

# .NET on Linux (Mono)

Monday, May 16, 2011  
2:36 PM

Used to be owned by Novell. Now a separate company called Xamarin <http://www.xamarin.com/>

Miguel de Icaza main developer

Smaller team of devs. MonoTouch and MonoDroid products have an uncertain future at the moment. Xamarin has stated they will be developing competing / compatible products but they are 3 -4 months off.

Configuring Yum repo in Linux. This will allow you to "yum install" mono

Going to install this on Cent OS 5.5 (Basically the community version of RedHat. YMMV on different distros)

Get virtual Virtual Box images for free on SourceForge <http://sourceforge.net/projects/virtualboximage/>

Mono has RPM packages for the latest version (2.10) which make installing it simple with your RPM system of choice. CentOS uses YUM other distros will be slightly different.

## **First make sure that the YUM repo is properly configured**

Open the mono repo file

```
vi /etc/yum.repos.d/mono.repo
```

That file should contain the following:

```
[Mono]
name=Mono Stack (RHEL_5)
type=rpm-md
baseurl=http://ftp.novell.com/pub/mono/download-stable/RHEL_5/
gpgcheck=1
gpgkey=http://ftp.novell.com/pub/mono/download-stable/RHEL_5/repodata/repomd.xml.key
enabled=1
```

Save

**Remove any previous versions of mono** on the machine (for a clean install only)

```
yum remove mono-addon-core
```

That should remove everything since they are all dependent on mono core

**Verify all packages are removed by doing**

```
yum list | grep mono
```

Remove anything else that shows as installed

**Now you can Yum install** the various mono packages that are needed for 2.10

```
yum install mono-addon-core
```

And any additional packages you might want like WCF, WinFX Core, devel etc..

```
yum install mono-addon-winxcore
```

```
yum install mono-addon-wcf
```

```
yum install mono-addon-devel
```

Make sure your profile.d sets the mono path correctly

```
cd /etc/profile.d/
```

```
vi mono_path.sh
```

```
export PATH=$PATH:/opt/novell/mono/bin
```

```
export PKG_CONFIG_PATH=/opt/novell/mono/lib/pkgconfig/:
```

```
$PKG_CONFIG_PATH
```

After modifying the profile.d you need to logoff and logon again to see the changes. Once you logon again verify that mono is in the path and the correct version is installed by doing

```
which mono
```

```
mono --version
```

Which mono will tell you where the system has found the mono program. Mono --version will run the mono runtime n report what version you have installed. This verifies that mono is installed correctly and that it can be found in the path

Now that Mono s installed on your Linux machine you can build your .NET application using your familiar Windows development environment. You can compile test and run your application on windows to verify that all is working correctly. Then simply copy the binaries over to your Linux machine and execute the program with the mono runtime

```
mono myapp.exe
```

That's it! You now have a working mono environment to execute your .NET applications on Mono.

# Some Gotchas

Sunday, June 05, 2011  
10:32 AM

Some gotchas when running on Linux.

1. Most common mistake is file pathing. Remember that Windows uses backslashes for paths whereas Linux uses forward slashes.  
Tips:
  1. Do NOT hardcode paths as strings.
  2. Always use the Path class in the .NET. `Path.Combine` will use the `DirectorySeparatorCharacter` which will do the right thing on both OSes
  3. Make sure to watch out for config values. If you store paths in the App.config store them forward slashes NOT backslashes. Windows will understand forward slashes in paths and do the right thing whereas linux will not understand backslashes.
  4. If you are moving data from a client machine to a server machine and they are mixed OSes make sure to account for the differences in `DirectorySeparatorCharacter`s across OS'es
2. Case sensitivity
  1. Remember that Linux is a case sensitive OS. This becomes especially important when dealing with files and folders. Loading files and folders or searching will be case sensitive under mono and not under Windows .NET
  2. There is an environment variable that can be set to force mono to NOT be case sensitive. Set `MONO_IOMAP` to all if you want to turn off case sensitivity
3. If doing any sort of Interop with native code you need to keep a few things in mind
  1. The native code must be compiled in a cross platform way. C++ / CLI dll's are not handled by mono.
  2. Recompile your c++ with a cross platform compiler (gnu etc..) to be sure the native library will run on linux
  3. DLL Import does work under mono but you have to remember when Mono calls out to Linux to load a library it doesn't look for .dll files it looks for .so (shared object) files. So if your dll import specifies myclass.dll it will NOT work on linux.
  4. You can use `DLLImport("myclass")` and Windows will append the .dll for you and linux will append a .so which will work well.
  5. Note how the linux `LoadLibrary` call searches for dll's
    - a. A colon-separated list of directories in the user's `LD_LIBRARY_PATH` environment variable. This is a frequently-used way to allow native shared libraries to be found by a CLI program.
    - b. The list of libraries cached in `/etc/ld.so.cache`. `/etc/ld.so.cache` is created by editing `/etc/ld.so.conf` and running `ldconfig(8)`. Editing `/etc/ld.so.conf` is the preferred way to search additional directories, as opposed to using `LD_LIBRARY_PATH`, as this is more secure (it's more difficult to get a trojan library into `/etc/ld.so.cache` than it is to insert it into `LD_LIBRARY_PATH`).
    - c. `/lib`, followed by `/usr/lib`.
  6. Learn to love the `LD_LIBRARY_PATH` and set it to where your .so live (Often this means setting `LD_LIBRARY_PATH` to include "." The current working directory.
  7. Lots more info here : [http://mono-project.com/Interop\\_with\\_Native\\_Libraries](http://mono-project.com/Interop_with_Native_Libraries)

Runtime "debugging"

1. Run your app with `--verbose` option which will spew verbose debugging statements from the mono runtime. This helps determine where mono is looking to load your assemblies and can help find pathing errors.

2. Use log4net and copious amounts of logging to the console output.
3. Some people have been able to attach a remote debugger to the mono runtime from their windows machine and actually step through code. That would be totally awesome but I have yet to get that to work.
4. Some links:
  1. <http://tirania.org/blog/archive/2008/Sep-04.html>
  2. <http://monoloc.blogspot.com/2008/06/remotely-debug-running-process-with-net.html>