Net Ninny web proxy

Summary

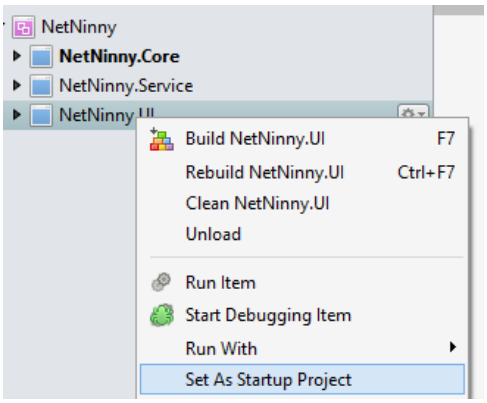
User manual: Net Ninny proxy server

1. Compilation

To compile the proxy, you will need either Visual Studio 2013 (with C# support, express edition works, Windows only) or MonoDevelop/Xamarin Studio (http://monodevelop.com/).

Using one of these IDEs, first open the solution file (NetNinny.sln). The solution contains three (3) projects.

- NetNinny.Core : contains the proxy's logic. It is compiled as a dynamic library.
- NetNinny.Service : contains the code for a console version of the proxy server.
- NetNinny.UI : contains the code for a graphical version of the proxy server.



You can generate either NetNinny.Service or NetNinny.UI. To compile and run one of these, right click on the project node you want to generate, and select "Set As Startup Project". Then run the project by pressing F5 or build it using F7 (Ctrl+Shift+B on visual studio).

2. Configuration and usage

2.1. NetNinny.Service

The NetNinny service needs to be run with some command line arguments to indicate how it should be configured. The arguments are:

```
[port=portnumber] [filters= path] [-v]
```

`--port=portnumber` : let you specify to the port on which the proxy should listen for incoming connections.

`--filters=path` : let you specify the path to the file from which the filtered strings should be read. The file should contain one bad expression per line.

`-v` : enables verbose mode.

`--force_utf8` : overwrite browser's Accept-Encoding field so it accepts only utf-8 encoding, thus allowing filtering more text files (which would be gzipped otherwise).

By default, the port is 5001, and the filters are constructed using hard-coded values (those given in the assignment).

2.2. NetNinny.UI

Once the NetNinny.UI program is run, you may change the "port" and "filters" values, then just press "Start". Some logs will appear in the tabs while the server is running.

3. Features

The proxy has support for the following features:

1. Handles both HTTP/1.0 and HTTP/1.1.
2. Handles HTTP GET requests. (*code ref: ProxyServer.cs:HandleConnexion()*)
3. Blocks and redirect requests with undesirable URLs using several filters, which can be configured. (*code ref: ProxyServer.cs:HandleConnexion()*)
4. Blocks and redirects request with inappropriate content (text only). The proxy only filters text/html or text/plain content. Thus, it will not try to filter script files, or other text files which are not content. It is only able to filter human-readable text, therefore it will not try to filter gzipped files.
5. There is no size limit for the transferred HTTP data. The non-text files are not buffered, they are forwarded per small blocks instead. (*code ref: ProxyServer.cs:ForwardData()*).
6. Compatible with all major browsers. (*code ref: ProxyServer.cs:HandleConnexion()*)
7. Proxy port configurable. (*code ref: ProxyServer.cs:RunServer()*)
8. Smart filtering: filters only uncompressed text files.
9. Handles HTTP POST method for file uploads. (*code ref: ProxyServer.cs:HandleConnexion()*).

"Bonus features":

a. Allow the user to select the words he wants to filter: the proxy reads a text file (whose filename can be given by the user, as mentioned previously in this document) containing one expression to filter on each line.
b. The proxy analyses content it receives to be able to determine the end of content. It handles both chunked transfer-encoding and default transfer-encoding. This method prevents the proxy from loading incomplete files (the call to receive might return 0 when the connection is too slow, ending the connection in some implementations (like *in SocketUtils.cs:ReceiveBytes() / ForwardBytes()*). See : *SocketUtils.cs:ReceiveData()* and *SocketUtils.cs:ForwardData()*.