# Compiling MonoTorrent

### Pre-requisites

\* A version of the .NET framework, .NET Core or Mono which can compile .NET Standard 2.0 projects.

\* Any IDE which can compile C# code.

### MSBuild:

To build using MSBuild, execute the following command:

$ msbuild /restore

### Makefiles:

To build using the Makefile, execute the following command:

$ make

### Using an IDE:

You can open src\MonoTorrent.sln in any IDE.

## How to use the sample applications

### MonoTorrent Client

To run the sample client you need to do the following:

1) Make sure that MonoTorrent.dll and SampleClient.exe are in the same folder

2) Create a folder in that directory called "Torrents".

3) Put any number of .torrent files into the Torrents directory. This files will all be loaded by the sample client.

5) Launch SampleClient.exe to begin downloading. All files will be downloaded to a directory called "Downloads".

Note: Only statistics will only be shown for the first .torrent loaded into the engine. So there is no real point

in loading more than one .torrent into the engine.

Developer Notes:

There are a few important things developers should note before creating a gui/service using the library. Firstly

there is no guarantee what thread the events will be fired on, so if you're doing GUI updates, you will need to

make sure that you perform your actual GUI update in a threadsafe manner.

### The Tracker

============

The code of the Tracker is located in MonoTorrent.Tracker. There is one sample Tracker implementation

in MonoTorrent.TrackerApp.

The Tracker is a piece of Software which listens for HttpRequests. Each Request can either be an Announce

or Scrape request. Therefore the Tracker needs code which handles HttpRequests. The Tracker was programmed

in such a way that it is independent of the http handling code. The http handling code is called Frontend.

There are currently two Frontends implemented. The first uses the class HttpListener. This implementation

got most attention. The second one uses the Asp.Net infrastructe and the HttpHandles classes. It therefore

could be used in xsp2, mod\_mono or even in IIS. But the second implementation was just a proof of concept

and is not tested but should be functional. There exists and Frontend directory which contains all the

Frontend handling code.

There is also a Backend part of the Tracker. The backend is responsible for storing the Informations per

Torrent which should be Announced to the peers. There is currently one Backend implementation which uses

the .Net internal Datastructes List<> and Dictionary<> called SimpleTorrentManager. A Backend needs to

implement the ITorrentManager interface.

If you would like to start the Tracker the TrackerEngine is the class you would like to use. The sequence

below is enough to start the Tracker:

TrackerEngine engine = TrackerEngine.Instance

engine.Address = "127.0.0.1";

engine.Port = 10000;

engine.Frontend = TrackerFrontend.InternalHttp;

engine.Start();

Adding Torrents is easy too. Just get an Tracker instance and call AddTorrent:

Torrent t = new Torrent();

t.LoadTorrent(path);

TrackerEngine.Instance.Tracker.AddTorrent(t);

The two code snippets are enough to start a simple tracker. If you would like to tune the Tracker to use

less Bandwidth you can set various things in the Tracker instance. One such thing would be to use the

compact response format:

Tracker.Instance.AllowNonCompact = true;

The other place where you can tune is to implement the IIntervalAlgorithm. An implementor can controll at

which rate the peers should request an Announce or Scrape. Currently we have an static implementation

which uses static defaults taken from the original BitTorrent implementation. It's even possible to higher

the intervals based on the number of peers using the Torrent.

Tracker.cs is the code where everything is glued together. It is something like the heart of the Tracker

implementation.

Howto Test:

If you want to test the Tracker just compile it with MonoDevelop. The Tracker searches (and creates if not

found) for a directory named ./torrents. In this Directory every Torrent is loaded on startup. To test

the Torrent you need some torrents pointing at the Tracker. This can be done with the unit test. Just

run gnunit2 and load the Common.dll in src/bin/Debug. Copy the single.torrent and torrentcreator.torrent

into src/bin/Debug/torrents and start the Tracker in MonoDevelop. Then you can load the Torrents in Azureus

or BitTorrent and check if the Tracker reacts on Announces and Scrapes.