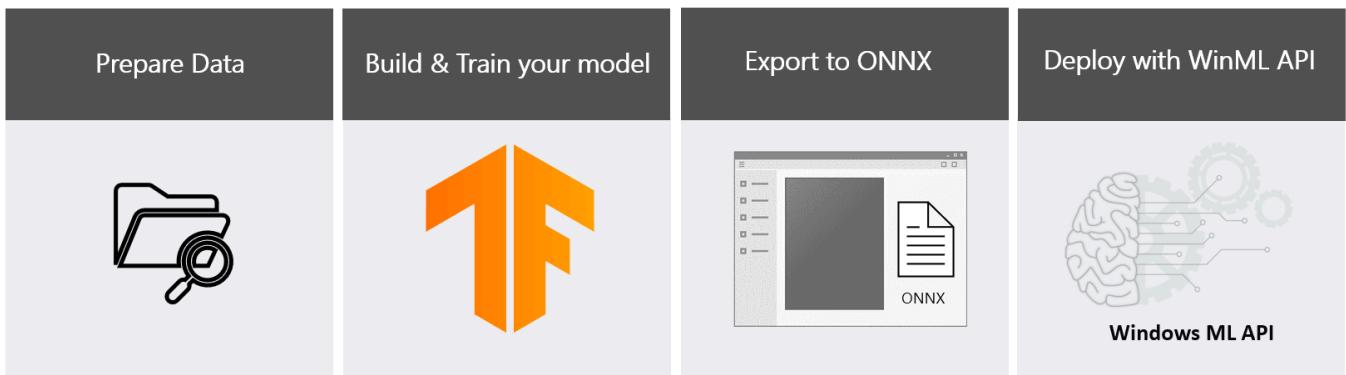
Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

TensorFlow + DirectML with Windows ML: Real-time object detection from video



This tutorial shows how to locally train and evaluate a real-time object detection model in a UWP application. The model will be trained with TensorFlow locally on your machine through the DirectML APIs, which provides GPU accelerated training across all Windows devices. The trained model will then be integrated into a UWP app which uses your webcam to detect objects in the frame in real-time, locally using Windows ML APIs.

We'll start by enabling TensorFlow on your machine.

If you'd like to learn how to train your model with TensorFlow, you can proceed to [Train a Model](#).

If you have a TensorFlow model, but want to learn how to convert it to an ONNX format suitable for use with WinML APIs, see [convert your model](#).

If you have a model and want to learn how to create a WinML app from scratch, navigate to [Deploy your model](#).

Enable GPU acceleration for TensorFlow with DirectML

To enable TensorFlow on your machine, proceed through the following steps.

Check your version of Windows

The TensorFlow with DirectML package on native Windows works on Windows 10 Version 1709 (Build 16299) or later versions of Windows. You can check your build version number by running `winver` via the **Run** command (`Windows logo key + R`).

Check for GPU driver updates

Ensure you have the latest GPU driver installed. Select **Check for updates** in the **Windows Update** section of Windows **Settings**.

Set up the TensorFlow with DirectML preview

For use with TensorFlow, we recommend setting up a virtual Python environment inside Windows. There are many tools you can use to setup a virtual Python environment — for these instructions, we'll use [Anaconda's miniconda](#). The rest of this setup assumes you use a miniconda environment.

Set up Python environment

! Note

In the commands below, we use Python 3.6. However, the `tensorflow-directml` package works in a Python 3.5, 3.6 or 3.7 environment.

Download and install the [Miniconda Windows installer](#) on your machine. If you need it, there is [additional guidance for setup](#) on Anaconda's site. Once Miniconda is installed, create an environment using Python named directml and activate it through the following commands:

```
conda create --name directml python=3.6 conda activate directml
```

Install the Tensorflow with DirectML package

! Note

The `tensorflow-directml` package only supports TensorFlow 1.15.

Install the TensorFlow with DirectML package through pip by running the following command:

```
pip install tensorflow-directml
```

Verify package installation

Once you've installed the `tensorflow-directml` package, you can verify that it runs correctly by adding two tensors. Copy the following lines into an interactive Python session:

```
py
```

```
import tensorflow.compat.v1 as tf  
  
tf.enable_eager_execution(tf.ConfigProto(log_device_placement=True))  
  
print(tf.add([1.0, 2.0], [3.0, 4.0]))
```

You should see output similar to the following, with the add operator placed on the DML device.

Next Steps

Now that you've gotten your prerequisites sorted out, you can proceed to creation of your WinML model. [In the next part](#), you'll use TensorFlow to create your real-time object-detection model.

 **Important**

TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Last updated on 09/18/2025

Train an object detection model with TensorFlow

Article • 12/30/2021

Now that we've configured TensorFlow, we'll use the YOLO architecture to train the object detection model. YOLO is a neural network which predicts bounding boxes and class probabilities from an image in a single evaluation. YOLO models can process over 60 frames per second, making it a great architecture for detecting objects in videos. You can find more information on how YOLO works [here](#).

Using YOLO

First, download [this YOLO sample file](#), which contains helper scripts to get started.

When using YOLO, we have 3 options:

1. **Utilize pre-trained model weights for YOLO.** The pre-trained model has been trained on a large dataset with 80 classes (categories) for everyday objects like bus, person, sandwich, etc. If you want to download a pre-trained YOLO model in ONNX format, you can do so [here](#). Then, you can proceed to [the final stage of this tutorial](#) to learn how to integrate that model into an app.
2. **Implement transfer learning with a custom dataset.** Transfer learning is a method for using a trained model as a starting point to train a model solving a different but related task. This tutorial will use the pre-trained YOLO weights with 80 classes to train a model with 20 classes with the [VOC dataset](#). If you want to create your own dataset with custom classes, see instructions [here](#).
3. **Train YOLO from scratch.** This technique is not recommended, because it is very difficult to converge. The original YOLO paper trained darknet on imagenet (containing hundreds of thousands of photos) before training the whole network as well.

Implement transfer learning on pre-trained YOLO weights to the VOC dataset:

Let's proceed with the second option, and implement transfer learning with the following steps.

1. In a miniconda window, navigate to the yolo sample directory and run the following command to install all the required pip packages for YOLO.

```
pip install -r requirements.txt
```

2. Run the setup script to download the data and pre-trained weights

```
python setup.py
```

3. Transform the dataset. See `tools/voc2012.py` for implementation - this format is based on the [tensorflow object detection API](#). Many fields are not required, but have here been filled in for compatibility with the official API.

```
python tools/voc2012.py \
--data_dir './data/voc2012_raw/VOCdevkit/VOC2012' \
--split train \
--output_file ./data/voc2012_train.tfrecord

python tools/voc2012.py \
--data_dir './data/voc2012_raw/VOCdevkit/VOC2012' \
--split val \
--output_file ./data/voc2012_val.tfrecord
```

4. Train the model. Run the following commands:

```
python convert.py
python detect.py --image ./data/meme.jpg # Sanity check

python train.py \
--dataset ./data/voc2012_train.tfrecord \
--val_dataset ./data/voc2012_val.tfrecord \
--classes ./data/voc2012.names \
--num_classes 20 \
--mode fit --transfer darknet \
--batch_size 16 \
--epochs 10 \
--weights ./checkpoints/yolov3.tf \
--weights_num_classes 80
```

You now have a re-trained model with 20 classes, ready for use.

Next steps

Now that we've created a TensorFlow model, we need to [convert it to ONNX format](#) for use with the Windows Machine Learning APIs.

Convert TensorFlow model to ONNX

Article • 12/30/2021

In [the previous step of this tutorial](#), we created a machine learning model with TensorFlow. Now, we'll convert it to the ONNX format.

Here, we'll use the `tf2onnx` tool to convert our model, following these steps.

1. Save the tf model in preparation for ONNX conversion, by running the following command.

```
python save_model.py --weights ./data/yolov4.weights --output  
./checkpoints/yolov4.tf --input_size 416 --model yolov4
```

2. Install `tf2onnx` and `onnxruntime`, by running the following commands.

```
pip install onnxruntime  
pip install git+https://github.com/onnx/tensorflow-onnx
```

3. Convert the model, by running the following command.

```
python -m tf2onnx.convert --saved-model ./checkpoints/yolov4.tf --output model.onnx  
--opset 11 --verbose
```

Next steps

We've now converted our model to an ONNX format, suitable for use with Windows Machine Learning APIs. In the final stage of this tutorial, we [integrate it into a Windows app](#).

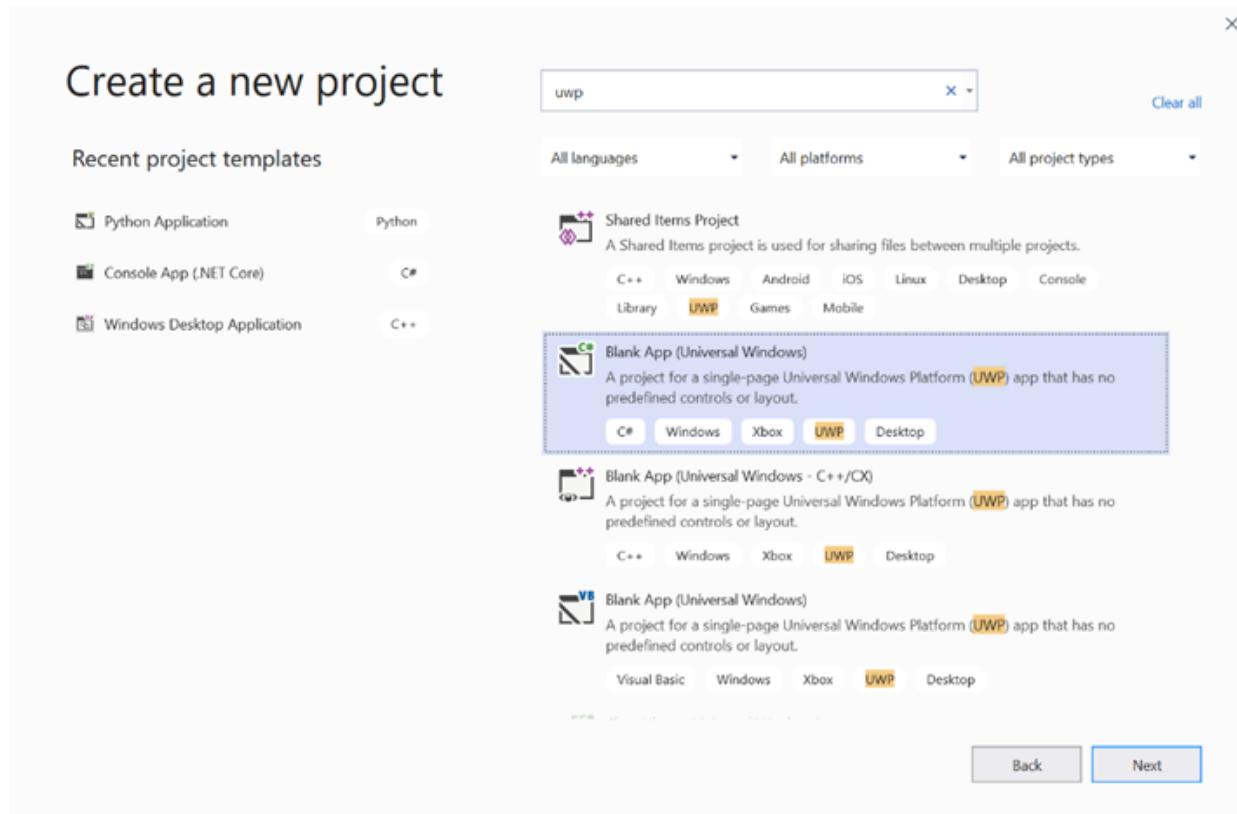
Deploy your TensorFlow model in a Windows app with the Windows Machine Learning APIs

Article • 12/30/2021

This final section will explain how to create a simple UWP app with a GUI to stream the webcam and detect objects by evaluating our YOLO model with Windows ML.

Create a UWP app in Visual Studio

1. Open Visual Studio and select `Create a new project`. Search for UWP and select `Blank App (Universal Windows)`.



2. On the next page, configure your project settings by giving the project a Name and Location. Then select a target and minimum OS version of your app. To use Windows ML APIs you must use X, or you can choose the NuGet package to support down to X. If you chose to use the NuGet package, follow these instructions [link].

Call Windows ML APIs to evaluate the model

Step 1: Use the Machine Learning Code Generator to generate wrapper classes for Windows ML APIs.

Step 2: Modify generated code in the generated .cs file. The final file looks like this:

C#

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Windows.Media;
using Windows.Storage;
using Windows.Storage.Streams;
using Windows.AI.MachineLearning;
namespace yolodemo
{

    public sealed class YoloInput
    {
        public TensorFloat input_100; // shape(-1,3,416,416)
    }

    public sealed class YoloOutput
    {
        public TensorFloat concat_1600; // shape(-1,-1,-1)
    }

    public sealed class YoloModel
    {
        private LearningModel model;
        private LearningModelSession session;
        private LearningModelBinding binding;
        public static async Task<YoloModel>
CreateFromStreamAsync(IRandomAccessStreamReference stream)
        {
            YoloModel learningModel = new YoloModel();
            learningModel.model = await
LearningModel.LoadFromStreamAsync(stream);
            learningModel.session = new
LearningModelSession(learningModel.model);
            learningModel.binding = new
LearningModelBinding(learningModel.session);
            return learningModel;
        }
        public async Task<YoloOutput> EvaluateAsync(YoloInput input)
        {
            binding.Bind("input_1:0", input.input_100);
            var result = await session.EvaluateAsync(binding, "0");
            var output = new YoloOutput();
            output.concat_1600 = result.Outputs["concat_16:0"] as
TensorFloat;
            return output;
        }
    }
}
```

```
    }  
}
```

Evaluate each video frame to detect objects and draw bounding boxes.

1. Add the following libraries to MainPage.xaml.cs.

C#

```
using System.Threading.Tasks;  
using Windows.Devices.Enumeration;  
using Windows.Media;  
using Windows.Media.Capture;  
using Windows.Storage;  
using Windows.UI;  
using Windows.UI.Xaml.Media.Imaging;  
using Windows.UI.Xaml.Shapes;  
using Windows.AI.MachineLearning;
```

2. Add the following variables in `public sealed partial class MainPage : Page`.

C#

```
private MediaCapture _media_capture;  
private LearningModel _model;  
private LearningModelSession _session;  
private LearningModelBinding _binding;  
private readonly SolidColorBrush _fill_brush = new  
SolidColorBrush(Colors.Transparent);  
private readonly SolidColorBrush _line_brush = new  
SolidColorBrush(Colors.DarkGreen);  
private readonly double _line_thickness = 2.0;  
private readonly string[] _labels =  
{  
    "<list of labels>"  
};
```

3. Create a structure for how the detection results are formatted.

C#

```
internal struct DetectionResult  
{  
    public string label;  
    public List<float> bbox;
```

```
        public double prob;  
    }
```

4. Create a Comparer object which compares two objects of type Box. This class will be used to draw bounding boxes around the detected objects.

C#

```
class Comparer : IComparer<DetectionResult>  
{  
    public int Compare(DetectionResult x, DetectionResult y)  
    {  
        return y.prob.CompareTo(x.prob);  
    }  
}
```

5. Add the following method to initialize the device's webcam stream and begin processing each frame to detect objects.

C#

```
private async Task InitCameraAsync()  
{  
    if (_media_capture == null || _media_capture.CameraStreamState  
== Windows.Media.Devices.CameraStreamState.Shutdown ||  
_media_capture.CameraStreamState ==  
Windows.Media.Devices.CameraStreamState.NotStreaming)  
    {  
        if (_media_capture != null)  
        {  
            _media_capture.Dispose();  
        }  
  
        MediaCaptureInitializationSettings settings = new  
MediaCaptureInitializationSettings();  
        var cameras = await  
DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);  
        var camera = cameras.FirstOrDefault();  
        settings.VideoDeviceId = camera.Id;  
  
        _media_capture = new MediaCapture();  
        await _media_capture.InitializeAsync(settings);  
        WebCam.Source = _media_capture;  
    }  
  
    if (_media_capture.CameraStreamState ==  
Windows.Media.Devices.CameraStreamState.NotStreaming)  
    {  
        await _media_capture.StartPreviewAsync();  
        WebCam.Visibility = Visibility.Visible;  
    }  
}
```

```
        ProcessFrame();
    }
```

6. Add the following method to process each frame. This method calls EvaluateFrame and DrawBoxes, which we will implement in a later step.

C#

```
private async Task ProcessFrame()
{
    var frame = new
VideoFrame(Windows.Graphics.Imaging.BitmapPixelFormat.Bgra8,
(int)WebCam.Width, (int)WebCam.Height);
    await _media_capture.GetPreviewFrameAsync(frame);
    var results = await EvaluateFrame(frame);
    await DrawBoxes(results.ToArray(), frame);
    ProcessFrame();
}
```

7. Create a new Sigmoid float

C#

```
private float Sigmoid(float val)
{
    var x = (float)Math.Exp(val);
    return x / (1.0f + x);
}
```

8. Create a threshold for detecting objects correctly.

C#

```
private float ComputeIOU(DetectionResult DRa, DetectionResult DRb)
{
    float ay1 = DRa.bbox[0];
    float ax1 = DRa.bbox[1];
    float ay2 = DRa.bbox[2];
    float ax2 = DRa.bbox[3];
    float by1 = DRb.bbox[0];
    float bx1 = DRb.bbox[1];
    float by2 = DRb.bbox[2];
    float bx2 = DRb.bbox[3];

    Debug.Assert(ay1 < ay2);
    Debug.Assert(ax1 < ax2);
    Debug.Assert(by1 < by2);
    Debug.Assert(bx1 < bx2);
```

```

        // determine the coordinates of the intersection rectangle
        float x_left = Math.Max(ax1, bx1);
        float y_top = Math.Max(ay1, by1);
        float x_right = Math.Min(ax2, bx2);
        float y_bottom = Math.Min(ay2, by2);

        if (x_right < x_left || y_bottom < y_top)
            return 0;
        float intersection_area = (x_right - x_left) * (y_bottom -
y_top);
        float bb1_area = (ax2 - ax1) * (ay2 - ay1);
        float bb2_area = (bx2 - bx1) * (by2 - by1);
        float iou = intersection_area / (bb1_area + bb2_area -
intersection_area);

        Debug.Assert(iou >= 0 && iou <= 1);
        return iou;
    }
}

```

9. Implement the following list, to track the current objects detected in the frame.

C#

```

private List<DetectionResult> NMS(IReadOnlyList<DetectionResult>
detections,
    float IOU_threshold = 0.45f,
    float score_threshold=0.3f)
{
    List<DetectionResult> final_detections = new
List<DetectionResult>();
    for (int i = 0; i < detections.Count; i++)
    {
        int j = 0;
        for (j = 0; j < final_detections.Count; j++)
        {
            if (ComputeIOU(final_detections[j], detections[i]) >
IOU_threshold)
            {
                break;
            }
        }
        if (j==final_detections.Count)
        {
            final_detections.Add(detections[i]);
        }
    }
    return final_detections;
}

```

10. Implement the following method.

C#

```

private List<DetectionResult> ParseResult(float[] results)
{
    int c_values = 84;
    int c_boxes = results.Length / c_values;
    float confidence_threshold = 0.5f;
    List<DetectionResult> detections = new List<DetectionResult>();
    this.OverlayCanvas.Children.Clear();
    for (int i_box = 0; i_box < c_boxes; i_box++)
    {
        float max_prob = 0.0f;
        int label_index = -1;
        for (int j_confidence = 4; j_confidence < c_values;
j_confidence++)
        {
            int index = i_box * c_values + j_confidence;
            if (results[index] > max_prob)
            {
                max_prob = results[index];
                label_index = j_confidence - 4;
            }
        }
        if (max_prob > confidence_threshold)
        {
            List<float> bbox = new List<float>();
            bbox.Add(results[i_box * c_values + 0]);
            bbox.Add(results[i_box * c_values + 1]);
            bbox.Add(results[i_box * c_values + 2]);
            bbox.Add(results[i_box * c_values + 3]);

            detections.Add(new DetectionResult()
            {
                label = _labels[label_index],
                bbox = bbox,
                prob = max_prob
            });
        }
    }
    return detections;
}

```

11. Add the following method, to draw the boxes around the objects detected in the frame.

C#

```

private async Task DrawBoxes(float[] results, VideoFrame frame)
{
    List<DetectionResult> detections = ParseResult(results);
    Comparer cp = new Comparer();
    detections.Sort(cp);
    IReadOnlyList<DetectionResult> final_detetions =
NMS(detections);

```

```

        for (int i=0; i < final_detetions.Count; ++i)
        {
            int top = (int)(final_detetions[i].bbox[0] * WebCam.Height);
            int left = (int)(final_detetions[i].bbox[1] * WebCam.Width);
            int bottom = (int)(final_detetions[i].bbox[2] *
WebCam.Height);
            int right = (int)(final_detetions[i].bbox[3] *
WebCam.Width);

            var brush = new ImageBrush();
            var bitmap_source = new SoftwareBitmapSource();
            await bitmap_source.SetBitmapAsync(frame.SoftwareBitmap);

            brush.ImageSource = bitmap_source;
            // brush.Stretch = Stretch.Fill;

            this.OverlayCanvas.Background = brush;

            var r = new Rectangle();
            r.Tag = i;
            r.Width = right - left;
            r.Height = bottom - top;
            r.Fill = this._fill_brush;
            r.Stroke = this._line_brush;
            r.StrokeThickness = this._line_thickness;
            r.Margin = new Thickness(left, top, 0, 0);

            this.OverlayCanvas.Children.Add(r);
            // Default configuration for border
            // Render text label

            var border = new Border();
            var backgroundColorBrush = new
SolidColorBrush(Colors.Black);
            var foregroundColorBrush = new
SolidColorBrush(Colors.SpringGreen);
            var textBlock = new TextBlock();
            textBlock.Foreground = foregroundColorBrush;
            textBlock.FontSize = 18;

            textBlock.Text = final_detetions[i].label;
            // Hide
            textBlock.Visibility = Visibility.Collapsed;
            border.Background = backgroundColorBrush;
            border.Child = textBlock;

            Canvas.SetLeft(border, final_detetions[i].bbox[1] * 416 +
2);
            Canvas.SetTop(border, final_detetions[i].bbox[0] * 416 + 2);
            textBlock.Visibility = Visibility.Visible;
            // Add to canvas
            this.OverlayCanvas.Children.Add(border);

```

```
        }  
    }
```

12. Now that we've handled the necessary infrastructure, it's time to incorporate the evaluation itself. This method evaluates the model against the current frame to detect objects.

C#

```
private async Task<List<float>> EvaluateFrame(VideoFrame frame)  
{  
    _binding.Clear();  
    _binding.Bind("input_1:0", frame);  
    var results = await _session.EvaluateAsync(_binding, "");  
    Debug.Print("output done\n");  
  
    TensorFloat result = results.Outputs["Identity:0"] as  
TensorFloat;  
    var shape = result.Shape;  
    var data = result.GetAsVectorView();  
  
    return data.ToList<float>();  
}
```

13. Our app needs to start somehow. Add a method which begins the webcam stream and model evaluation when the user presses the Go button.

C#

```
private void button_go_Click(object sender, RoutedEventArgs e)  
{  
    InitModelAsync();  
    InitCameraAsync();  
}
```

14. Add a method to call Windows ML APIs to evaluate the model. First the model is loaded from storage, and then a session is created and bound to memory.

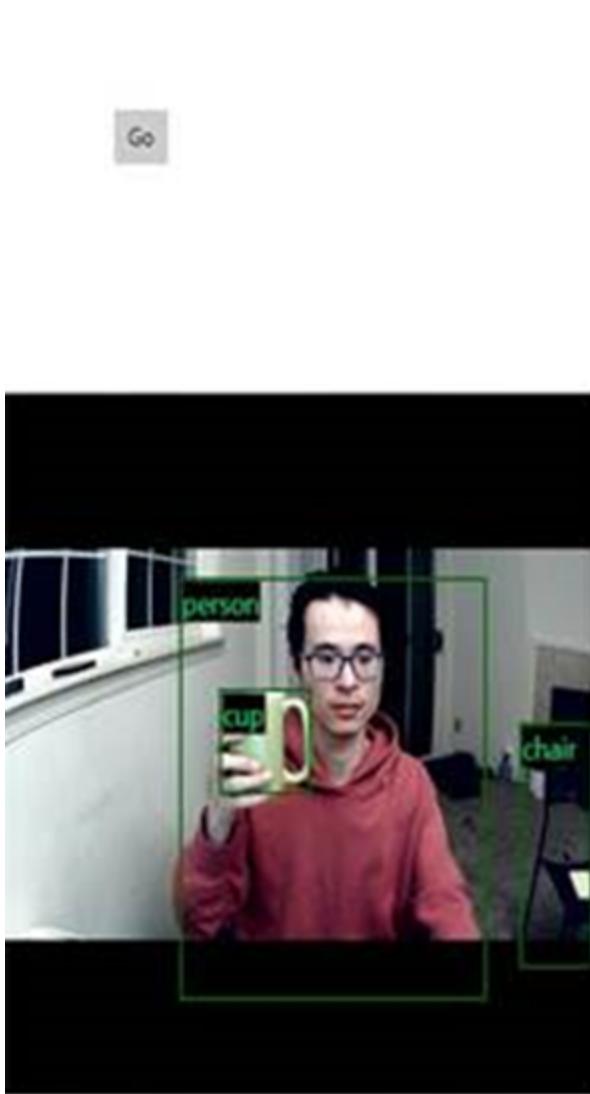
C#

```
private async Task InitModelAsync()  
{  
    var model_file = await  
StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-  
appx:///Assets//Yolo.onnx"));  
    _model = await  
LearningModel.LoadFromStorageFileAsync(model_file);  
    var device = new  
LearningModelDevice(LearningModelDeviceKind.Cpu);
```

```
    _session = new LearningModelSession(_model, device);
    _binding = new LearningModelBinding(_session);
}
```

Launch the application

You have now successfully created a real-time object detection application! Select the **Run** button on the top bar of Visual Studio to launch the app. The app should look like this.



Additional Resources

To learn more about topics mentioned in this tutorial, visit the following resources:

- Windows ML tools: Learn more tools like the [Windows ML Dashboard](#), [WinMLRunner](#), and the [mglenn](#) Windows ML code generator.
- ONNX model: Learn more about the ONNX format.
- [Windows ML performance and memory](#): Learn more how to manage app performance with Windows ML.
- [Windows Machine Learning API reference](#): Learn more about three areas of Windows ML APIs.

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

Article • 12/30/2021

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.

<https://www.youtube-nocookie.com/embed/yC3HKAy0aj4>

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

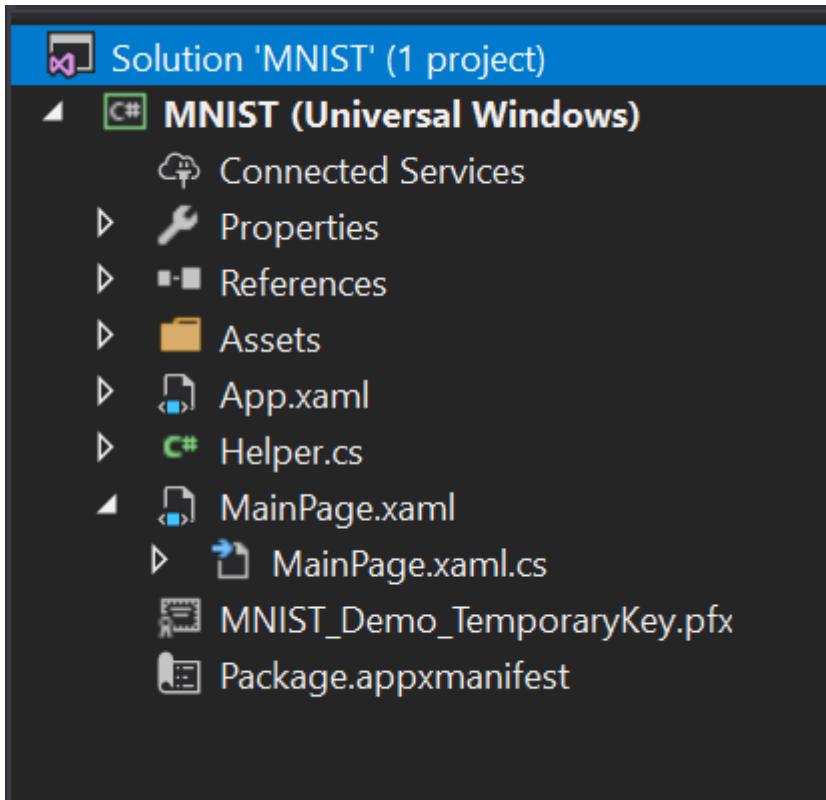
Once you've downloaded the project from GitHub, launch Visual Studio and open the `MNIST_Demo.sln` file (it should be located at `<Path to repo>\Windows-Machine-Learning\Samples\MNIST\Tutorial\cs`). If the solution is shown as unavailable, you'll need to right-click the project in the **Solution Explorer**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



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**.



Handwritten Digit: Result:



Recognize

Clear Digit

ⓘ 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 ↗](#).

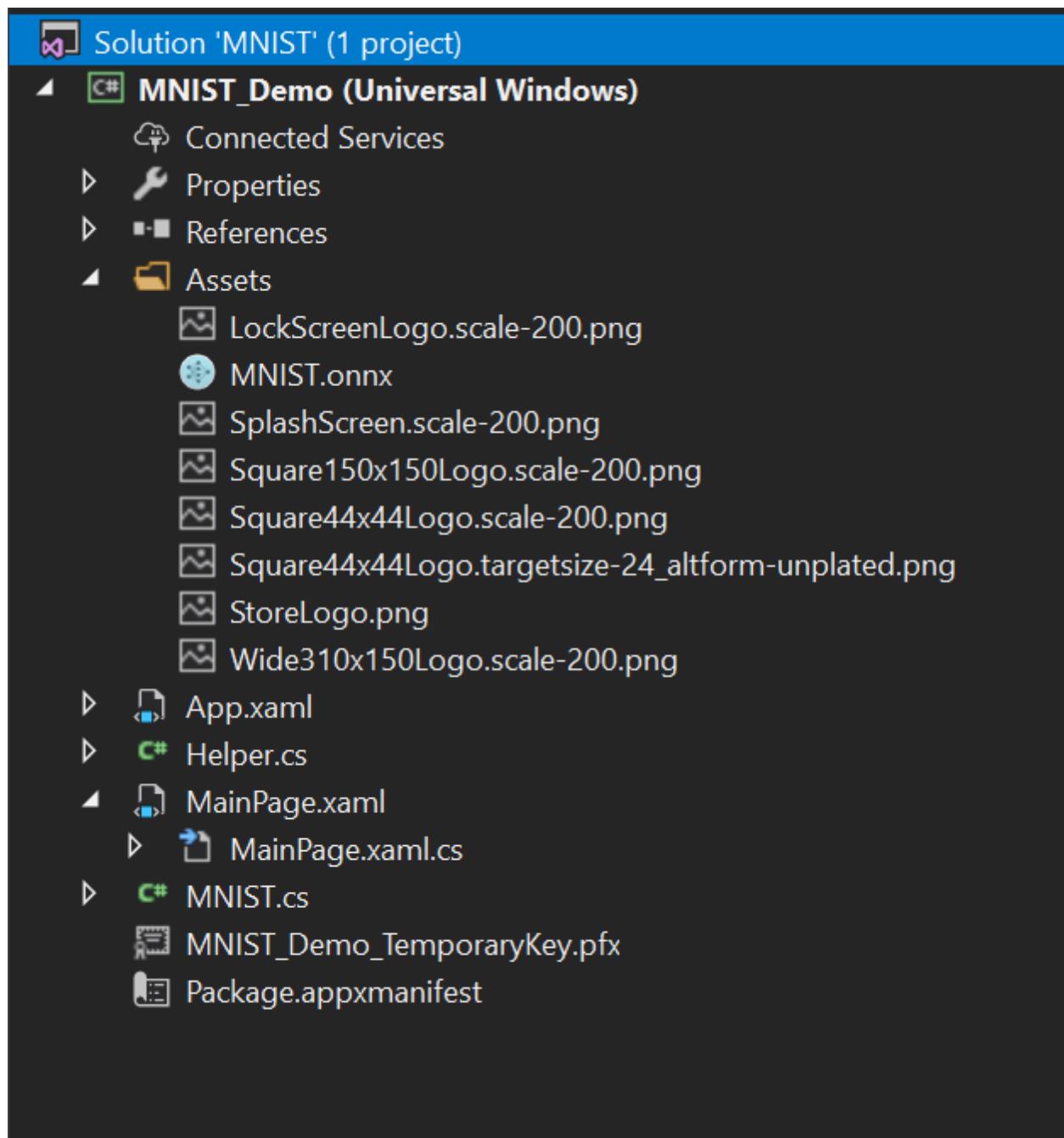
The MNIST model has already been included in your **Assets** folder, and you will need to add it to your application as an existing item. You can also download the pre-trained model from the [ONNX Model Zoo](#) on GitHub.

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.



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:

```
C#  
  
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.

```
C#
```

```
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.

C#

```
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.

C#

```
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.

C#

```
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**.



Handwritten Digit: Result:



2

Recognize

Clear Digit

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

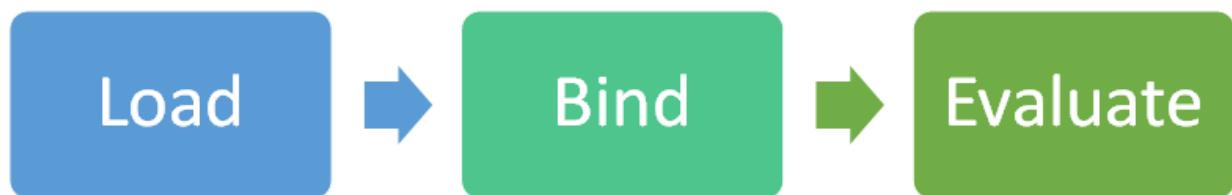
Use the following resources for help with Windows ML:

- 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](#).

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

Article • 12/30/2021

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):

```
C++

#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:

```
C++
```

```
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:

C++

```
// 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:

C++

```
// 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:

C++

```
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:

C++

```
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:

C++

```
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:

C++

```
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:

C++

```

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:

C++

```
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.

C++

```
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 input 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:

C++

```
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".

C++

```
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:

C++

```
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:

C++

```

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:

```
C++
```

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

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

```
C++
```

```
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:

```
sh
```

```
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 ↗](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Port an existing Windows ML app to NuGet package (C++)

Article • 12/30/2021

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 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 switch back and forth between using in-box Windows ML and the NuGet package.

C++

```
#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.

```
c++  
  
#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`.

Save Build Configurations

Right-click on the solution in the **Solution Explorer** and select **Properties**. In the **Properties** window, select **Configuration Manager**. Open the drop-down menu for **Active solution configuration** and choose **<New...>**. Enter the name of the new solution configuration and make sure **Create new project configurations** is checked. Now, preprocessor definitions can be saved in desired build configurations.

Build and run

Your application now successfully uses the WinML NuGet Package.

Select an execution device

Article • 12/30/2021

With Windows ML APIs, you can select a device you prefer to evaluate your model on. You select a device when you create a session. To do so, you will use the `LearningModelDevice` class, which defines the device used to evaluate the machine learning model.

Here, we'll make changes in the `classifier.cs` file of the previous app, so we can run our model evaluation on our device's GPU. We didn't touch that file in the previous tutorial, as it was automatically generated by WinML Code Generator.

1. Open the `classifier.cs` file and add the following code into the `CreateFromStreamAsync` method.

C#

```
// Select GPU or another DirectX device to evaluate the model.  
LearningModelDevice device = new  
LearningModelDevice(LearningModelDeviceKind.DirectX);
```

To choose a different execution device, simply switch the `DirectX` field to a different one.

2. Now, you can select this device when you create a session. Change the `LearningModelSession` constructor to specify the execution device.

C#

```
// Create the evaluation session with the model and device.  
learningModel.session = new LearningModelSession(learningModel.model,  
device);
```

The `CreateFromStreamAsync` method now as follows:

C#

```
CreateFromStreamAsync(IRandomAccessStreamReference stream)  
{  
    classifierModel learningModel = new classifierModel();  
    learningModel.model = await  
    LearningModel.LoadFromStreamAsync(stream);  
  
    // Select GPU or another DirectX device to evaluate the model.  
    LearningModelDevice device = new  
    LearningModelDevice(LearningModelDeviceKind.DirectX);
```

```
// Create the evaluation session with the model and device.  
learningModel.session = new  
LearningModelSession(learningModel.model, device);  
learningModel.binding = new  
LearningModelBinding(learningModel.session);  
return learningModel;  
}
```

That's all! You successfully selected the GPU for model evaluation.

If you do not 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.

Windows AI supports using a device that the caller has already created. To learn more about advanced device creation, review the [Create a session](#) documentation.

 **Note**

To learn more about how you select your execution device, please review the [LearningModelDevice](#) documentation.

Use encrypted models with Windows ML

Article • 12/30/2021

There are several static methods we can apply on the `LearningModel` class to load machine learning models such as load the model from a file in your app, from a file on disk or load a model from a stream.

Loading from a stream method allows better control over the model. In this case, you can choose to have the model encrypted on disk and decrypt it only in memory prior to calling one of the `LoadFromStream` methods.

In this tutorial, you will learn how to integrate an encrypted machine learning model with Windows ML application (C#).

Windows ML APIs do not provide machine learning encryption service and will not be held responsible for any damage or loss of any kind.

Get ONNX model

In this tutorial, you will use SqueezeNet model in ONNX format to perform the encryption, decryption and load from the stream.

Download or clone the [SqueezeNet Object Detection sample app](#) from GitHub to get the `SqueezeNet.onnx` model.

Specify the required declarations and variables

1. Copy the below using statements declarations to get access to all the APIs that you'll need:

C#

```
using System;
using System.Threading.Tasks;
using Windows.AI.MachineLearning;
using Windows.Security.Cryptography;
using Windows.Security.Cryptography.Core;
using Windows.Storage;
using Windows.Storage.Streams;
using Windows.UI.Xaml.Controls;
```

You'll define two special variables for a key and an initialization vector.

The *key* is a variable of the [CryptographicKey class](#), which represents a symmetric (or an asymmetric) key pair. You will need an object from this class since you will use the [AsymmetricAlgorithmProvider](#) method to create or import keys.

The *initialization vector* is a variable of the [IBuffer class](#), which represents a referenced array of bytes used by byte stream read and write interfaces.

Both [CryptographicKey](#) and [IBuffer](#) class variables are used to encrypt and decrypt the stream.

3. Add the following variable declarations and the [MainPage](#) class after the using statements inside the crypto namespace:

C#

```
namespace crypto
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a
    Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        private CryptographicKey _key;
        private IBuffer _initialization_vector;
        public MainPage()
        {
            this.InitializeComponent();

            Run();
        }
    }
}
```

Encrypt the Model

Windows APIs provide a rich set of features and capabilities, which can enhance the functionality of the Windows ML API set. Here, you'll use an encryption and decryption service provided in the Windows API set to produce a stream in memory, and use Windows ML APIs to load the model from that stream.

You can use any encryption service to encrypt your machine learning model, per your convenience. In this tutorial, we'll use the encryption method – [SymmetricAlgorithmNames](#) – [.AesCbcPkcs7](#).

The `SymmetricAlgorithmNames` class helps you to retrieve symmetric key algorithms to apply symmetric key encryption on your model. This type of encryption requires that the same key used for encryption also be used for decryption.

Using the `SymmetricKeyAlgorithmProvider` class, you can select an algorithm and create your key. In this tutorial, we'll use the `.AesCbcPkcs7` algorithm.

The `AES_CBC_PKCS7` algorithm represents an advanced Encryption Standard (AES) algorithm, coupled with a cipher-block chaining mode of operation and PKCS#7 padding.

1. The code below shows how to generate the key and encrypt the model.

C#

```
async Task<IBuffer> EncryptAsync(StorageFile model_file)
{
    // get a buffer for the model file
    var file_buffer = await
Windows.Storage.FileIO.ReadBufferAsync(model_file);

    // set up the encryption algorithm
    var algorithm =
SymmetricKeyAlgorithmProvider.OpenAlgorithm(SymmetricAlgorithmNames.AesCbcPk
cs7);
    uint key_length = 32;
    var key_buffer = CryptographicBuffer.GenerateRandom(key_length);
    _key = algorithm.CreateSymmetricKey(key_buffer);
    _initialization_vector =
CryptographicBuffer.GenerateRandom(algorithm.BlockLength);

    // perform the encryption
    var encrypted_buffer = CryptographicEngine.Encrypt(_key, file_buffer,
_initialization_vector);

    return encrypted_buffer;
}
```

[NOTE!] Interested in learning more about Cryptographic Keys? Please review the [Cryptographic keys documentation](#).

Decrypt the Model and Load from the stream

Before loading the model, you need to decrypt it using the `CryptographicEngine.Decrypt` method. `CryptographicEngine.Decrypt` is a method to decrypt content that was previously encrypted by using a symmetric or asymmetric

algorithm. When calling the method, you'll need to provide the previously generated key.

To access the decrypted model, you'll use the `InMemoryRandomAccessStream` class, which provides random access of data in input and output streams stored in memory instead of on disk.

As the last step, you'll create a session to load the model from the stream, using the `LearningModel.LoadFromStreamAsync` method. You can call this method as a synchronous or asynchronous task.

The code below shows how to decrypt the model using the generated key, write it to a stream, and then load the model from the stream.

C#

```
async Task DecryptAndRunAsync(IBuffer encryptyed_buffer)
{
    // decrypt the buffer
    var decrypted_buffer = CryptographicEngine.Decrypt(_key,
    encryptyed_buffer, _initialization_vector);

    // write it to a stream
    var decrypted_stream = new InMemoryRandomAccessStream();
    await decrypted_stream.WriteAsync(decrypted_buffer);

    // load the model from the stream
    var model = await
    LearningModel.LoadFromStreamAsync(RandomAccessStreamReference.CreateFromStre
    am(decrypted_stream));

    // create a session
    var session = new LearningModelSession(model);
}
```

Run the model

Copy the following Run method to call the methods defined earlier.

C#

```
async Task Run()
{
    // get the model file
    var model_file = await StorageFile.GetFileFromApplicationUriAsync(new
    Uri($"ms-appx:///Assets/SqueezeNet.onnx"));
```

```
// encrypt the model file.  
var encryptyed_buffer = await EncryptAsync(model_file);  
  
// decrypt the model file and load it  
await DecryptAndRunAsync(encryptyed_buffer);  
}
```

Summary

That's it! You have successfully loaded the model to your Windows ML app.

After you have loaded the model, you can continue to create a session, bind model inputs and outputs, and evaluate the model to complete your Windows ML app.

Additional Resources

To learn more about topics mentioned in this tutorial, visit the following resources:

- [Cryptographic keys documentation](#)
- [CryptographicKey class](#)
- [CryptographicEngine.Decrypt method](#)
- [SymmetricAlgorithmNames class](#)

Automatic code generation with **mlgen**

Article • 04/17/2024

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.

Getting the tool

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](#).

Using **mglen**

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.

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](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

WinML Dashboard

Article • 12/30/2021

WinML Dashboard is a tool for viewing, editing, converting, and validating machine learning models for Windows ML inference engine. The engine is built into Windows 10 and evaluates trained models locally on Windows devices using hardware optimizations for CPU and GPU to enable high performance inferences.

Getting the tool

You can [download the WinML Dashboard here](#), or you can build the app from source following the instructions below.

Build from source

When building the app from source, you'll need the following:

[+] Expand table

Requirements	Version	Download	Command to check
Python3	3.4+	here	<code>python --version</code>
Yarn	latest	here	<code>yarn --version</code>
Node.js	latest	here	<code>node --version</code>
Git	latest	here	<code>git --version</code>
MSBuild	latest	here	<code>msbuild -version</code>
Nuget	latest	here	<code>nuget help</code>

All six prerequisites *must be added to the Environment Path*. Note that MSBuild and Nuget will be included in a Visual Studio 2017 installation.

Steps to build and run

To run the WinML Dashboard, follow these steps:

1. In the command line, clone the repository: `git clone https://github.com/Microsoft/Windows-Machine-Learning`

2. Within the repository, enter the following to access the right folder: `cd Tools/WinMLDashboard`
3. Run `git submodule update --init --recursive` to update Netron.
4. Run `yarn` to download dependencies.
5. Then, run `yarn electron-prod` to build and start the desktop application, which will launch the Dashboard.

All available Dashboard commands can be seen at [package.json](#).

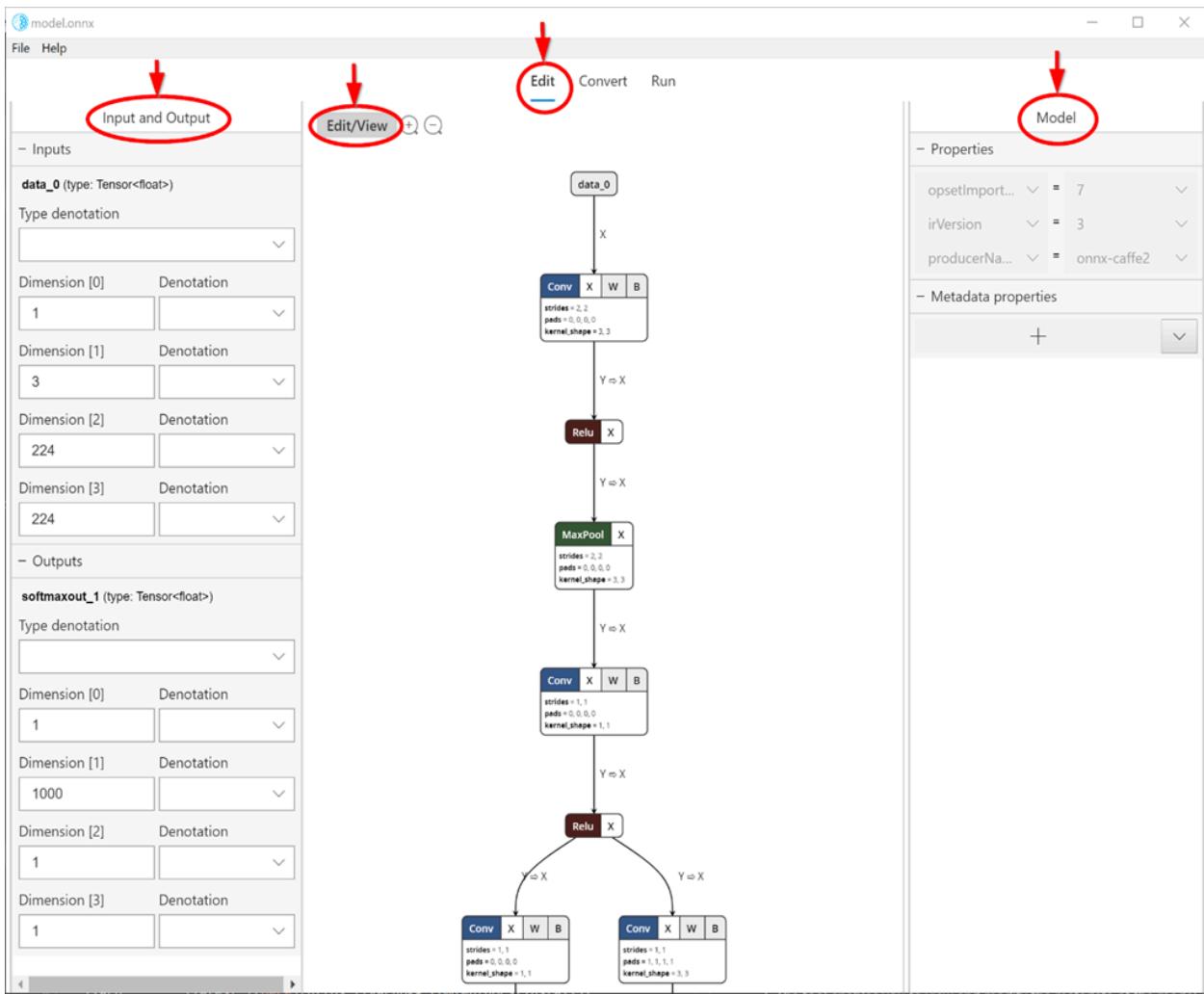
Viewing and Editing Models

The Dashboard uses [Netron](#) for viewing machine learning models. Although WinML uses ONNX format, the Netron viewer supports viewing several different framework formats.

Many times a developer may need to update certain model metadata or modify model input and output nodes. This tool supports modifying model properties, metadata and input/output nodes of an ONNX model.

Selecting the `Edit` tab (center top as shown in the snip below) takes you to viewing and editing panel. The left pane in the panel allows editing model input and output nodes, and the right pane allows editing Model properties. The center portion shows the graph. At this time, editing support is limited to model input/output node (and not inner nodes), model properties and model metadata.

The `Edit/View` button switches from *Edit mode* to *View-only mode*, and vice versa. *View-only mode* doesn't allow editing and enables Netron viewer's native features such as the ability to see detailed information for each node.



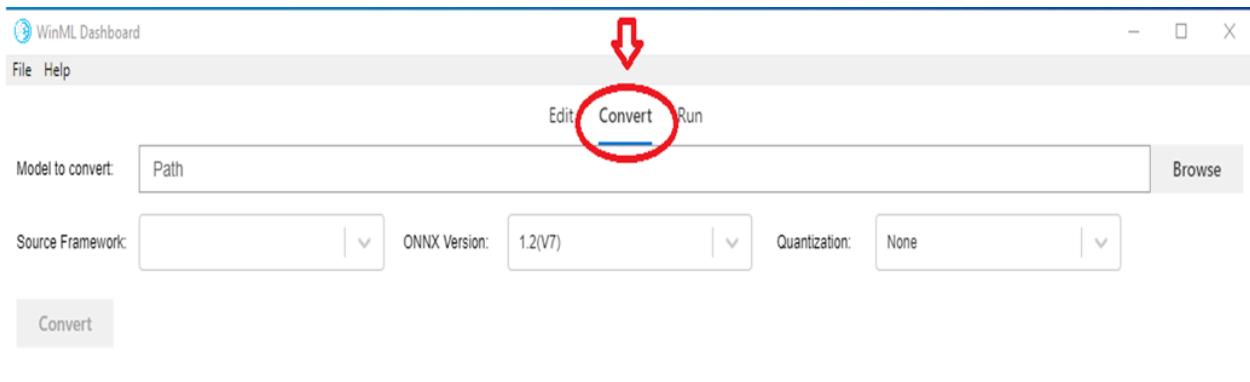
Converting models

Today there are several different frameworks available for training and evaluating machine learning models, which makes it difficult for app developers to integrate models into their product. Windows ML uses the [ONNX machine learning model format](#) that allows conversion from one framework format to another, and this Dashboard makes it easy to convert models from different frameworks to ONNX.

The Convert tab supports converting to ONNX from the following source frameworks:

- Apple Core ML
- TensorFlow (subset of models convertible to ONNX)
- Keras
- Scikit-learn (subset of models convertible to ONNX)
- Xgboost
- LibSVM

The tool also allows validation of the converted model by evaluating the model with built-in Windows ML inference engine using synthetic data (default) or real input data on CPU or GPU.

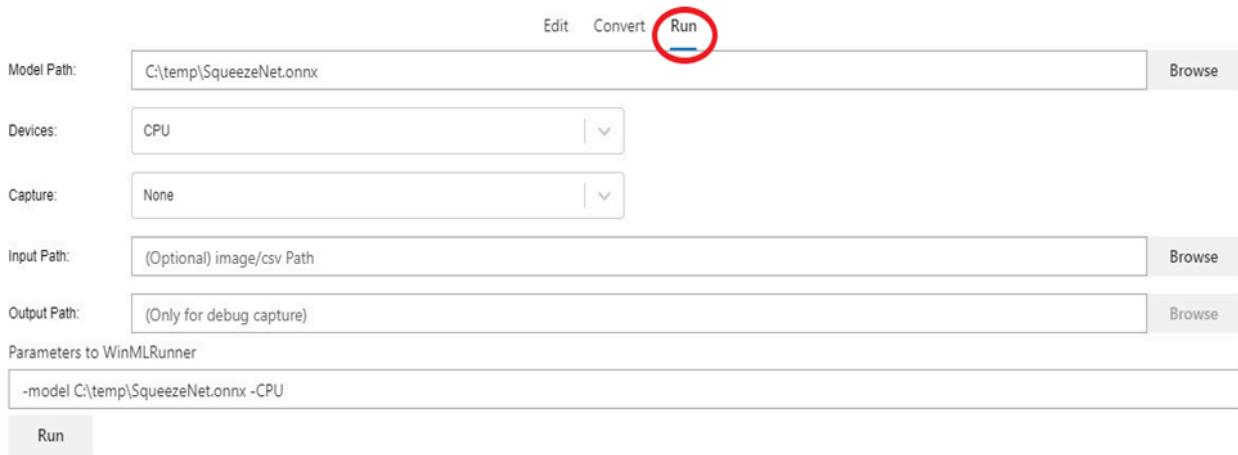


Validating Models

Once you have an ONNX model, you can validate whether the conversion has happened successfully and that the model can be evaluated in Windows ML inference engine. This is done using the `Run` tab (see snip below).

You can choose various options such as CPU (default) vs GPU, real input vs synthetic input (default) etc. The result of model evaluation appears in the console window at the bottom.

Note that model validation feature is only available on [Windows 10 October 2018 Update](#) or newer version of Windows 10, as the tool relies on built-in Windows ML inference engine.



```
- Console output

WinML Runner
GPU: AMD Radeon Pro WX 3100

Loading model (path = C:\temp\SqueezeNet.onnx)...
=====
Name: squeezeNet_old
Author: onnx-caffe2
Version: 9223372036854775807
Domain:
Description:
Path: C:\temp\SqueezeNet.onnx
Support FP16: false

Input Feature Info:
Name: data_0
Feature Kind: Float

Output Feature Info:
Name: softmaxout_1
Feature Kind: Float

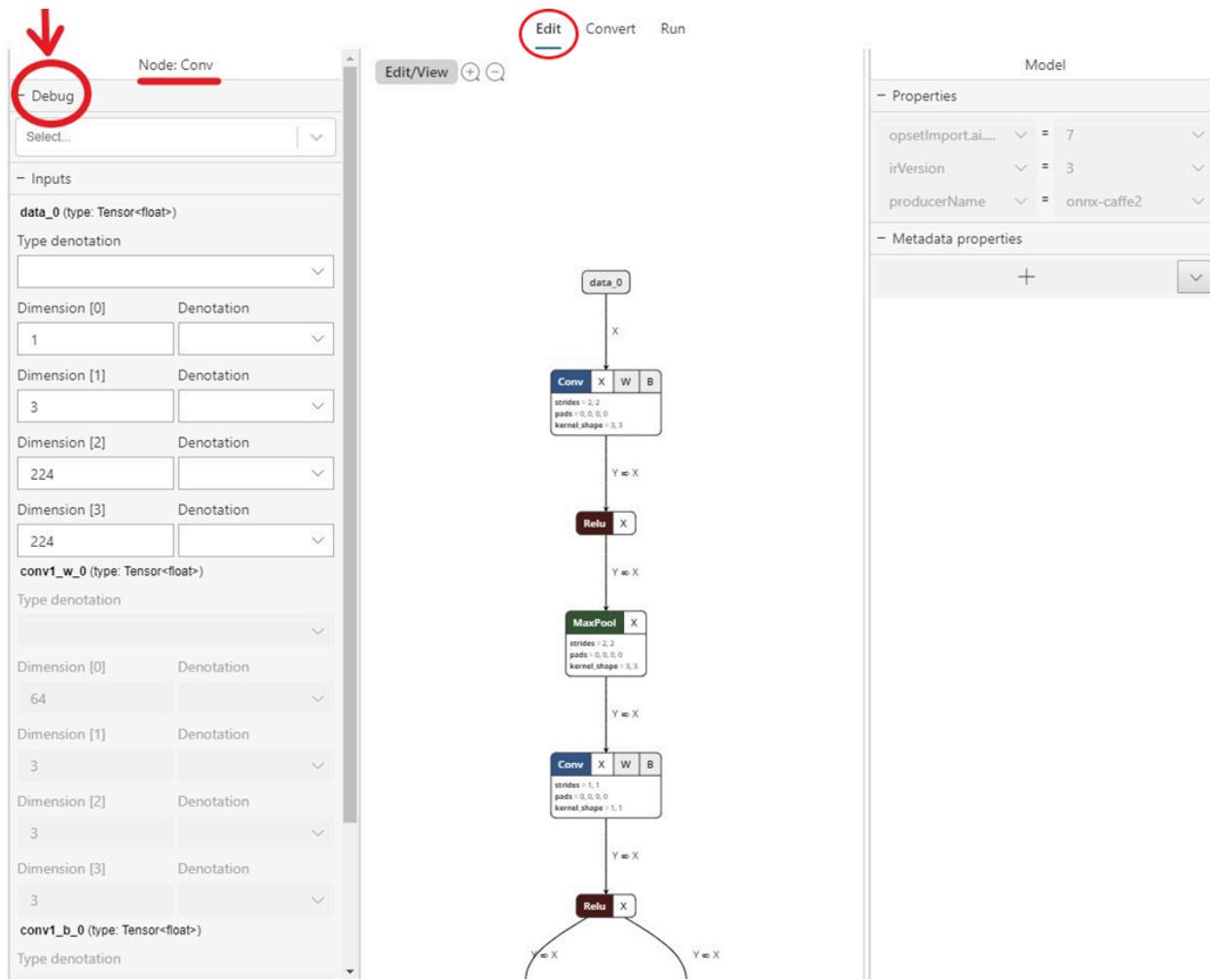
=====
Binding (device = CPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation = WinML)...[SUCCESS]
Evaluating (device = CPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation = WinML)...[SUCCESS]
Outputting results.
Feature Name: softmaxout_1
resultVector[111] has the maximal value of 0.120487
```

Debugging Inference

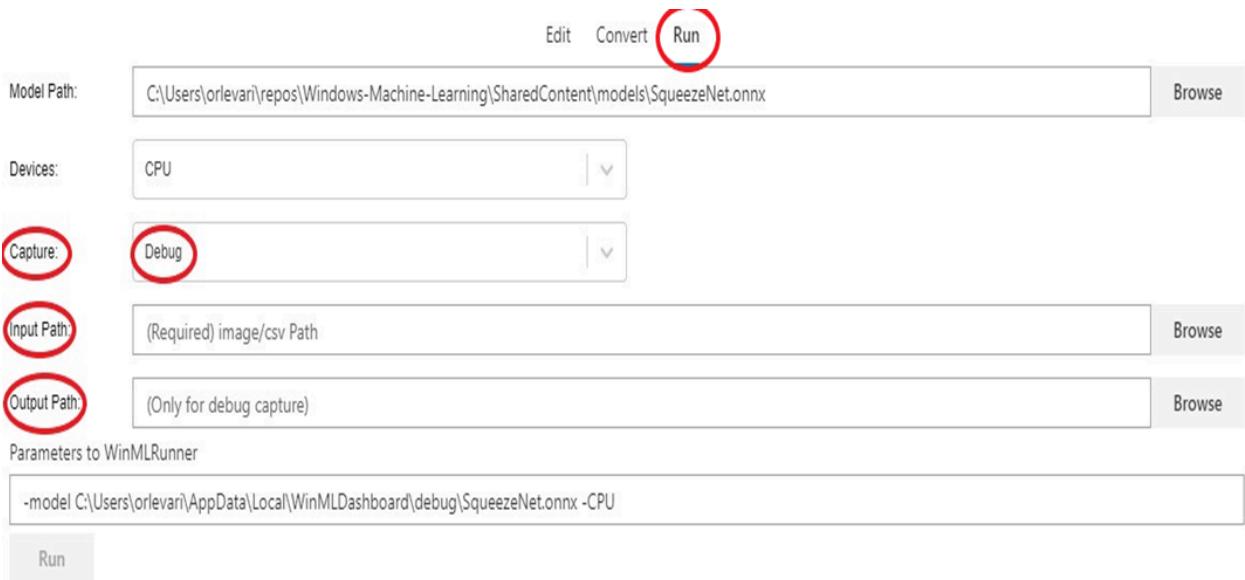
You can utilize the debug feature of WinML Dashboard to gain insight into how raw data is flowing through operators in your model. You can also choose to visualize this data for computer vision inference.

To debug your model follow these steps:

1. Navigate to the `Edit` tab and select the operator for which you wish to capture intermediate data. On the left side panel, there will be a `Debug` menu where you can select the formats of intermediate data you wish to capture. The options are currently **text** and **PNG**. **Text** will output a text file containing the dimensions, data type and raw tensor data produced by this operator. **PNG** will format this data into an image file which can be useful for computer vision applications.



2. Navigate to the **Run** tab and select the model you wish to debug.
3. For the **Capture** field, select **Debug** from the dropdown.
4. Select an input image or csv to supply to your model at execution. Note that this is required when capturing Debug data.
5. Select an output folder to export debug data.
6. Select **Run**. Once execution is complete you can navigate to this selected folder to view your Debug capture.



You can also open debug view in the Electron app with one of the following options:

- Run it with `flag --dev-tools`
- Or select `View -> Toggle Dev Tools` in the application menu
- Or press `Ctrl + Shift + I`.

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

WinMLRunner

Article • 02/13/2025

WinMLRunner is a tool to test if a model runs successfully when evaluated with the Windows ML APIs. You can also capture evaluation time and memory usage on the GPU and/or CPU. Models in .onnx or .pb format can be evaluated where the input and output variables are tensors or images. There are 2 ways you can use WinMLRunner:

- [Download the command-line python tool ↗](#).
- Use within the [WinML Dashboard](#). For more information see [the WinML Dashboard documentation](#)

Run a Model

First, open the downloaded Python tool. Navigate to the folder containing WinMLRunner.exe, and run the executable as shown below. Make sure to replace the installation location with what matches yours:

```
.\WinMLRunner.exe -model SqueezeNet.onnx
```

You can also run a folder of models, with a command such as the following.

```
WinMLRunner.exe -folder c:\data -perf -iterations 3 -CPU` \  
  
```

Running a good model

Below is an example of running a model successfully. Notice how first the model loads and outputs model metadata. Then the model runs on the CPU and GPU separately, outputting the binding success, evaluation success, and model output.

```
.\WinMLRunner.exe -model SqueezeNet.onnx
WinML Runner
GPU: AMD Radeon Pro WX 3100

Loading model (path = SqueezeNet.onnx)...
=====
Name: squeezeNet_old
Author: onnx-caffe2
Version: 9223372036854775807
Domain:
Description:
Path: SqueezeNet.onnx
Support FP16: false

Input Feature Info:
Name: data_0
Feature Kind: Float

Output Feature Info:
Name: softmaxout_1
Feature Kind: Float

=====
Binding (device = CPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor)...[SUCCESS]
Evaluating (device = CPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor)...[SUCCESS]
Outputting results..
Feature Name: softmaxout_1
resultVector[818] has the maximal value of 1

Binding (device = GPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor)...[SUCCESS]
Evaluating (device = GPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor)...[SUCCESS]
Outputting results..
Feature Name: softmaxout_1
resultVector[818] has the maximal value of 1
```

Running a bad model

Below is an example of running a model with incorrect parameters. Notice the FAILED output when evaluating on GPU.

```
=====
Name: bad_model
Author:
Version: 0
Domain: onnxml
Description:
Path: C:\bad_model.onnx
=====

Loading model...[SUCCESS]
Binding Model on GPU...[SUCCESS]
Evaluating Model on GPU...[FAILED]
c:\winml\ dll\ mloperatorauthorimpl.cpp(1081)\Windows.AI.MachineLearning.dll!00007FFEBC308C37: (caller: 00007FFEBC308C37) Exception(4) :
[winrt::Windows::AI::MachineLearning::implementation::AbiOpKernel::Compute::<lambda_0245d72c2a362d137084c22c5297e48a>::operator ()
```

Device selection and optimization

By default, the model runs on the CPU and GPU separately, but you can specify a device with a -CPU or -GPU flag. Here is an example of running a model 3 times using only the CPU:

```
WinMLRunner.exe -model c:\data\concat.onnx -iterations 3 -CPU
```

Log performance data

Use the `-perf` flag to capture performance data. Here is an example of running all the models in the data folder on the CPU and GPU separately 3 times and capturing performance data:

```
WinMLRunner.exe -folder c:\data iterations 3 -perf
```

Performance measurements

The following performance measurements will be output to the command-line and .csv file for each load, bind, and evaluate operation:

- **Wall-clock time (ms)**: the elapsed real time between the beginning and end of an operation.
- **GPU time (ms)**: time for an operation to be passed from CPU to GPU and execute on the GPU (note: Load() is not executed on the GPU).
- **CPU time (ms)**: time for an operation to execute on the CPU.
- **Dedicated and Shared Memory Usage (MB)**: Average kernel and user-level memory usage (in MB) during evaluate on the CPU or GPU.
- **Working Set Memory (MB)**: The amount of DRAM memory that the process on the CPU required during evaluation. Dedicated Memory (MB) - The amount of memory that was used on the VRAM of the dedicated GPU.
- **Shared Memory (MB)**: The amount of memory that was used on the DRAM by the GPU.

Sample performance output:

```

.\WinMLRunner.exe -model SqueezeNet.onnx -perf
WinML Runner
Printing available GPUs with DXGI..
Index: 0, Description: AMD Radeon Pro WX 3100

Loading model (path = .\SqueezeNet.onnx)...
=====
Name: squeezenet_old
Author: onnx-caffe2
Version: 9223372036854775807
Domain:
Description:
Path: .\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]

Results (device = CPU, numIterations = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation = WinML):
First Iteration Performance (load, bind, session creation, and evaluate):
Load: 436.598 ms
Bind: 0.8575 ms
Session Creation: 120.181 ms
Evaluate: 177.233 ms

Working Set Memory usage (evaluate): 9.95313 MB
Working Set Memory usage (load, bind, session creation, and evaluate): 45.6289 MB
Peak Working Set Memory Difference (load, bind, session creation, and evaluate): 46.5625 MB

Dedicated Memory usage (evaluate): 0 MB
Dedicated Memory usage (load, bind, session creation, and evaluate): 0 MB

Shared Memory usage (evaluate): 0 MB
Shared Memory usage (load, bind, session creation, and evaluate): 0 MB


Creating Session with GPU: AMD Radeon Pro WX 3100
Binding (device = GPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation = WinML)...[SUCCESS]
Evaluating (device = GPU, iteration = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation = WinML)...[SUCCESS]

Results (device = GPU, numIterations = 1, inputBinding = CPU, inputDataType = Tensor, deviceCreationLocation = WinML):
First Iteration Performance (load, bind, session creation, and evaluate):
Load: 436.598 ms
Bind: 5.1858 ms
Session Creation: 285.041 ms
Evaluate: 25.7202 ms

Working Set Memory usage (evaluate): 1.21484 MB
Working Set Memory usage (load, bind, session creation, and evaluate): 42.8047 MB
Peak Working Set Memory Difference (load, bind, session creation, and evaluate): 44.1152 MB

Dedicated Memory usage (evaluate): 10.082 MB
Dedicated Memory usage (load, bind, session creation, and evaluate): 15.418 MB

Shared Memory usage (evaluate): 1 MB
Shared Memory usage (load, bind, session creation, and evaluate): 6.04688 MB

```

Test sample inputs

Run a model on the CPU and GPU separately, and by binding the input to the CPU and the GPU separately (4 total runs):

```
WinMLRunner.exe -model c:\data\SqueezeNet.onnx -CPU -GPU -CPUBoundInput -  
GPUBoundInput
```

Run a model on the CPU with the input bound to the GPU and loaded as an RGB image:

```
WinMLRunner.exe -model c:\data\SqueezeNet.onnx -CPU -GPUBoundInput -RGB
```

Capturing Trace Logs

If you want to capture trace logs using the tool, you can use logman commands in conjunction with the debug flag:

```
logman start winml -ets -o winmllog.etl -nb 128 640 -bs 128logman update  
trace winml -p {BCAD6AEE-C08D-4F66-828C-4C43461A033D} 0xffffffffffffffffffff  
0xff -ets WinMLRunner.exe -model C:\Repos\Windows-Machine-  
Learning\SharedContent\models\SqueezeNet.onnx -debuglogman stop winml -ets
```

The `winmllog.etl` file will appear in the same directory as the `WinMLRunner.exe`.

Reading the Trace Logs

Using the `traceprt.exe`, run the following command from the command-line.

```
traceprt.exe winmllog.etl -o logdump.csv -of CSV
```

Next, open the `logdump.csv` file.

Alternately, you can use the Windows Performance Analyzer (from Visual Studio). Launch the Windows Performance Analyzer, and open `winmllog.etl`.

```

Required command-line arguments:
-model <path>           : Fully qualified path to a .onnx or .pb model file.
    or
-folder <path>          : Fully qualified path to a folder with .onnx and/or .pb models, will run all of the models in the folder.

#Optional command-line arguments:
-version: prints the version information for this build of WinMLRunner.exe
-CPU : run model on default CPU
-GPU : run model on default GPU
-GPUHighPerformance : run model on GPU with highest performance
-GPUMinPower : run model on GPU with the least power
-GPUAdapterName <adapter name substring>; run model on GPU specified by its name. NOTE: Please only use this flag on DXCore supported
>CreateDeviceOnClient : create the D3D device on the client and pass it to WinML to create session
>CreateDeviceInWinML : create the device inside WinML
-CPUBoundInput : bind the input to the CPU
-GPUBoundInput : bind the input to the GPU
-RGB : load the input as an RGB image
-BGR : load the input as a BGR image
-Tensor [function] : load the input as a tensor, with optional function for input preprocessing
    Optional function arguments:
        Identity(default) : No input transformations will be performed.
        Normalize <scale> <means> <stddevs> : float scale factor and comma separated per channel means and stddev for normalization.
-Perf [all]: capture performance measurements such as timing and memory usage. Specifying "all" will output all measurements
-Iterations : # times perf measurements will be run/averaged. (maximum: 1024 times)
-Input <path to input file>; binds image or CSV to model
-InputImageFolder <path to directory of images> : specify folder of images to bind to model" << std::endl;
-TopK <number>; print top <number> values in the result. Default to 1
-BaseOutputPath [<fully qualified path>] : base output directory path for results, default to cwd
-PerfOutput [<path>] : fully qualified or relative path including csv filename for perf results
-SavePerIterationPerf : save per iteration performance results to csv file
-PerIterationPath <directory_path> : Relative or fully qualified path for per iteration and save tensor output results. If not speci-
-SaveTensorData <saveMode>; saveMode: save first iteration or all iteration output tensor results to csv file [First, All]
-DebugEvaluate: Print evaluation debug output to debug console if debugger is present.
-Terse: Terse Mode (suppresses repetitive console output)
-AutoScale <interpolationMode>; Enable image autoscaling and set the interpolation mode [Nearest, Linear, Cubic, Fant]
-GarbageData.MaxValue <maxValue>; Limit generated garbage data to a maximum value. Helpful if input data is used as an index.

Concurrency Options:
-ConcurrentLoad: load models concurrently
-NumThreads <number>; number of threads to load a model. By default this will be the number of model files to be executed
-ThreadInterval <milliseconds>; interval time between two thread creations in milliseconds

```

Note that -CPU, -GPU, -GPUHighPerformance, -GPUMinPower -BGR, -RGB, -tensor, -CPUBoundInput, -GPUBoundInput are not mutually exclusive (i.e. you can combine as many as you want to run the model with different configurations).

Dynamic DLL Loading

If you want to run WinMLRunner with another version of WinML (e.g. comparing the performance with an older version or testing a newer version), simply place the windows.ai.machinelearning.dll and directml.dll files in the same folder as WinMLRunner.exe. WinMLRunner will look for these DLLs first and fall back to C:/Windows/System32 if it doesn't find them.

Known issues

- Sequence/Map inputs are not supported yet (the model is just skipped, so it doesn't block other models in a folder);
- We cannot reliably run multiple models with the -folder argument with real data. Since we can only specify 1 input, the size of the input would mismatch with most

of the models. Right now, using the -folder argument only works well with garbage data;

- Generating garbage input as Gray or YUV is not currently supported. Ideally, WinMLRunner's garbage data pipeline should support all inputs types that we can give to winml.
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ONNX Conversion (ONNXMLTools)

Article • 12/30/2021

ONNXMLTools enables you to convert models from different machine learning toolkits into [ONNX](#).

Installation and use instructions are available [at the ONNXMLTools GitHub repo](#).

Support

Currently, the following toolkits are supported.

- Keras (a wrapper of [keras2onnx converter](#))
- Tensorflow (a wrapper of [tf2onnx converter](#))
- scikit-learn (a wrapper of [skl2onnx converter](#))
- Apple Core ML
- Spark ML (experimental)
- LightGBM
- libscm
- XGBoost
- H2O
- CatBoost

Pytorch also has a build-in ONNX exporter. [Check here](#) for further details.

ⓘ 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](#).

Windows Machine Learning samples

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.

Samples

The following sample applications are available on GitHub.

[\[+\] Expand table](#)

Name	Description
AdapterSelection (Win32 C++)	A desktop application that demonstrates how to choose a specific device adapter for running your model.
BatchSupport	Shows how to bind and evaluate batches of inputs with Windows ML.
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 Windows ML 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 Windows ML.
Emoji8 (UWP C#)	Shows how you can use Windows ML 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.
NamedDimensionOverrides	Demonstrates how to override named dimensions to concrete values in order to optimize model performance.
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.

Name	Description
RustSqueezeNet ↗	Rust projection of WinRT using SqueezeNet.
SqueezeNet Object Detection (Win32 C++, UWP C#/JavaScript, .NET5, .NETCORE) ↗	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.
StreamFromResource ↗	Shows how to take an embedded resource that contains an ONNX model and convert it to a stream that can be passed to the LearningModel constructor.
StyleTransfer ↗ (C#)	A UWP app which performs style transfer on user-provided input images or web camera streams.
winml_tracker (ROS C++) ↗	A ROS (Robot Operating System) node which uses Windows ML 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 ↗](#).

Last updated on 02/13/2025

Integrate a model into your app with Windows ML

Article • 02/13/2025

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.

<https://www.youtube-nocookie.com/embed/Nc2wstifyoE>

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 `<CppWinREnabled>true</CppWinREnabled>` property to the project's .vcxproj file:

```
XML

<Project ...>
  <PropertyGroup Label="Globals">
    <CppWinREnabled>true</CppWinREnabled>
  ...

```

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](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Load a model

Article • 12/30/2021

ⓘ 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:

C#

```
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](#).

Create a session

Article • 12/30/2021

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.

<https://www.youtube-nocookie.com/embed/NM5CYUMMp-w>

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:

C#

```
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](#).

Bind a model

Article • 12/30/2021

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.

[https://www.youtube-nocookie.com/embed/WD5bNZwz2A8 ↗](https://www.youtube-nocookie.com/embed/WD5bNZwz2A8)

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:

C#

```
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.
 - o 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.
 - o NominalRange_0_255 will be assumed
 - o SRGB will be assumed

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

- o **BitmapBounds**: If specified, these are the cropping boundaries to apply prior to sending the image to the model.
- o **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*.

```
C#
```

```
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](#).

Evaluate the model inputs

Article • 12/30/2021

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:

C#

```
// 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;
    });
}
```

```

        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](#).

Windows ML performance and memory

Article • 12/30/2021

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 [ONNXMLTools](#) 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 ↗](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Executing multiple ML models in a chain

Article • 12/30/2021

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.

C++

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

cs

```
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.

C++

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

cs

```
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:

C++

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

cs

```
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".

C++

```
//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");
}

```

CS

```

//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:

C++

```

//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);
```

CS

```
//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:

C++

```
//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"");
```

CS

```
//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.

C++

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

CS

```
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 ↗](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Release notes

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

ⓘ Important

For the latest documentation about Windows Machine Learning, see [What is Windows ML](#). That documentation describes APIs that are in the `Microsoft.Windows.AI.MachineLearning` namespace, which ships in the Windows App SDK. Those APIs supersede the ones documented here, which are in the `Windows.AI.MachineLearning` namespace, and were shipped in 2018.

Windows ML NuGet Package - Version 1.9

- [Download NuGet here.](#)
- [Built on ONNX Runtime 1.9.](#)
- WinML - DLL dependency fix supports learning models on Windows 8.1.

Windows ML NuGet Package - Version 1.8

- [Download NuGet here.](#)
- [Built on ONNX Runtime 1.8.](#)
- New native WinML API, `SetIntraOpThreadSpinning`. This API is used to toggle IntraOp thread spin behavior. When enabled, and when there is no current workload, IntraOp threads will continue to spin for some additional time as it waits for additional work to complete. This can result in better performance for the current workload, but may impact performance of other unrelated workloads. This toggle is enabled by default.

Windows ML NuGet Package - Version 1.7

- [Download NuGet here](#)
- [Built on ONNX Runtime 1.7](#)
- .NET5 support - will work with .NET5 Standard 2.0 Projections.
- Image descriptors expose `NominalPixelRange` properties
- Native support added for additional pixel ranges [0..1] and [-1..1] in image models.
- A new property is added to the `ImageFeatureDescriptor` runtime class to expose the `ImageNominalPixelRange` property in `ImageFeatureDescriptor`. Other similar properties exposed are the image's `BitmapPixelFormat` and `BitmapAlphaMode`.

- Bug fixes and performance improvements.
- DirectML PIX markers to Redist added to enable profiling graph at operator level.
- Fixes applied to ensure the package correctly installs on C# UWP projects in Visual Studio.

Windows ML NuGet Package - Version 1.6

- [Download NuGet here ↗](#)
- [Built on ONNX Runtime 1.6 ↗](#)
- Support for UWP applications targeting Windows Store deployment for both CPU and GPU.
- WindowsAI Redist now includes a statically linked C-Runtime package for additional deployment options.
- Minor API Improvements: Users are now able to bind Iterable as inputs and outputs, and able to create Tensor* via multiple buffers.

Windows ML NuGet Package - Version 1.5

- Support for UWP applications targeting Windows Store deployment (CPU only).
- Support for .NET and .NET framework applications.
- Support for RUST Developers - [sample and documentation available ↗](#)
- New APIs to for additional performance control:
 - [IntraopNumThreads](#): Provides an ability to change the number of threads used in the threadpool for Intra Operator Execution for CPU operators through LearningModelSessionOptions.
 - [SetNamedDimensionOverrides]((/native-apis/SetNamedDimensionOverrides.md)): Provides the ability to override named input dimensions to concrete values through LearningModelSessionOptions in order to achieve better runtime performance.
- Support for additional ONNX format image type denotations – Gray8, normalized [0..1] and normalized [-1..1].
- Reduced package size by separating debug symbols into separate distribution 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](#).

Which AI solution is right for me?

Article • 02/13/2025

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 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.

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

FAQ (Frequently Asked Questions)

This page contains answers to the most popular questions from the community about Windows ML.

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 `onnxmiltools` instead of `winmltools`?

Yes, you can, but you will need to make sure you install the correct version of [onnxmiltools](#) 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](#).