Networking

Introduction

- The C# programming language offers variety of networking related classes in the `System.Net.*` name space.

  These classes support various standard network protocols, such as HTTP, TCP/IP and FTP.

  `WebClient`

  This class implements the Facade design pattern while providing support for simple download/upload operations via HTTP and FTP.

  `WebRequest & WebResponse`

  These classes represent requests and responses involved with client side HTTP and FTP operations.

Introduction

`HttpListener`

This class can be used for building an HTTP server.

`SmtpClient`

These class can be used for developing a mailing client via SMTP.

`Dns`

This class assists with converting between domain names and their IP addresses.

`TcpClient, UdpClient, TcpListener & Socket`

These classes can be used for developing TCP\IP and UDP client server applications.

Computer Addresses

- Each computer in a network has an IP number. The .NET framework supports both the IPV4 and the IPV6 addressing systems.
- The `System.Net.IPAddress` class represents a computer address.

```
...
IPAddress address = new IPAddress(new byte[]{127,0,0,01});
IPAddress address = new IPAddress("127.0.0.1");
...
```

Computer Addresses

- Both UDP and TCP protocols support each IP address with 65,535 ports, allowing each computer with a single IP address to run multiple applications, each one of them on a its own separated port.

  The ports in the range of 1..1024 are usually kept for standard applications, such as HTTP (port 80), SMTP (port 25) etc.

Computer Addresses

- Each combination of an IP number and a port address is represented in .NET by an `IPEndPoint` object.

```
...
IPAddress address = new IPAddress(new byte[]{127,0,0,01});
IPEndPoint point = new IPEndPoint(address,8080);
...
```

URIs

- URI is a formatted string that describes a resource on the Internet on on a Local Area Network (Intranet).

  The resource can be a file, an email address, a web page or even a small executable code that generates content dynamically sent back to the browser.

URIs

- We can easily construct a URI object by passing over a string of any of the following optional formats:

URI String

It can be either a web URI address such as http://www.jacado.com or a local file URI address such as file://michh/docs/memo.doc.

Local File Absolute Path

It can be any full path to a file on our computer such as c:\docs\memo.txt.

UNC Path

It can be any UNC path to a file on our local area network, such as \\michh\share\docs\memo.txt.

URIs

- The Uri's '`IsLoopback`' property indicates whether it references the local host.

URIs

- The Uri's '`IsFile`' property indicates whether the Uri references a local file.

  If '`IsFile`' is true, referring '`LocalPath`' property returns an absolute path we can access by calling the `File.Open` method.

URIs

- The Uri's '`IsUnc`' property indicates whether the Uri references a UNC path.

URIs

- The `Uri`'s '`EscapeUriString()`' method converts a string into a valid URL by converting all characters with an ASSCII value bigger than 127 into their hexadecimal representation.

WebRequest & WebResponse

- The `WebRequest` and the `WebResponse` encapsulate a request and a response.

The `WebClient` Class

- The `WebClient` class is a facade class that does all work involved with using the `WebRequest` and `WebResponse` classes.
- The `WebClient`'s `BaseAddress` property allows us to specify a string to be prefixed to all addresses.

The `WebClient` Download Methods

- The `WebClient` class includes the following Download methods:

```
public void DownloadFile(string address, string fileName);
public string DownloadString(string address);
public byte[] DownloadData(string address);
public Stream OpenRead(string address);
```

The `WebClient` Upload Methods

- The `WebClient` class includes the following Upload methods:

```
public byte[] UploadFile(string address, string fileName);
public byte[] UploadFile(string address, string method,
                         string fileName);
public string UploadString(string address, string data);
public string UploadString(string address, string method,
                           string data);
public byte[] UploadData(string address, byte[] data);
public byte[] UploadData(string address, string method,
                         byte[] data);
public byte[] UploadValues(string address, string method,
                           NameValueCollection data);
```

The `WebClient` Upload Methods

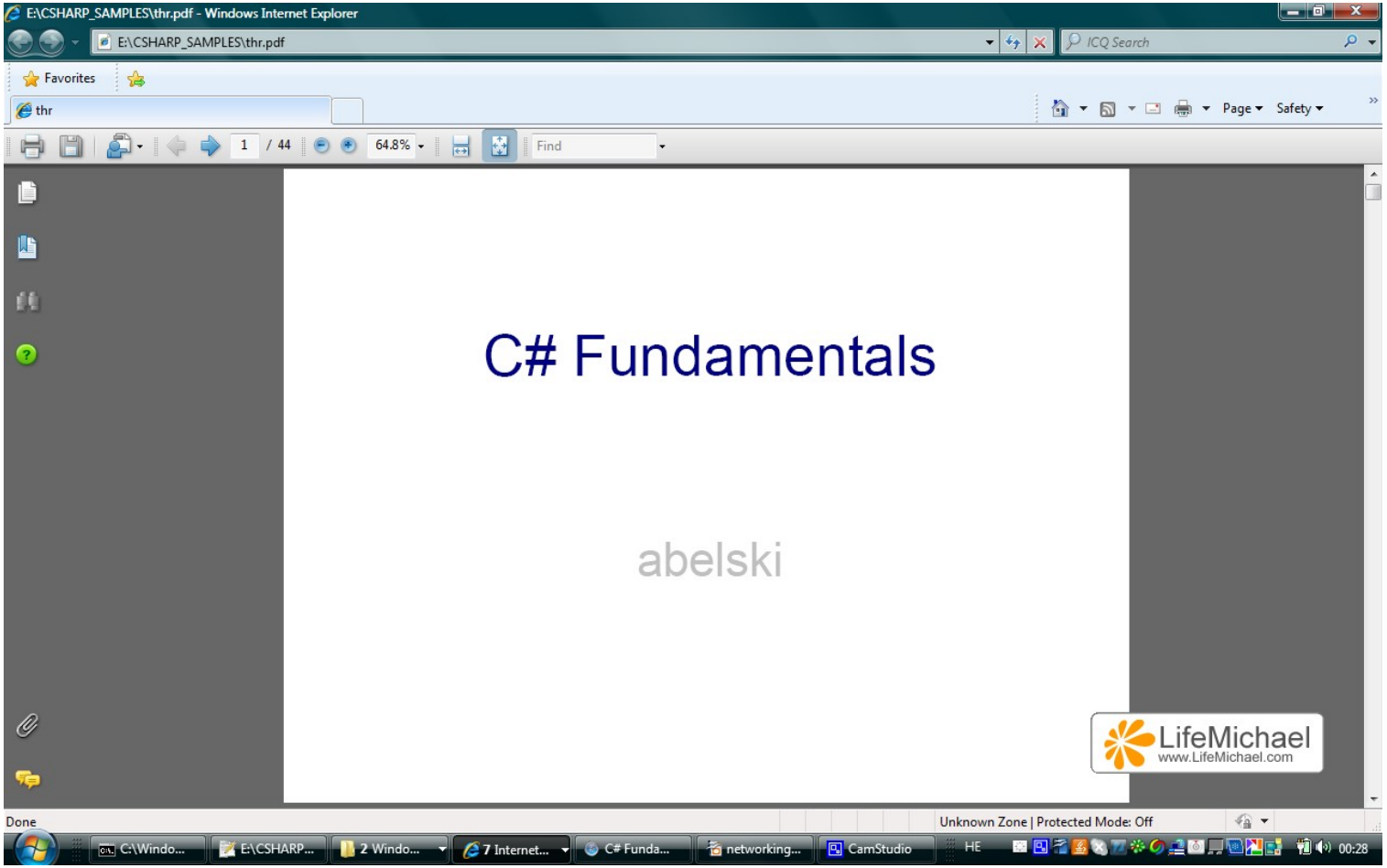
```
public Stream OpenWrite(string address);
public Stream OpenWrite(string address, string method);
```

- Calling the `UploadValues` method we can post values to HTTP form.

The `WebClient` Sample

```
using System;
using System.Net;

namespace abelski.csharp
{
    class WebClientDemo
    {
        static void Main()
        {
            using(WebClient web = new WebClient())
            {
                web.DownloadFile(
                    "http://www.abelski.com/courses/csharp/introduction.pdf",
                    "thr.pdf");
            }
            System.Diagnostics.Process.Start("thr.pdf");
        }
    }
}
```

The `WebClient` Sample



The `WebRequest` Class

- Using the `WebRequest` class the first step is instantiating that class with a URI pointing at the resource we request.

  The next steps might be assigning the `Proxy` property as well as the `Credentials` property (if there is a need).

The `WebRequest` Class

- In order to upload data we should call the `GetRequestStream` on the `WebRequest` object and write to the stream we get.

The `WebRequest` Class

- In order to download data we should first call `GetResponse` on the `WebRequest` object. Calling this method returns a reference for a `WebResponse` object.
- Calling `GetResponseStream` on the `WebResponse` object should provide us with a stream we can read.

The `WebRequest` Sample

```
using System;
using System.Net;
using System.IO;

namespace abelski.csharp
{
    class WebRequestDemo
    {
        static void Main()
        {
            WebRequest request = null;
            HttpWebResponse response = null;
            Stream dataStream = null;
            StreamReader reader = null;
            try
            {
                request = WebRequest.Create ("http://www.yahoo.com/");
                response = (HttpWebResponse)request.GetResponse ();
                Console.WriteLine (response.StatusDescription);
                dataStream = response.GetResponseStream ();
```

The `WebRequest` Sample

```
                reader = new StreamReader (dataStream);
                string responseFromServer = reader.ReadToEnd ();
                Console.WriteLine (responseFromServer);
            }
            catch(Exception e)
            {
                Console.WriteLine(e.ToString());
            }
            finally
            {
                reader.Close ();
                dataStream.Close ();
                response.Close ();
            }
        }

    }
}
```

The `WebRequest` Sample

```
C:\Windows\system32\cmd.exe

E:\CSHARP_SAMPLES>WebRequestDemo
OK
<html>
<head>
<title>Yahoo!</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="robots" content="noindex, nofollow">
<meta name="robots" content="noarchive">
<meta http-equiv="PICS-Label" content='(PICS-1.1 "http://www.icra.org/ratin
.html" l r (cz 1 lz 1 nz 1 oz 1 vz 1) gen true for "http://www.yahoo.com" r
1 lz 1 nz 1 oz 1 vz 1) "http://www.rsac.org/ratingsv01.html" l r (n 0 s 0 v
0) gen true for "http://www.yahoo.com" r (n 0 s 0 v 0 l 0))'>
<base href="http://www.yahoo.com/_ylh=X3oDMTFnZHRzaG12BF9TAzI3MTYxNDkEcGlkA
QwNDIxMjAEdGVzdAMwBHRtcGwDdGFibGUuaHRtbA--/" target="_top">
<style type="text/css">
a{color:#16387c;}
a:link,a:visited{text-decoration:none;}
a:hover{text-decoration:underline;}
</style>
<style type="text/css" media="all">
#p{width:310px;}
form{margin:0;}
</style>
</head>
<body link="#16387c" vlink="#16387c">
```

Proxy Server

- The proxy server is an intermediate through which HTTP and FTP requests are routed.
- Organizations usually hold a proxy server for security reasons. The proxy server can request its clients to authenticate. The proxy server can screen the content its users browse.

Proxy Server

- The proxy server has its own address. The `Proxy` property the `WebClient` and the `WebRequest` have enable us to set them with a `WebProxy` object, that represents a proxy server.

Proxy Server

```
...
WebProxy prox = new WebProxy("122.131.12.23",720);
prox.Credentials = new NetworkCredential("iuser","ipass");
WebClient web = ...
web.Proxy = prox;
...
```

Proxy Server

```
...
WebProxy prox = new WebProxy("122.131.12.23",720);
prox.Credentials = new NetworkCredential("iuser","ipass");
WebRequest request = ...
request.Proxy = prox;
...
```

Proxy Server

- If we don't have a proxy and we don't set the `Proxy` property to '`null`' on all `WebClient` and `WebRequest` objects the framework will attempt to automatically detect the proxy settings. That might cost us up to 30 sec.

Proxy Server

- We can set a global default proxy by referring the `DefaultWebProxy` property in `WebRequest` class.

```
...
WebRequest.DefaultWebProxy = prox;
...
```

Authentication

- Using an object instantiated from `NetworkCredential` we can provide a username and a password when accessing an HTTP or FTP website.

```
...
using (WebClient client = new WebClient())
{
  ...
  client.Credentials = new NetworkCredentials(username,password);
  ...
}
...
```

Authentication

- The `NetworkCredential` class supports dialog based authentication protocols as Basic and Digest.

Parallel Execution

- Communication over the network can be time consuming. In many cases it is more efficient to have multiple `WebClient` or multiple `WebRequest` objects executed concurrently.

Parallel Execution

```
using System;
using System.Net;
using System.IO;
using System.Threading;

namespace abelski.csharp
{
    class ConcurrentNetworkingDemo
    {
        static void Main()
        {
            string[] links = {
                "http://www.abelski.com/courses/csharp/introduction.pdf",
                "http://www.abelski.com/courses/csharp/basic.pdf",
                "http://www.abelski.com/courses/csharp/oop.pdf"
                };
            for(int i=0; i<links.Length; i++)
            {
                new Thread(Download).Start(links[i]);
            }
        }
```

Parallel Execution

```
        static void Download(Object ob)
        {
            using(WebClient client = new WebClient())
            {
                try
                {
                    string[] str = ((string)ob).Split('/');
                    string filename = str[str.Length-1];
                    Console.WriteLine(filename+" started to download");
                    client.Proxy = null;
                    client.DownloadFile((string)ob,filename);
                    Console.WriteLine(filename+" was saved");
                }
                catch(WebException e)
                {
                    Console.WriteLine(e.Status);
                }
            }
        }
    }
}
```

Parallel Execution

```
C:\Windows\system32\cmd.exe

E:\CSHARP_SAMPLES>csc ConcurrentNetworkingDemo.cs
Microsoft (R) Visual C# 2008 Compiler version 3.5.30729.1
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.


E:\CSHARP_SAMPLES>ConcurrentNetworkingDemo
introduction.pdf started to download
basic.pdf started to download
oop.pdf started to download
oop.pdf was saved
basic.pdf was saved
introduction.pdf was saved

E:\CSHARP_SAMPLES>
```

The HTTP Headers

- The `WebClient` and the `WebRequest` objects enable us to add custom HTTP headers for our request as well as to enumerate the HTTP headers assigned to our response.

```
...
using(WebClient client = new WebClient()) {
    ...
    client.Headers.Add("header_name","header_value");
    ...
    foreach(string name in client.ResponseHeaders.Keys) {
        Console.WriteLine(name+"="+client.ResponseHeaders[name]);
    }
}
...
```

The HTTP Headers

```
using System;
using System.Net;
using System.IO;
using System.Threading;

namespace abelski.csharp
{
    class HttpHeadersDemo
    {
        static void Main()
        {
            using(WebClient client = new WebClient())
            {
                    client.Proxy = null;
                    client.DownloadString("http://www.jacado.com");
                    foreach(string name in client.ResponseHeaders.Keys)
                    {
                        Console.WriteLine(name
                            +"="+client.ResponseHeaders[name]);
                    }
            }
        }
    }
}
```

The HTTP Headers

```
E:\CSHARP_SAMPLES>HttpHeadersDemo
Connection=close
Content-Length=69
Content-Type=text/html; charset=UTF-8
Date=Sun, 27 Sep 2009 22:18:11 GMT
ETag=W/"69-1176812108000"
Last-Modified=Tue, 17 Apr 2007 12:15:08 GMT
Server=Apache/2.0.52 (CentOS)

E:\CSHARP_SAMPLES>_
```

The Query String

- The `WebClient` object provides an easy way to add query strings through the `QueryString` dictionary style property.

```
http://www.abelski.com/store.php?id=12&name=moshe
```

Query String

The Query String

```
using System;
using System.Net;
using System.IO;
using System.Threading;

namespace abelski.csharp
{
    class QueryStringDemo
    {
        static void Main()
        {
            using(WebClient client = new WebClient())
            {
                client.Proxy = null;
                client.QueryString.Add("productId","207573");
                client.QueryString.Add("storeId","2218");
                client.QueryString.Add("deviceId","500");
                client.QueryString.Add("platformId","20");
```

The Query String

```
                    client.DownloadFile(
                        "http://www.handango.com/catalog/ProductDetails.jsp",
                        "classicroulette.html");
                    System.Diagnostics.Process.Start("classicroulette.html");
                }
            }
        }
    }
```

The Query String

Jacado Classic Roulette : ...

file:///E:/CSHARP_SAMPLES/classicroulette.html

סימניות אחרות | Course: Twitter Busi... | http://www.nrg.co.il... | Customize Links

Jacado Classic Roulette

- by Zindell Technologies, Ltd.
- [+]Be the first to submit a review!

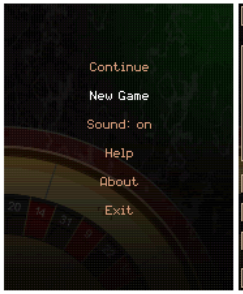
Check Compatibility Please select your device

$5.95

- +59 Reward Points
-
- Tell a Friend »

« Back to Overview

Classic Roulette is a simulation game based on standard casino European Roulette, where an ivory ball rolls in a spinning wheel that consists of 36 numbers and 0. The ball randomly stops on one of 37 numbers. Your goal is to predict the outcome of each spin. You bet against the bank and your winnings are determined by how well you have predicted the ball"s final resting-place. The simulation game provides full betting possibilities, spinning simulation and win/loss situations.



LifeMichael
www.LifeMichael.com

This game was developed by Jacado

HTML Forms

- Working with a `WebClient` object we can post data to HTML forms. We can do that by calling the `UploadValues` method on the `WebClient` object we work with.
- The next sample uses the form at the following URL address: http://www.abelski.com/courses/csharp/sumform.html.

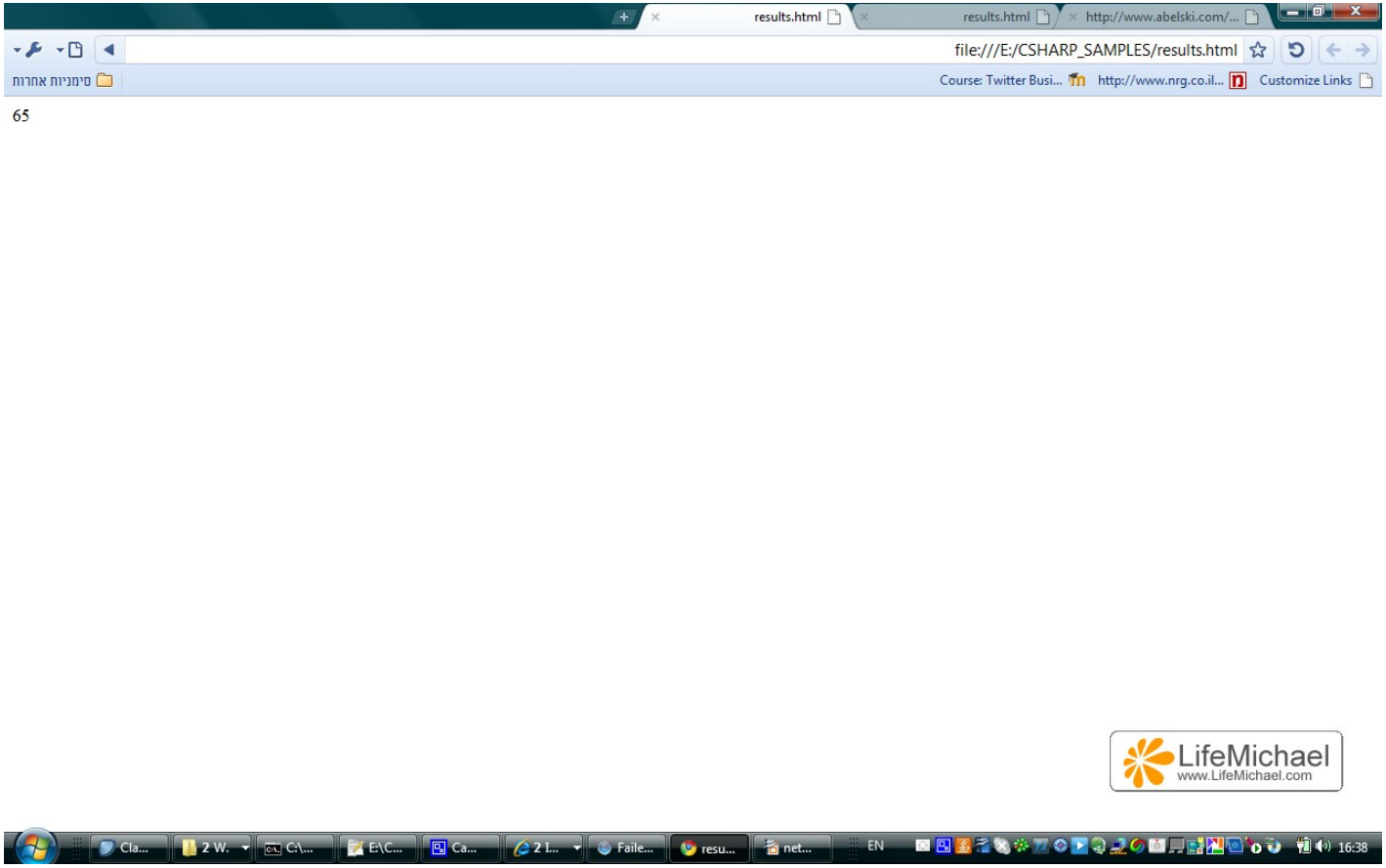
HTML Forms

```
using System;
using System.Net;
using System.IO;
using System.Threading;

namespace abelski.csharp
{
    class WebFormsDemo
    {
        static void Main()
        {
            using(WebClient client = new WebClient())
            {
                    System.Collections.Specialized.NameValueCollection collection =
                        new System.Collections.Specialized.NameValueCollection();
                    collection.Add("numA","43");
                    collection.Add("numB","22");
```

HTML Forms

```
                    client.Proxy = null;
                    byte[] result = client.UploadValues(
                        "http://www.abelski.com/courses/csharp/sumform.php",
                        "POST",collection);
                    System.IO.File.WriteAllBytes("results.html",result);
                    System.Diagnostics.Process.Start("results.html");

            }
        }
    }
}
```

HTML Forms



Cookies

- By default, the `HttpWebRequest` object ignores the received cookies. In order to accept cookies we should create a `CookieContainer` object and assign it to the `WebRequest` we work with.

Cookies

```
using System;
using System.Net;
using System.IO;
using System.Threading;

namespace abelski.csharp
{
    class CookiesDemo
    {
        static void Main()
        {

                CookieContainer container = new CookieContainer();
                HttpWebRequest request = (HttpWebRequest)WebRequest.Create(
                    "http://www.bing.com");
                request.Proxy = null;
                request.CookieContainer = container;
```

Cookies

```
            using(HttpWebResponse response =
                (HttpWebResponse)request.GetResponse())
            {
                foreach(Cookie cookie in response.Cookies)
                {
                    Console.WriteLine("name="+cookie.Name);
                    Console.WriteLine("value="+cookie.Value);
                    Console.WriteLine("path="+cookie.Path);
                    Console.WriteLine("domain="+cookie.Domain);
                    Console.WriteLine();
                }
            }

        }
    }
}
```

Cookies

```
C:\Windows\system32\CMD.exe

E:\CSHARP_SAMPLES>CookiesDemo
name=_FS
value=mkt=-IL&ui=he-IL
path=/
domain=.bing.com

name=MUID
value=5C73C6418C294CDF900AEE8A8C79D3D2
path=/
domain=.bing.com

name=OrigMUID
value=5C73C6418C294CDF900AEE8A8C79D3D2%2ca3a2d6dcbf484ba288ad65bed110af54
path=/
domain=.bing.com

name=OVR
value=flt=0&DomainVertical=0&Cashback=cbtest4&MSCorp=kievfinal&GeoPerf=0
path=/
domain=.bing.com

name=RMS
value=T=262144
path=/
domain=www.bing.com
```

Cookies

- We can assign the received cookies to further requests we send back.
- We can do that by assigning the `CookieContainer` object to each and every new `WebRequest` object we create.

```
...
request.CookieContainer = container;
...
```

Cookies

- We can create a new cookie and add it to the `CookieContainer` we want to send back to the server.

```
...
CookieContainer container = new CookieContainer();
Cookie cookie = new Cookie("id","a2323fssd","/",".abelski.com");
container.Add(cookie);
...
```

- The `CookieContainer` can hold various cookies with different domains and paths. The `WebRequest` object will send back only those that match the path and domain of the URL it refers to.

HTTP Server

- We can develop our own HTTP server by using the `HttpListener` class.

HTTP Server

```
using System;
using System.Net;
using System.IO;
using System.Threading;
using System.Text;

namespace abelski.csharp
{
    class SimpleHttpServer
    {
        static void Main()
        {
            HttpListener listener = null;
            try
            {
                listener = new HttpListener();
                listener.Prefixes.Add(
                    "http://localhost:1300/simpleserver/");
                listener.Start();
```

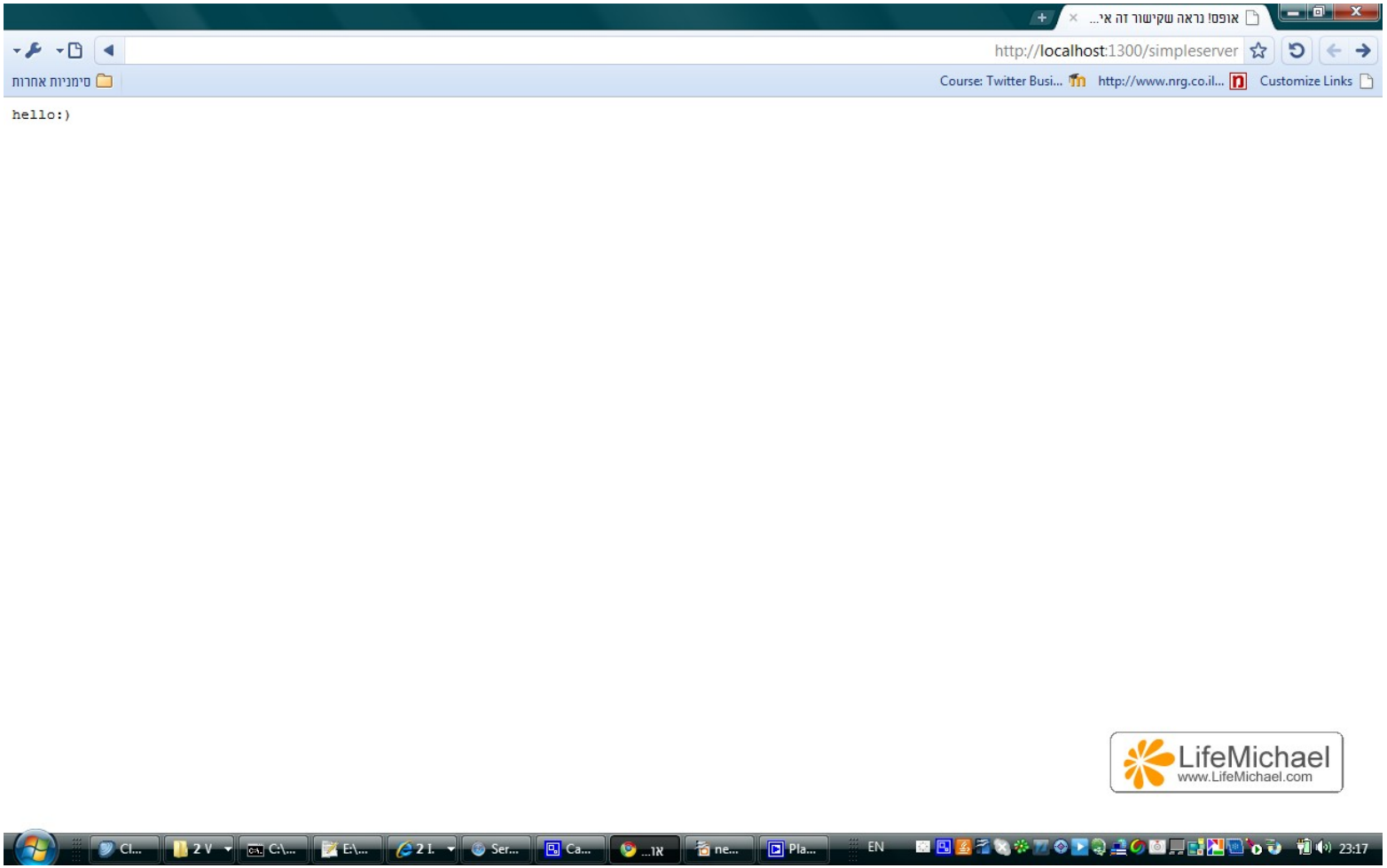
HTTP Server

```
while(true)
{
    Console.WriteLine("waiting...");
    HttpListenerContext context = listener.GetContext();
    string msg = "hello:)";
    context.Response.StatusCode = (int)HttpStatusCode.OK;
    using(Stream stream = context.Response.OutputStream)
    {
        using(StreamWriter writer = new StreamWriter(stream))
        {
            writer.Write(msg);
        }
    }
    Console.WriteLine("msg sent...");
}
```

HTTP Server

```
            }
            catch(WebException e)
            {
                Console.WriteLine(e.Status);
            }
            finally
            {
                listener.Stop();
            }
        }
    }
}
```

HTTP Server



FTP

- We can use the `WebClient` class to develop FTP applications both for uploading and for downloading data.

FTP

```
using System;
using System.Net;
using System.IO;
using System.Threading;
using System.Text;

namespace abelski.csharp
{
    class SimpleFTPServer
    {
        static void Main()
        {
            using(WebClient client = new WebClient())
            {
                client.Proxy = null;
                client.Credentials =
                    new NetworkCredential("_________","_________");
                client.BaseAddress = "ftp://www.abelski.com/";
                client.UploadString("temp.txt",
                    "hello. this is a try");
            }
        }
    }
}
```

FTP

- We can use the methods defined as string constants in `WebRequestMethods.FTP` in order to perform various FTP operations.

`AppendFile`	`PrintWorkingDirectory`
`DeleteFile`	`RemoveDirectory`
`DownloadFile`	`Rename`
`GetDateTimestamp`	`UploadFile`
`GetFileSize`	`UploadFileWithUniqueNmae`
`ListDirectory`	`MakeDirectory`
`ListDirectoryDetails`	

FTP

- In order to use these methods you should assign the method string constant to the web request's `Method` property.

FTP

```
using System;
using System.Net;
using System.IO;
using System.Threading;
using System.Text;

namespace abelski.csharp
{
    class FTPServerMethods
    {
        static void Main()
        {
            FtpWebRequest request = null;
            FtpWebResponse respons = null;
            StreamReader reader = null;
            request = (FtpWebRequest)WebRequest.
                Create("ftp://www.abelski.com/abelski.com");
            request.Proxy = null;
            request.Credentials = new NetworkCredential(
                    "________","______");
            request.Method = WebRequestMethods.Ftp.ListDirectory;
```

FTP

```
            respons = (FtpWebResponse)request.GetResponse();
            reader = new StreamReader(respons.GetResponseStream());
            Console.WriteLine(reader.ReadToEnd());

        }
    }
}
```

FTP

```
C:\Windows\system32\CMD.exe

E:\CSHARP_SAMPLES>FtpServerMethods
abelski.com/error_log.1.gz
abelski.com/moodledata
abelski.com/access_log.1.gz
abelski.com/access_log
abelski.com/error_log.3.gz
abelski.com/access_log.2.gz
abelski.com/access_log.4.gz
abelski.com/awstats
abelski.com/error_log.2.gz
abelski.com/access_log.3.gz
abelski.com/cgi-bin
abelski.com/error_log.4.gz
abelski.com/html
abelski.com/error_log

E:\CSHARP_SAMPLES>
```

The `Dns` Class

- Using the `Dns` class we can interact with the Domain Name Service (DNS) servers that convert between IP numbers (e.g. 20.110.208.104) and human friendly URL addresses (e.g. www.abelski.com).
- Calling the `GetHostAddresses()` static method we can convert from a friendly URL address (domain name) to its IP numbers.

  In some cases, there are more than one IP number assigned for a specific domain name.

The Dns Class

```
using System;
using System.Net;

namespace abelski.csharp
{
    class DNSDemo
    {
        static void Main()
        {
            string[] domains =
                {"www.google.com","www.bing.com","www.abelski.com",
                "www.xperato.com","www.zindell.com","www.jacado.com",
                "www.zindego.com","www.zuntel.com"};
            foreach(string domain in domains)
            {
                Console.WriteLine(domain);
                foreach(IPAddress ip in Dns.GetHostAddresses(domain))
                {
                    Console.WriteLine(ip.ToString());
                }
                Console.WriteLine();
            }
        }
    }
}
```

The `Dns` Class

```
E:\CSHARP_SAMPLES>DnsDemo
www.google.com
209.85.135.106
209.85.135.147
209.85.135.104
209.85.135.99
209.85.135.103
209.85.135.105

www.bing.com
212.25.69.145
212.25.69.160

www.abelski.com
216.154.217.186

www.xperato.com
216.154.217.186

www.zindell.com
216.154.217.186

www.jacado.com
216.154.217.186
```

The `Dns` Class

- Calling the `GetHostEntry` static method we can convert from an IP number to the domain name.

```
...
IPHostName host = Dns.GetHostEntry("216.154.217.186");
Console.WriteLine(host.HostName);
...
```

The Dns Class

```
using System;
using System.Net;
using System.IO;
using System.Threading;
using System.Text;

namespace abelski.csharp
{
    class DNSAnotherDemo
    {
        static void Main()
        {
            string[] ips = {"209.85.135.106","199.238.166.245"};
            foreach(string ip in ips)
            {
                Console.WriteLine(ip);
                Console.WriteLine(Dns.GetHostEntry(ip).HostName);
                Console.WriteLine();
            }
        }
    }
}
```

The `Dns` Class

```
C:\Windows\system32\CMD.exe

E:\CSHARP_SAMPLES>DnsAnotherDemo
209.85.135.106
mu-in-f106.google.com

199.238.166.245
whois.securesites.net


E:\CSHARP_SAMPLES>
```

SMTP

- The `SmtpClient` class enables us to send emails. The `MailMessage` class encapsulates an email message.
- Once `SmtpClient` is instantiated we should assign the SMTP server to the `Host` property.
- Once instantiating `MailMessage` we should assign the required values to its various properties (`Sender`, `From`, `To`, `CC`, `Subject` etc.).

SMTP

- Sending attachments is done by instantiating the `Attachment` class and passing over the reference of the new object to the `Add` method we should call on the `Attachments` property of the `MailMessage` object we are working with.

SMTP

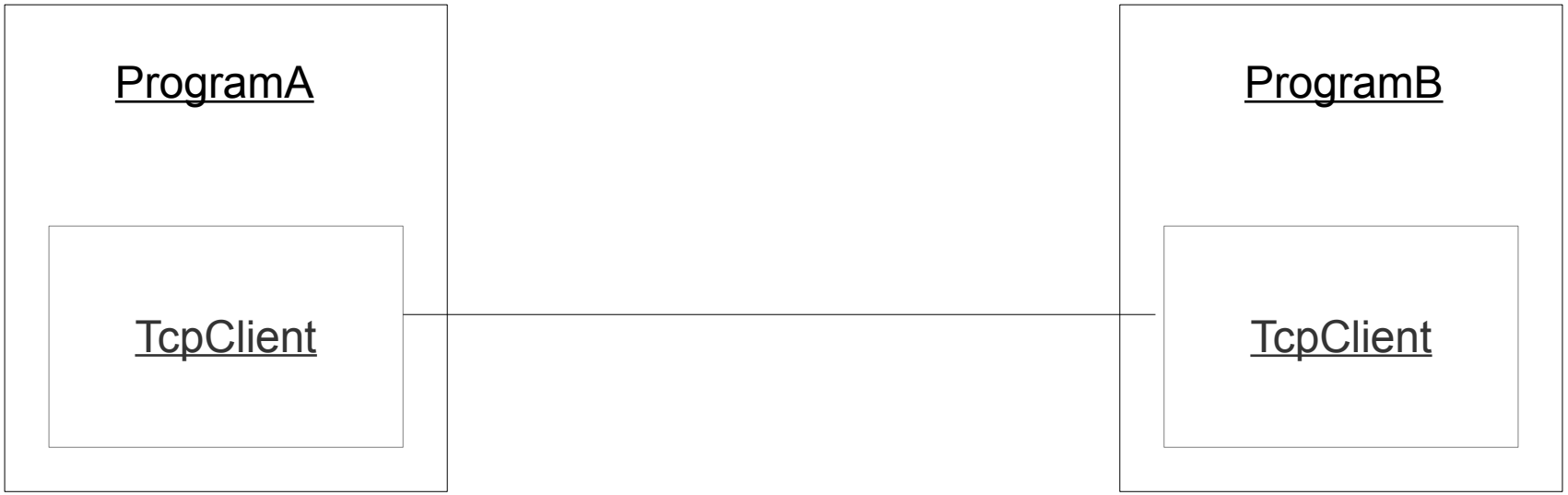
```
...
SmtpClient client = new SmtpClient();
client.Host = "out.bezeqint.net";
MailMessage message = new MailMessage();
message.Sender = new MailAddress("david@jacado.com", "david");
message.From = new MailAddress("david@jacado.com","david");
message.To.Add(new MailAddress("haim.michael@gmail.com","haim"));
message.CC.Add(new MailAddress("haim.michael@zindell.com","michael"));
message.Subject = "Happy New Year!";
message.Body = "Happy New Year to you and your family:) See you soon.";
message.IsBodyHtml = false;
message.Priority = MailPriority.Low;
Attachment attachment = new
    Attachment("photo.jpg",System.Net.Mime.MediaTypeNames.Image.Jpeg);
message.Attachments.Add(attachment);
client.Send(message);
...
```

The TCP Protocol

- The TCP (Transmission Content Protocol) protocol sets the fundamental transport layer. Most of the available Internet services are delivered on top of the TCP layer. HTTP, FTP, SMTP are just few of them.

The `TcpClient` & `TcpListener` Classes

- The `TcpClient` class facade the required underneath classes and methods for connecting, sending data and receiving data over the web in a synchronous blocking mode.



The `TcpClient` & `TcpListener` Classes

- In order to get two programs connected with each other over the web using the `TCP/IP` protocol we should first decide which of the two programs should be executed first so it could function as kind of a server waiting for a request to create a connection with.
- The `TcpListener` class provides simple methods that listen for incoming request to connect and accept it. These methods work synchronously with the thread that calls them.

The `TcpClient` & `TcpListener` Classes

```
using System;
using System.Net;
using System.IO;
using System.Threading;
using System.Text;
using System.Net.Mail;
using System.Net.Mime;
using System.Net.Sockets;

namespace abelski.csharp
{
    class TCPSimpleServer
    {
        static void Main()
        {
            TcpListener listener = null;
            TcpClient client = null;
            NetworkStream stream = null;
            BinaryWriter writer = null;
            BinaryReader reader = null;
```

The `TcpClient` & `TcpListener` Classes

```
try
{
    listener = new TcpListener(
        new IPAddress(new byte[] {127,0,0,1}),1300);
    listener.Start();
    while (true)
    {
        Console.WriteLine("waiting...");
        using (client = listener.AcceptTcpClient())
        {
            using (stream = client.GetStream())
            {
                string sent = "THREE";
                reader = new BinaryReader(stream);
                String received = reader.ReadString();
                if (received.Equals("1")) sent = "ONE";
                else if(received.Equals("2")) sent = "TWO";
                writer = new BinaryWriter(stream);
                writer.Write(sent);
            }
        }
    }
}
```

The `TcpClient` & `TcpListener` Classes

```
            catch(WebException e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                if(listener!=null) listener.Stop();
                if(writer!=null) writer.Close();
                if(reader!=null) reader.Close();
            }
        }
    }
}
```

The `TcpClient` & `TcpListener` Classes

```
using System;
using System.Net;
using System.IO;
using System.Threading;
using System.Text;
using System.Net.Mail;
using System.Net.Mime;
using System.Net.Sockets;

namespace abelski.csharp
{
   class TCPSimpleClient
    {
        static void Main()
        {
            TcpClient client = null;
            NetworkStream stream = null;
            TcpListener listener = null;
            BinaryWriter writer = null;
            BinaryReader reader = null;
```

The `TcpClient` & `TcpServer` Classes

```
try
{
    using (client = new TcpClient("127.0.0.1",1300))
    {
        Console.WriteLine("connection was established...");
        using (stream = client.GetStream())
        {
            string sent = "1";
            string received = null;
            writer = new BinaryWriter(stream);
            writer.Write(sent);
            Console.WriteLine(sent+" was sent...");
            reader = new BinaryReader(stream);
            received = reader.ReadString();
            Console.WriteLine(received+" was received...");
        }
    }
}
```

The `TcpClient` & `TcpServer` Classes

```
            catch(WebException e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                if(listener!=null) listener.Stop();
                if(writer!=null) writer.Close();
                if(reader!=null) reader.Close();
            }
        }
    }
}
```

The `TcpClient` & `TcpServer` Classes

```
C:\Windows\system32\cmd.exe
connection was established...
stream was retrieved from the connection...
1 was sent...
ONE was received...

E:\CSHARP_SAMPLES>TcpSimpleClient
connection was established...
stream was retrieved from the connection...
1 was sent...
ONE was received...

E:\CSHARP_SAMPLES>_
```

```
C:\Windows\system32\CMD.exe - TcpSimpleServer
ONE was sent...
waiting...
connection request accepted...
1 was received...
ONE was sent...
waiting...
connection request accepted...
1 was received...
ONE was sent...
waiting...
```