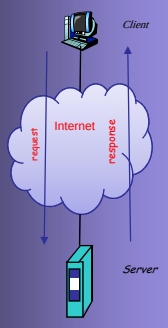


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 8.2: HTTP (Part 2)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

- Part 1:
  - Learn the essentials of the two main versions of HTTP protocol
    - HTTP 1.0 [RFC 1945]: http://www.ietf.org/rfc/rfc1945.txt
    - HTTP 1.1 [RFC 2616]: http://www.ietf.org/rfc/rfc2616.txt
- Part 2:
  - Learn how to write HTTP Clients using WebClient, WebRequest, WebResponse, etc.
  - Learn how to request resources requiring authentication
  - Learn how to pass HTTP request through a proxy server
  - Learn how to write a simple HTTP server

Writing HTTP Clients: WebClient

- Although TcpClient and Socket may be used, C# provides other classes that simplifies writing HTTP clients.
- The simplest of these is the **System.Net.WebClient** class.
  - **Constructor:**

```
public WebClient(); // It has only the default constructor
```

  - **Properties:**

ICredentials Credentials	Gets/sets the network credentials used to authenticate the request
WebHeaderCollection Headers	Gets/sets a collection of headers associated with the request.
WebHeaderCollection ResponseHeaders	Gets/sets a collection of headers associated with the response.
NameValueCollection QueryString	Gets/sets a collection of query name-value pairs for use with GET request

Writing HTTP Clients: WebClient …

Methods:

byte[] DownloadData(string uri)	Downloads data from a resource with the specified URI.
void DownloadFile(string uri, string fileName)	Downloads data from a resource with the specified URI to a local file
Stream OpenRead(string uri)	Opens a readable stream for reading data from a resource with the specified URI.
Stream OpenWrite(string uri, string method)	Opens a stream for posting data to a resource with the specified URI
byte[] UploadData(string uri, string method, byte[] data)	Posts a data to a resource with the specified URI .
byte[] UploadFile(string uri, string method, string fileName)	Posts data in a local file to a resource with the specified URI

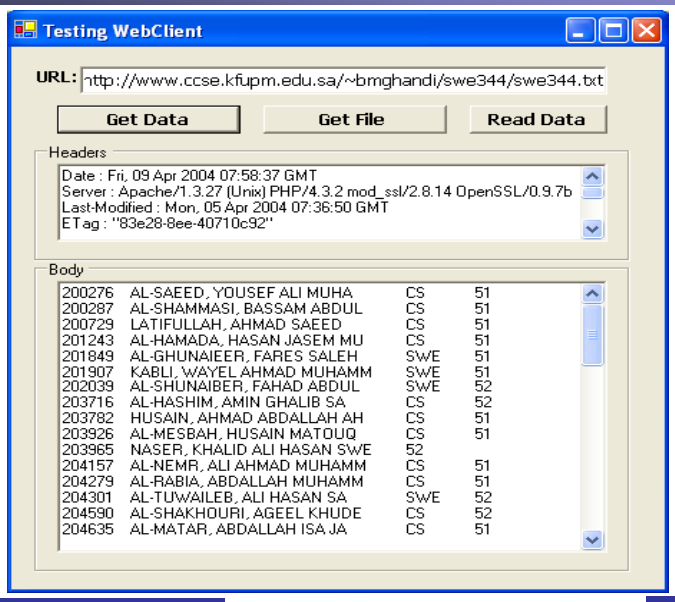
Example 1

```
1.  private WebClient client = new WebClient();
2.  void OnGetData(object sender, System.EventArgs e){
3.    try {
4.      byte[] data = client.DownloadData(urlBox.Text);
5.      resultBox.Text = Encoding.ASCII.GetString(data);
6.      ShowHeaders();
7.    }
8.    catch (WebException ex) {
9.      MessageBox.Show(ex.Message, "Exception");
10.   }
11. }
12. void OnGetFile(object sender, System.EventArgs e){
13.   try {
14.     string fname = urlBox.Text.Substring(
15.                                   urlBox.Text.LastIndexOf("/")+1);
16.     client.DownloadFile(urlBox.Text, fname);
17.     resultBox.Text = "File Downloaded";
18.     ShowHeaders();
19.   }
20.   catch (WebException ex) {
21.     MessageBox.Show(ex.Message, "Exception");
22.   }}
```


Example 1 …

```
24. void OnReadData(object sender, System.EventArgs e) {
25.   try {
26.     StreamReader reader = new StreamReader(
27.                                   client.OpenRead(urlBox.Text));
28.     String line;
29.     resultBox.Text = "";
30.     while ((line=reader.ReadLine())!= null)
31.       resultBox.Text += line + "\r\n";
32.     reader.Close();
33.     ShowHeaders();
34.   }
35.   catch (WebException ex) {
36.     MessageBox.Show(ex.Message, "Exception");
37.   }
38. }
39. void ShowHeaders() {
40.   WebHeaderCollection whc = client.ResponseHeaders;
41.   headerBox.Text = "";
42.   for(int i=0; i<whc.Count; i++) {
43.     headerBox.Text += whc.GetKey(i) + " : " + whc.Get(i) + "\r\n";
44.   }
45.  }
```

Example 1 …



Writing HTTP Clients: NetworkCredential

- Access to a web resource may be restricted – the client must be authenticated before gaining access.
- For such resource, an instance of the **NetworkCredential** class is created and assigned to the **Credentials** property of the **WebClient** instance before sending the request.

Constructors:

```
NetworkCredential();
NetworkCredential(string userName, string password);
NetworkCredential(string userName, string password, string domain);
```

Example:

```
WebClient client = new WebClient();
client.Credentials =
        new NetworkCredential("bmghandi","pass", "itc");
//code to download resource follows
```

Writing HTTP Clients: WebRequest, WebResponse

- For more control over a HTTP Client, C# provides the pair of classes, **WebRequest** and **WebResponse**.
- These classes provide more methods and more properties to directly set/get http headers, including a Proxy property, used to make a request through a proxy server.
- The **WebRequest** and **WebResponse** are abstract classes. To create objects, their subclasses are used depending on the type of the URI.
- WebRequest has concrete subclasses, **HttpWebRequest** and **FileWebRequest**.
- WebResponse has concrete subclasses **HttpWebResponse** and **FileWebResponse**.

Writing HTTP Clients: WebRequest, WebResponse …

WebRequest Properties:

long ContentLength	Gets/sets the content length of the request data being sent
ICredentials Credentials	Gets/sets the network credentials used for authenticating the request
WebHeaderCollection Headers	Gets/sets the collection of headers associated with the request
string Method	Gets/sets the method to use in the request
bool PreAuthenticate	Get/sets if to send authentication info. with the initial request. If false, wait for an authentication challenge before sending.
IWebProxy Proxy	Gets/sets the network proxy to use to access this Internet resource
int TimeOut	Gets/sets the length of time before the request times out

Writing HTTP Clients: WebRequest, WebResponse …

WebRequest Methods:

static WebRequest Create(string uri)	Initializes a new WebRequest instance for the specified URI
Stream GetRequestStream()	Returns a Stream for writing data to the Internet resource
WebResponse GetResponse()	Returns a response to an Internet request

Notes:

- There are also methods, BeginGetRequestStream, BeginGetResponse, EndGetRequestStream, EndGetResponse ,for making asynchronous requests.
- An asynchronous request can be aborted using the Abort method.

Writing HTTP Clients: WebRequest, WebResponse …

WebResponse Properties and Methods

long ContentLength	Gets the content length of data being received
string ContentType	Gets the content type of the data being received
WebHeaderCollection Headers	Gets a collection of header name-value pairs associated with this request
Uri ResponseUri	gets the URI of the Internet resource that actually responded to the request
Stream GetResponseStream()	returns the data stream from the Internet resource
void Close()	closes the response stream

Writing HTTP Clients: WebProxy, GlobalProxySelection

- To communicate through a proxy, an instance of the WebProxy class must be created and assigned to the **Proxy** property of the WebRequest instance before making the request.
  - Alternatively, the WebProxy instance may be assigned globally (for all requests) using the GlobalProxySelection class.
- The most complete version of the WebProxy constructor has the following form:

```
public WebProxy(string Address, bool BypassOnLocal,
                string[] BypassList, ICredentials Credentials)
```

- The **GlobalProxySelection** class is used to define a default proxy which all instances of WebRequest will use.
  - It has a property, **Select**, which is assigned the WebProxy instance.
- Note: Both WebProxy and GlobalProxySelection classes can also be used with **WebClient**.

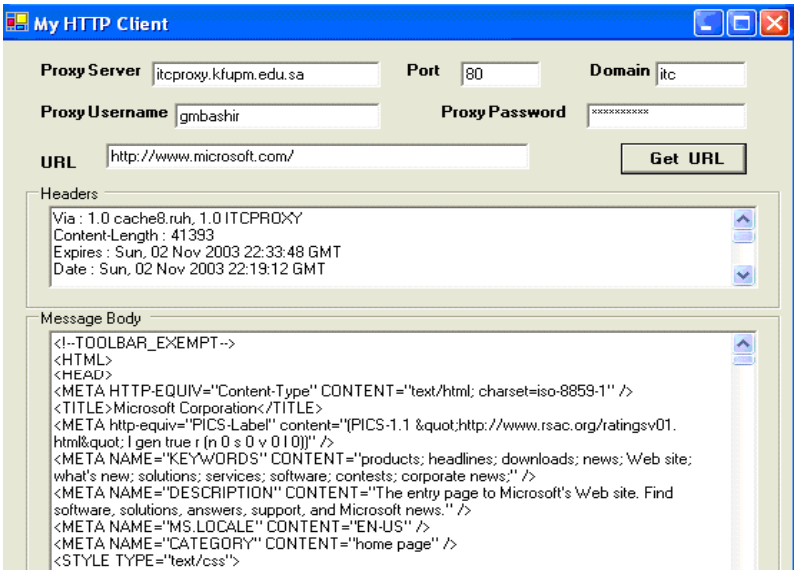
Example 2

```
void OnGetUrlClicked(object sender, System.EventArgs e) {
  headerBox.Text = "";
  resultBox.Text = "";
  WebProxy proxyObject = null;
  if ( proxyPasswordBox.Text != "") {
    string [] localSites ={"www.kfupm.edu.sa",
                           "www.ccse.kfupm.edu.sa"};
    NetworkCredential myCred = new NetworkCredential(
                 proxyUsernameBox.Text, proxyPasswordBox.Text,
                 proxyDomainBox.Text);
    proxyObject=new WebProxy( proxyServerBox.Text+":"+
                    proxyPortBox.Text, true,localSites, myCred);
    //GlobalProxySelection.Select = proxyObject;
  }
  if (!hasProtocol(urlBox.Text))
    urlBox.Text = "http://"+ urlBox.Text;
  //if no file name is given append slash
  if (urlBox.Text.LastIndexOf('/') ==
                    urlBox.Text.LastIndexOf("//")+1)
    urlBox.Text +="/";
```

Example 2 …

```
21.    try {
22.      WebRequest req = WebRequest.Create(urlBox.Text);
23.      if (proxyObject != null)
24.        req.Proxy = proxyObject;
25.          WebResponse result = req.GetResponse();
26.          Stream rs = result.GetResponseStream();
27.          StreamReader sr = new StreamReader(rs);
28.          resultBox.Text=sr.ReadToEnd();
29.
30.      WebHeaderCollection whc = result.Headers;
31.      for (int i=0; i<whc.Count; i++) {
32.        headerBox.Text += whc.GetKey(i) + " : " + whc.Get(i) + "";
33.      }
34.      result.Close();
35.    }
36.    catch(Exception) {
37.      resultBox.Text = "Exception has occured";
38.    }
39.  }
```


Example 2 …



Writing a Http Server

- HTTP server is created using
  - TcpListener class or the Socket class
  - HttpListener class (introduced in .NET Framework 2.0)
    - http://msdn.microsoft.com/en-us/library/system.net.httplistener.aspx
- The server is created as a Multi-Client sever taking into account the HTTP protocol (1.0 or 1.1).

Example 3

```
using System;
//...deleted some using statements here
class SimpleHttpServer {
  static int count = 0;

  public static void Main(string[] args) {
    string docFolder = Directory.GetCurrentDirectory();
    int port = 8080;
    TcpListener tcpListener = new TcpListener(IPAddress.Any, port);
    tcpListener.Start();

    Console.WriteLine("Waiting for a connection on port 8080 ... ");

    while (true) {
      TcpClient client = tcpListener.AcceptTcpClient();
      Console.WriteLine("Visitor Count: "+ (++count));
      ConnectionHandler handler = new ConnectionHandler(client);
      ThreadPool.QueueUserWorkItem(new
              WaitCallback(handler.HandleConnection), docFolder);
    }
  }
}
```

Example 3 …

```
23. class ConnectionHandler {
24.   TcpClient client;
25.   NetworkStream stream;
26.   StreamReader reader;
27.   StreamWriter writer;
28.   string docFolder;
29.   string request;
30.   public ConnectionHandler(TcpClient client) {
31.     this.client = client;
32.     stream = client.GetStream();
33.     reader = new StreamReader(stream);
34.     writer = new StreamWriter(stream);
35.   }
36.   public void HandleConnection(Object state) {
37.     docFolder = (string) state;
38.     request = reader.ReadLine(); //read the request;
39.     string input;
40.     //read rest of the request
```


Example 3 …

```
41.     while ((input = reader.ReadLine()).Length != 0);
42.       string[] token = request.Split(' ');
43.       if (token.Length == 3 && token[0] == "GET" &&
44.          token[1].StartsWith("/") && token[2].StartsWith("HTTP")){
45.           string fileName = token[1].Replace('/', '\\');
46.           fileName = docFolder + fileName;
47.           if (File.Exists(fileName)) {
48.               writer.Write("HTTP/1.0 200 OK\r\n");
49.               SendHeaders(fileName);
50.               SendFile(fileName);
51.           }
52.           else
53.               WriteError(404, "Not Found");
54.       }
55.       else {
56.          WriteError(400, "Bad Request");
57.     } //while
58.     stream.Close();
59.     client.Close();
60.   } //method: HandleConnection
```

Example 3 …

```
61.     public void WriteError(int status, string message) {
62.       writer.Write("HTTP/1.0 " + status + " " + message + "\r\n");
63.       writer.Write("\r\n");
64.       writer.Flush();
65.     }
66.     public void SendHeaders(string fileName) {
67.       FileInfo file = new FileInfo(fileName);
68.       writer.Write("Content-Length: " + file.Length + "\r\n");
69.       writer.Write("\r\n"); //blank line here!
70.       writer.Flush();        //need to flush the headers
71.     }
72.     public void SendFile(string fileName) {
73.       FileStream file = new FileStream(fileName, FileMode.Open,
74.                                  FileAccess.Read, FileShare.Read);
75.       byte[] data = new byte[4096];
76.       while (file.Position < file.Length) {
77.         int read = file.Read(data, 0, data.Length);
78.         stream.Write(data, 0, read);
79.       }
80.       stream.Flush();
81.       file.Close();
82.     }}
```


HTTPListener

- Provides a simple, programmatically controlled HTTP protocol listener
- To use HttpListener, create a new instance of the class using the HttpListener constructor and use the Prefixes property to gain access to the collection that holds the strings that specify which URI prefixes the HttpListener should process
- A URI prefix string is composed of
  - a scheme (http or https),
  - a host,
  - an optional port,
  - and an optional path.
- Example of a complete prefix string is
  - http://www.contoso.com:8080/customerData/
- Note: Prefixes must end in a forward slash ("/").
- The HttpListener object with the prefix that most closely matches a requested URI responds to the request.
- Multiple HttpListener objects cannot add the same prefix; an exception is thrown if a HttpListener adds a prefix that is already in use

HTTPListener …

- When a port is specified, the host element can be replaced with “*” or “+”
  - Example: “http://*:8080/” and “https://+:8080”
  - “*” indicates that the HttpListener accepts requests sent to the port if the requested URI does not match any other prefix
  - “+” specifies that the HttpListener accepts all requests sent to a port
  - The “*” and “+” characters can be present in prefixes that include paths.
- To begin listening for requests from clients, add the URI prefixes to the collection and call the Start method.
- HttpListener offers both synchronous and asynchronous models for processing client requests.
  - Requests and their associated responses are accessed using the **HttpListenerContext** object returned by the **GetContext** method or its asynchronous counterparts (**BeginGetContext** and **EndGetContext** methods)
  - Synchronous model: is appropriate if your application should block while waiting for a client request and if you want to process only one request at a time.
    - Using the synchronous model, call the GetContext method, which waits for a client to send a request. The method returns an HttpListenerContext object to you for processing when one occurs.
  - asynchronous model: the application does not block while waiting for requests and each request is processed in its own execution thread.
    - Use the BeginGetContext method to specify an application-defined method to be called for each incoming request. Within that method, call the EndGetContext method to obtain the request, process it, and respond.

HTTPListener…

- In either model, incoming requests are accessed using
  - the HttpListenerContext..::.Request property and are represented by **HttpListenerRequest** objects.
  - Similarly, responses are accessed using the HttpListenerContext..::.Response property and are represented by **HttpListenerResponse** objects.
  - These objects share some functionality with the **HttpWebRequest** and **HttpWebResponse** objects, but the latter objects cannot be used in conjunction with **HttpListener** because they implement client, not server, behaviors.
- An HttpListener can require client authentication.
  - You can either specify a particular scheme to use for authentication, or you can specify a delegate that determines the scheme to use.
  - You must require some form of authentication to obtain information about the client's identity.
  - For additional information, see the User, AuthenticationSchemes, and AuthenticationSchemeSelectorDelegate properties.

Example

```
// This example requires the System and System.Net namespaces.
public static void SimpleListenerExample(string[] prefixes) {
if (!HttpListener.IsSupported) {
    Console.WriteLine ("Windows XP SP2 or Server 2003 is required to use
                the HttpListener class.");
    return;
} // URI prefixes are required, for example "http://contoso.com:8080/index/".
if (prefixes == null || prefixes.Length == 0)
     throw new ArgumentException("prefixes"); // Create a listener.
HttpListener listener = new HttpListener();
// Add the prefixes.
foreach (string s in prefixes) { listener.Prefixes.Add(s); }
listener.Start(); Console.WriteLine("Listening...");
// Note: The GetContext method blocks while waiting for a request.
HttpListenerContext context = listener.GetContext();
HttpListenerRequest request = context.Request;
// Obtain a response object.
HttpListenerResponse response = context.Response;
// Construct a response.
string responseString = "<HTML><BODY> Hello world!</BODY></HTML>";
byte[] buffer = System.Text.Encoding.UTF8.GetBytes(responseString);
// Get a response stream and write the response to it.
response.ContentLength64 = buffer.Length;
System.IO.Stream output = response.OutputStream;
output.Write(buffer,0,buffer.Length);
// You must close the output stream.
output.Close();
listener.Stop(); }
```


Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- HTTP 1.0 http://www.ietf.org/rfc/rfc1945.txt
- HTTP 1.1 http://www.ietf.org/rfc/rfc2616.txt
- Books
  - Richard Blum, C# Network Programming. Sybex 2002.
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm