Evaluation environment for Windows games 3.0 README

Overview

The evaluation environment for Windows games is now a subset of the expanded Game Porting Toolkit. The evaluation environment helps game developers try out their existing Windows games right on Apple Silicon Macs running macOS 15 Sequoia or higher. Using community projects that bundle the evaluation environment, or building your own environment using the included binaries and installation and configuration instructions, you can run your game to get a sense of how it can feel and play right away, even before you begin your porting journey.

Requirements

- The evaluation environment for Windows games only runs on Apple Silicon Macs running macOS 15 Sequoia or higher.
- Translated games require more resources, so developer-focused Macs with 16GB of RAM or more are recommended.
- The provided macOS graphics bridge libraries must be configured with a custom version of the Wine translation environment in order to create your evaluation environment.
- Instructions and scripts for building and configuring the custom version of Wine are included here.

Installation and Setup

Use a pre-built evaluation environment

While simple to use once configured and installed, the evaluation environment for Windows games can take time to configure and use correctly depending on your proficiency and comfort with command-line tools and your experience using translation tools like WINE. The following free and commercial products incorporate the supplemental evaluation layers from this distribution within a pre-built WINE environment.

- Dean Greer's (aka GCenX) **homebrew-wine** (https://github.com/Gcenx/homebrew-wine) and **game-porting-toolkit** (https://github.com/Gcenx/game-porting-toolkit) casks - use **`brew install --cask --no-quarantine gcenx/wine/<XYZZY>`** to easily install either of the more complete environments and the graphical translation layer.
- CodeWeaver's **CrossOver**. (https://www.codeweavers.com/crossover) CodeWeavers offers a 14-day free trial of CrossOver, which includes integration of the graphical translation layer.

Note: early in the macOS 26 Tahoe beta period these pre-built tools may still be carrying the prior version of D3DMetal. You can temporarily update these tools to use the latest version as follows.

- **homebrew-wine** and **game-porting-toolkit** casks - you can replace the copies of the evaluation environment libraries found at /Applications/Game\ Porting\ Toolkit.app/Contents/Resources/wine/lib/ with the libraries from this distribution:

  ```
  cd /Applications/Game\ Porting\ Toolkit.app/Contents/
  Resources/wine/lib

  mv external external.old; mv wine wine.old

  ditto "/Volumes/Evaluation environment for Windows
  games 3.0/redist/lib/" .
  ```

- **CrossOver**: replace CrossOver's copies of the evaluation environment libraries found at /Applications/CrossOver.app/Contents/SharedSupport/CrossOver/lib64/apple_gptk with the libraries from this distribution:

  ```
  cd /Applications/CrossOver.app/Contents/SharedSupport/
  CrossOver/lib64/apple_gptk

  mv external external.old; mv wine wine.old

  ditto "/Volumes/Evaluation environment for Windows
  games 3.0/redist/lib/" .
  ```

Prepare your environment

To enable experimental MetalFX integration, perform the following steps:

- Rename `wine/x86_64-unix/nvngx-on-metalfx.so` to `wine/x86_64-unix/nvngx.so` if this hasn't already been done
- Rename `wine/x86_64-windows/nvngx-on-metalfx.dll` to `wine/x86_64-windows/nvngx.dll` if this hasn't already been done
- Copy both `nvngx.dll` and `nvapi64.dll` to the `windows\system32` directory your Wine prefix's virtual C: drive (open `~/my-game-prefix/drive_c/windows/system32`)

Environment Variables

Environment variables can be used to control some aspects of translation and emulation in the evaluation environment.

`D3DM_SUPPORT_DXR` - Defaults to 0 (OFF) on M1 & M2 Macs, and to 1 (ON) for M3 & later Macs. Setting this environment variable to 1 (ON) enables DirectX Raytracing (aka DXR) features in D3DMetal's DirectX 12 translation layer, so games querying for DXR support will find the support level and expected interfaces of DXR.

`ROSETTA_ADVERTISE_AVX` - Defaults to 0 (OFF). On macOS 15 Sequoia and later, setting this environment variable to 1 (ON) causes the CPU instruction translation layer to publish cpuid information to translated applications when running in the evaluation environment, so games querying instruction set extension capabilities before utilizing them can conditionally control their use of instruction extensions. This setting does not modify the availability of the instruction set in Rosetta; it only controls whether the processor advertises its support for these extensions.

`D3DM_ENABLE_METALFX` - Defaults to 0 (OFF). On macOS 26 Tahoe, setting this environment variable to 1 (ON) causes DLSS functions to be converted to MetalFX where possible. Setting this environment variable to 0 (OFF) causes DLSS functions to be not be available.

Logging

Logging output will appear in the Terminal window in which you launch your game as well as the system log, which can be viewed with the Console app found in Applications ▸ Utilities. Log messages from the evaluation environment for Windows games are prefixed with **D3DM** and are logged to the system log using the "D3DMetal" category. If you are experiencing an issue and want to send logging information through https://feedbackassistant.apple.com, please attach and send the full logs without filtering to **D3DM.**

Debugging your game with Metal debugger

`Note: You will need to disable System Integrity Protection (SIP) (`https://developer.apple.com/documentation/security/disabling_and_enabling_system_integrity_protection) **`to debug CrossOver's Wine processes. Reenable SIP after you finish debugging.`**

- Compile your shaders with embedded debug information (https://developer.apple.com/metal/shader-converter/#shader) by passing `-Zi -Qembed_debug` to the DX Compiler.
- In CrossOver, select a bottle to launch your game from.
- Enable D3DMetal in the Advanced Settings for the bottle.
- Launch your game by clicking Run Command, choosing your game executable, and inserting the following environment variables to enable Metal debugging and

processing of debug information: **`MTL_CAPTURE_ENABLED=1 D3DM_DXIL_PROCESS_DEBUG_INFORMATION=1`**

- In Xcode, click `Debug > Debug Executable…` from the menubar and select CrossOver.app (this is just to get a workspace window open)
- In the visible Scheme options, click the `Options` tab and change `GPU Frame Capture` from `Automatically` to `Metal`.
- Close Scheme.
- Click `Debug > Attach to Process` from the menubar and select your launched game process.
- After the debugger attaches to the process, you can capture your Metal workload (https://developer.apple.com/documentation/xcode/capturing-a-metal-workload-in-xcode#Capture-your-Metal-workload-while-debugging).

If lldb suspends the process due to handling `SIGUSR1`, you will need to run the following commands to ignore this signal and continue the process:

```
process handle —pass false —stop false —notify false
SIGUSR1
continue
```

Troubleshooting

My game won't run and crashes with an invalid instruction or complains about lack of certain instruction extensions

Invalid instruction crashes are sometimes caused when the Rosetta 2 instruction translation layer is unable to translate CPU instructions. You may be able to recompile a version of your game without certain instructions in order to evaluate its potential on Apple Silicon with the Game Porting Toolkit when you hit this error. You may also be able to use the `ROSETTA_ADVERTISE_AVX` environment variable to ensure your game recognizes available translation instruction extensions. When porting your code natively to Apple Silicon there are a variety or NEON and ARM instructions which offer high-performance replacements for AVX / AVX2, BMI, F16c and other less common instruction set extensions.

My game won't run because its anti-cheat or DRM software is incompatible with Wine translation.

You may be able to rebuild a custom version of your game in your Windows development environment with anti-cheat or DRM disabled for your own evaluation purposes. When porting your code natively to Apple Silicon and macOS, contact your anti-cheat or DRM provider—most have native Apple Silicon solutions for your native build, or you may find that existing macOS solutions like Hardened Runtime, Application Sandbox, and Application Attestation prevent forms of cheating or tampering that concern you.

My game won't run because it thinks the version of Windows is too old.

First, make sure you have selected an appropriate Windows version in winecfg. This affects the major and minor Windows versions that are reported to your game.

If your game checks for a specific minimum or an exact build version, you can alter this value by changing the `CurrentBuild` and `CurrentBuildNumber` values of the `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT` registry key. You must perform this step *after* selecting a Windows version in winecfg.

My game won't run because it requires Mono, .NET, or the MSVCRT runtime.

The evaluation environment for Windows games does not pre-install these runtime support packages. If your game makes use of one of these packages, consider searching for and downloading appropriate installers (.exe or .msi) and installing them to your evaluation environment.

My game won't boot anymore even though I made no changes.

If the game stopped booting without being updated, you can try clearing the shader cache.
Run the following commands:

```
cd $(getconf DARWIN_USER_CACHE_DIR)/d3dm
cd «GAME_NAME»
rm -r shaders.cache
```

Do you have a different problem or other feedback?

Please let us know through https://feedbackassistant.apple.com.