Mono Development for Wine

A Cry for Help

Outline

- What is Wine Mono?
- Why does it matter?
- Current status?
- How to Develop
  - Building
  - Code Tree Overview
  - Using Changed Mono
  - Debugging
  - Sending Patches
  - Writing Tests

What is Wine Mono?

- Wine Mono is a fork of the Mono project, and a Win32 build of Mono packaged for the Wine Project.
- Differences from upstream are avoided whenever possible.
- Wine cannot use Linux Mono because it needs to call back into Wine.
- The package is a .msi file that Wine can automatically download and/or install, and that users can install (and uninstall) manually.
- The package also includes registry keys and files to prevent programs from installing/needing native .NET.
- Includes some “projects” not in the main Mono repo, such as VB.

Why does it matter?

- Using MS .NET means we rely on MS to keep the download up (MS has removed redist downloads in the past) and provide redists for future releases.
- The .NET EULA requires a Windows license.
- We can’t use .NET to port applications because of the EULA.
- (I can’t make a very convincing argument that the broader Wine community should care because .NET works with much less effort, and should almost always work at least as well as Mono, but I really don’t like to rely on MS components.)

Current Status

- Wine Mono's compatibility is practically 0. I have seen rare cases where it works for a normal Windows program that genuinely uses .NET.
- At the rate I am (and Alistair is) working on it, I don't expect much improvement in the next few years (though I keep hoping for a breakthrough in mixed-mode).
- There are easy problems in the Wine bugzilla right now that no one is working on (search for component=mscoree).

Building

- Fetch the code:
  - git clone –recursive git://github.com/madewokherd/wine-mono
- Install Wine, gmcs, and mingw-w64 compilers for both x86 and x86_64 targets. A 64-bit OS is not required, but old mingw may not work.
- Run build-winemono.sh (set MAKEOPTS=-j2 to use two cores)

Code Tree Overview

- All Windows .NET embedding API's are in the Wine code, in dlls/mscoree.
- The C parts of the Mono runtime are in mono/mono (hopefully shouldn't need to be touched much).
- Most .NET API's are in mono/mcs/class/*assembly*/*namespace*/*typename*.cs
  - Example: The System.Drawing.Drawing2D.GraphicsPath class is in mono/mcs/class/System.Drawing/System.Drawing.Drawing2D/GraphicsPath.cs
  - The namespace is everything before the last dot.
  - Usually the assembly name can be easily guessed based on the namespace.
  - System.Windows.Forms is in the Managed.Windows.Forms assembly.
  - Many classes under system are in the mscorlib assembly, which is located in the corlib directory.
- Visual Basic classes are in mono-basic.

Using Changed Mono

- To rebuild without starting from scratch, use build-winemono.sh with the -r switch.
- After running build-winemono.sh, install by running: msiexec /i winemono.msi
- If the version number hasn’t changed, uninstall with ‘wine uninstaller’ before installing the new msi file.
- Or you can copy individual files from image/ to drive_c/windows/mono/mono-2.0

Debugging

- Use the WINE_MONO_TRACE environment variable to trace managed code. Run mono --help-trace for documentation on the syntax.
- If WINE_MONO_TRACE is set, all exceptions will be printed.
- Usually this is useful, but some exceptions are benign. Mono's winforms usually causes an exception involving UIAutomation that can be ignored.
- To get a stack trace of an exception, use WINE_MONO_TRACE=E:System.NotImplementedException (or fill in the actual type).
- To trace everything that enters/exits managed code, use WINE_MONO_TRACE=wrapper.
- Be careful, this can trace private/internal functions.

Sending Patches

- Usually, changes should go to the appropriate upstream project, not Wine Mono.
- The Mono project is at http://github.com/mono, and they prefer github pull requests.
- See https://help.github.com/articles/using-pull-requests
- The gist is that you make your own fork of upstream's repo, push your change to your fork, then use the github website to make the request.
- You can also make pull requests on wine-mono and its forks, or ask me to merge some changes you need from upstream.

Writing Tests (1/2)

- Mono includes a test suite based on the nunit framework, which vaguely reflects .NET's behavior when it was last tested, if the API is from .NET.
- The official way to run the tests involves make (and Cygwin on Windows).
- Use build-winemono.sh with the -t switch to create a stand-alone version of the tests (in a directory named tests-net_version) that will run on any .NET runtime.
- To run tests from that directory, run nunit-console.exe with the dll file containing the tests you want.

Writing Tests (2/2)

- Test code is located in mono/mcs/class/<assembly>/Test
- Use the /run switch to run only one test.
- Use /fixture to run tests from a single class of tests.

Questions?