A simplistic HTTP Server on Windows Phone

This article explains how to implement a simplistic HTTP server that is able to run on Windows Phone 8 or later. This can be used to export files or to provide a platform independent UI on a user's PC.

Introduction

This sample consists mainly of a `HttpServer` class that includes all functionality needed to serve HTTP-requests from a browser. To demonstrate how to work with the actual Request-Strings it will allow you to list all contacts stored on the phone and to search through them. The sample only implements a small subset of the HTTP standard as defined in RFC 2616 but is sufficient to complete requests from a real world WebBrowser.

Warning: This sample extracts information from the device's phone book. Unfortunately the phone book isn't populated with test data on the emulator. The returned pages will therefore be empty in emulator testing. It is therefore suggested to test the sample on an actual device.

Basis of the HttpServer functionality - TCP sockets

The HTTP protocol is based on the TCP protocol. Starting with Windows Phone 8 you can create TCP Host Sockets using the Windows.Networking.Sockets.StreamSocketListener class. We're using an instance of this class in our `HttpServer` to listen for incoming connections. Aside from that we will also store wether the server is currently active (listening for incoming connections) and the port our listening socket will be opened on. Putting those pieces together we get the following two methods for starting and stopping the server.

```
public async void Start()
{
    if (!mIsActive)
    {
    mIsActive = true;
    mListener = new StreamSocketListener();
    mListener.Control.QualityOfService = SocketQualityOfService.Normal;
    mListener.ConnectionReceived += mListener_ConnectionReceived;
    await mListener.BindServiceNameAsync(mPort.ToString());
    }

}

public void Stop()
{
    if (mIsActive)
    {
    mListener.Dispose();
    mIsActive = false;
    }
}
```

As you can see in the `Start()` method incoming connections will be handed over to the `mListener_ConnectionReceived` method to be handled. In this sample we will simply use this method to hand the actual handling over to another method that is being run on a ThreadPool-Thread. That way the event can return and the listener can listen to additional requests.

```
async void mListener_ConnectionReceived(StreamSocketListener sender,
StreamSocketListenerConnectionReceivedEventArgs args)
{
    await Task.Run(() =>
    {
```


(C) Copyright Nokia 2014. All rights reserved.

```
        HandleRequest(args.Socket);
    });
}
```

Handling the request

The `HandleRequest(StreamSocket socket)` method will handle the actual Request received from the client browser. Using a `Windows.Storage.Streams.DataReader` on the input stream provided by the socket we read the actual request from the first line. Those consist of three parts separated by spaces: the HTTP verb, the path of the requested resource and the HTTP version string. We will only support the GET-verb and will ignore the HTTP version. The following lines contain the HTTP headers which will also be ignored. The request will end with two Newlines so we keep reading from the socket until we read an empty line. We will then write a response to the socket's output stream using a `Windows.Storage.Streams.DataWriter`. When finished we will need to tell the writer to actually transmit the data from it's buffers and then dispose of the socket to close the connection.

```
private async void HandleRequest(StreamSocket socket)
{
    //Initialize IO classes
    DataReader reader = new DataReader(socket.InputStream);
    DataWriter writer = new DataWriter(socket.OutputStream);
    writer.UnicodeEncoding = Windows.Storage.Streams.UnicodeEncoding.Utf8;

    //handle actual HTTP request
    String request = await StreamReadLine(reader);
    string[] tokens = request.Split(' ');
    if (tokens.Length != 3)
    {
        throw new Exception("invalid http request line");
    }
    string httpMethod = tokens[0].ToUpper();
    string httpUrl = tokens[1];

    //read HTTP headers - contents ignored in this sample
    while (!String.IsNullOrEmpty(await StreamReadLine(reader))) ;

    // ... writing of the HTTP response happens here

    await writer.StoreAsync();//actually write data to the network interface

    socket.Dispose();
}
```

To read lines from the reader we utilize our own `StreamReadLine(DataReader reader)` method. It loads byte-wise from the provided `DataReader` and returns the read characters when it encounters a newline.

```
private static async Task<string> StreamReadLine(DataReader reader)
{
    int next_char;
    string data = "";
    while (true)
    {
        await reader.LoadAsync(1);
        next_char = reader.ReadByte();
        if (next_char == '\n') { break; }
        if (next_char == '\r') { continue; }
```


(C) Copyright Nokia 2014. All rights reserved.

```
        data += Convert.ToChar(next_char);
    }
return data;
}
```

Writing the Response

When building the HTTP response we originally write all the data into a `StringBuilder` instead of directly to the `DataWriter`. This allows us to return an error in case we encounter an exception while generating the page. After writing the necessary HTTP headers for a successful response we start with the actual HTML markup. Upon reaching the HTML table destined to hold the actual contact information we analyse the URL requested by the client. In case the base folder was requested (/) we simply return all the contacts stored on the device. If the user entered a text following the device name we will look for contacts with that text in their name. If the user entered a URL pointing to a subfolder (e.g. http://<myip>/pics/) we will return a status code 404 signalling that the requested resource could not be found on the server. This serves mainly to demonstrate how such a response is structured. Lastly we will write the contents of our `StringBuilder` to the `DataWriter`. At this point we will reach again the code described in the handling of the request.

```
        private async void HandleRequest(StreamSocket socket)
        {
            //... code handling the request and extracting necessary information

            StringBuilder ret = new StringBuilder();
            try
            {
                bool notFound = false;

                //HTTP header
                ret.AppendLine("HTTP/1.0 200 OK");
                ret.AppendLine("Content-Type: text/html");
                ret.AppendLine("Connection: close");
                ret.AppendLine("");

                //beginning of HTML element
                ret.AppendLine("<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=utf-8\"><title>..::WP8 Device
Contacts::..</title></head>");
                ret.AppendLine("<body>");
                ret.AppendLine("<h1>Contacts</h1>");
                ret.AppendLine("<table>");

                ContactsSearchEventArgs args = null;

                if (httpUrl == "/")
                {
                    args = await SearchContactsAsync();
                }
                else if (!httpUrl.Substring(1).Contains("/"))
                {
                    args = await SearchContactsAsync(httpUrl.Substring(1));
                }
                else
                {
                    notFound = true;
                }
```


(C) Copyright Nokia 2014. All rights reserved.

```
                if (notFound)
                {
                    writer.WriteString("HTTP/1.0 404 File not found\r\n");
                    writer.WriteString("Connection: close\r\n");
                    writer.WriteString("\r\n");
                }
                else
                {
                    if (args.Results.Count() > 0)
                    {
                        foreach (var res in args.Results)
                        {
                            StringBuilder sb = new StringBuilder();

ret.AppendLine(String.Format("<tr><td><b>Name:</b></td><td><b>{0}</b></td></tr>",
res.DisplayName));
                            foreach (var num in res.PhoneNumbers)
                            {

sb.Append(String.Format("<tr><td>{0}</td><td>{1}</td></tr>", num.Kind.ToString(),
num.PhoneNumber));
                            }
                            if (res.PhoneNumbers.Count() > 0)
                            {
                                ret.AppendLine(sb.ToString());
                            }
                            else
                            {
                                ret.AppendLine("<tr><td colspan=\"2\">no phone numbers
found</td></tr>");
                            }
                        }
                    }
                    else
                    {
                        ret.AppendLine("<tr><td colspan=\"2\">no contacts
found</td></tr");
                    }

                    ret.AppendLine("</table>");
                    ret.AppendLine("</body>");
                    ret.AppendLine("</html>");

                    writer.WriteString(ret.ToString());
                }
            }
            catch (Exception ex)//any exception leads to an Internal server error
            {
                writer.WriteString("HTTP/1.0 500 Internal server error\r\n");
                writer.WriteString("Connection: close\r\n");
                writer.WriteString("\r\n");
                writer.WriteString(ex.Message);
            }

            //... code from handling the closing of the connection
        }
```


(C) Copyright Nokia 2014. All rights reserved.

Additional considerations

Security

The data is being transmitted unencrypted over the network. The sample also doesn't do any authentication. Therefore users should be told not to use the HTTP server on unsecure networks. This is especially important if the data being provided is as sensitive as the contact information we use in this sample.

Telling the user how to reach the WebServer

Under most circumstances users will not have assigned a DNS name to their phone. Therefore it will be necessary to tell them the IP address of the phone so they can connect to it. The article How to get the device IP addresses on Windows Phone describes how the IP addresses the device uses on the local network can be acquired.

Keeping the server running

Under most circumstances users will have their devices configured so the display times out after a certain time. In this case Windows Phone sends apps to sleep which would disconnect the user's Web Browser. One approach to deal with this situation would be to keep the display running. As this is not required by the functionality of the server it is a waste of battery power. The better option would be to configure the App to keep running under the lock screen. More information on how to do this and further considerations when using that functionality is available in the article Run Windows Phone application under lock screen.

Handling app lifecycle events

If the user navigates away from your app it will always be deactivated and potentially be tombstoned. If it was merely deactivated the OS will take care of reopening the port used by your HTTP server. In case of tombstoning though you will have to reinitialize the server yourself. To simplify the behaviour our sample app will always stop the HTTP server when being navigated away from and will start it as soon as it is navigated to.

Summary

You have seen how easy it is to reply to HTTP Requests using dynamically generated HTML markup. You will have noticed that a lot of information provided is simply ignored. Utilizing this additional information a lot of additional functionality could be provided (e.g. file uploads). More in depth analysis of the requested URL could also allow for the usage of HTML forms to allow the user to enter data into their Apps using their PC's Web Browser. I hope this article helps you in providing the basis for such more advanced functionality.

Version Hint

Windows Phone: [[Category:Windows Phone]]
[[Category:Windows Phone 7.5]]
[[Category:Windows Phone 8]]

Nokia Asha: [[Category:Nokia Asha]]
[[Category:Nokia Asha Platform 1.0]]

Series 40: [[Category:Series 40]]
[[Category:Series 40 1st Edition]] [[Category:Series 40 2nd Edition]]
[[Category:Series 40 3rd Edition (initial release)]] [[Category:Series 40 3rd Edition FP1]] [[Category:Series 40 3rd Edition FP2]]
[[Category:Series 40 5th Edition (initial release)]] [[Category:Series 40 5th Edition FP1]]
[[Category:Series 40 6th Edition (initial release)]] [[Category:Series 40 6th Edition FP1]] [[Category:Series 40 Developer Platform 1.0]] [[Category:Series 40 Developer Platform 1.1]] [[Category:Series 40 Developer Platform 2.0]]

Symbian: [[Category:Symbian]]
[[Category:S60 1st Edition]] [[Category:S60 2nd Edition (initial release)]] [[Category:S60 2nd Edition FP1]] [[Category:S60 2nd Edition FP2]] [[Category:S60 2nd Edition FP3]]
[[Category:S60 3rd Edition (initial release)]] [[Category:S60 3rd Edition FP1]] [[Category:S60 3rd Edition FP2]]
[[Category:S60 5th Edition]]
[[Category:Symbian^3]] [[Category:Symbian Anna]] [[Category:Nokia Belle]]

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.