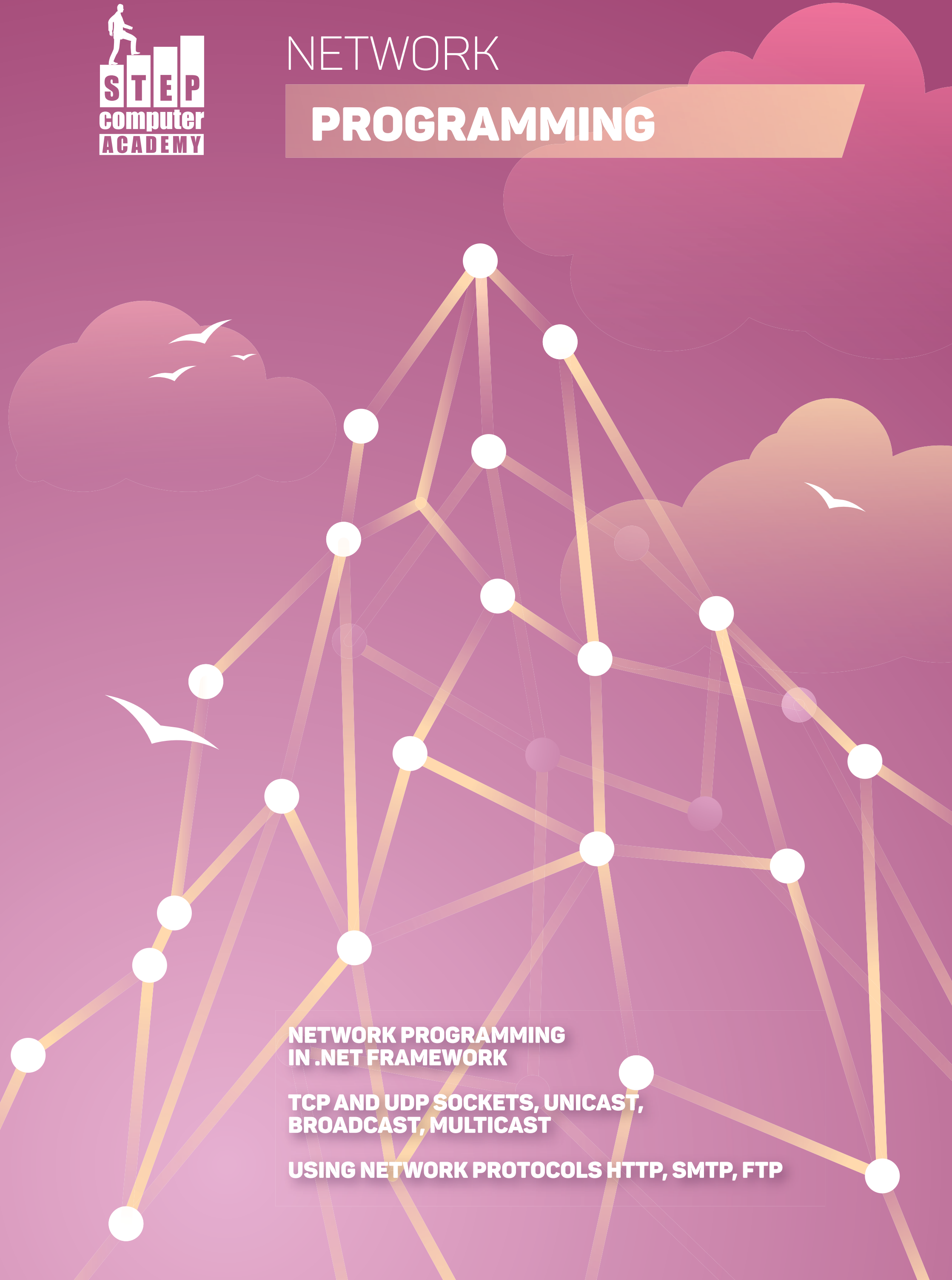
STEP computer ACADEMY

NETWORK PROGRAMMING



NETWORK PROGRAMMING IN .NET FRAMEWORK

TCP AND UDP SOCKETS, UNICAST, BROADCAST, MULTICAST

USING NETWORK PROTOCOLS HTTP, SMTP, FTP

Lesson 4

Use of network protocols HTTP, SMTP, FTP

Contents

1. Review of HTTP Protocol

HTTP is a protocol of an application layer above TCP/IP, which is used for rendering a hypertext in WWW and LANs. Mainly it is used to transfer web-pages contents. Apart from transferring web-pages contents, the protocol is used for data exchange applications. The queries, headings and result codes form the basis of the protocol. There are always two sides expressed within the protocol: a server and a client. The client sends the request in the form:

- Initial string
- Heading (or headings)
- Message body,

the server returns a result in the form:

- Initial string with the result code (or an error code)
- Heading (or headings)
- Message body

Browsers are the clients in WWW, for example IE or Opera. HTTP-servers like Apache and IIS can act in the capacity of servers.

Initial string of the client query is a method that is requested by the URI and protocol version (HTTP-Version). HTTP-headings can be divided into three groups: query headings, reply headings and headings that can be met both within the request and reply. Fields of a query heading allow a client transmitting additional information about the query and the very client to the server.

Table 1.Heading titles of HTTP-queries

No.	Heading	Purpose
1	Accept	A list of content types supported by a browser in the browser preference order
2	Accept-Charset	Supported encoding. It matters for a server that can output the same document in different encodings
3	Accept-Encoding	Supported encoding type. It matters for a server that can differently encode the same document
4	Accept-Language	Supported language. It matters for a server that can output the same document in different language versions
5	Authorization	User authorization attributes
6	From	Client address
7	Host	Host name from which a resource is requested
8	If-Modified-Since	Action in case of changed content
9	If-Match	Act in case of coincidence
10	If-None-Match	Act in case of inconsistency
11	If-Range	Act in case of over dimensioning
12	If-Unmodified-Since	Act in case of unchanged contents
13	Max-Forwards	Maximum number of references
14	Proxy-Authorization	User proxy authorization attributes
15	Range	Dimensions
16	Referer	URL from which we directed to this resource
17	User-Agent	Browser

Initial string of server response is a ***Status-Line***. It consists of the protocol version (***HTTP-Version***), ***Status-Code*** and ***Reason-Phrase*** that are separated by the symbols SP (space). CR (carriage return) and LF (line feed) are not acceptable in a ***Status-Line*** except the final CRLF sequence.

```
Status-Line = HTTP-Version SP Status-Code
        SP Reason-Phrase CRLF.
```

Table 2. Codes HTTP-server responses

No.	Status-Code	Reason-Phrase
1	100	Continue
2	101	Switching Protocols
3	200	OK
4	201	Created
5	202	Accepted
6	203	Non-Authoritative Information
7	204	No Content
8	205	Reset Content
9	206	Partial Content
10	300	Multiple Choices
11	301	Moved Permanently
12	302	Moved Temporarily
13	303	See Other
14	304	Not Modified
15	305	Use Proxy
16	400	Bad Request
17	401	Unauthorized
18	402	Payment Required
19	403	Forbidden
20	404	Not Found
21	405	Method Not Allowed
22	406	Not Acceptable
23	407	Proxy Authentication Required
24	408	Request Timeout
25	409	Conflict
26	410	Gone
27	411	Length Required
28	412	Prerequisite Wrong

No.	Status-Code	Reason-Phrase
29	413	Request Entity Too Large
30	414	Request-URI Too Long
31	415	Unsupported Media Type
32	500	Internal Server Error
33	501	Not Implemented
34	502	Bad Gateway
35	503	ServiceUnavailable
36	504	Gateway Timeout
37	505	HTTP Version Not Supported

Client or server CAN send an object (message). An entity consists of an ***entity-header*** and ***entity-body***, though some responses can include ***entity-headers*** only. An entity can be sent either by client, or by server. Entity-body (if any) is sent together with the HTTP query or response and has the format and encoding that is defined by the ***entity-header fields***. ***Entity-body*** is only represented in the message when there is a ***message-body***. ***Entity-body*** is received from the ***message-body*** by means of decoding indicated within a ***Transfer-Encoding*** field. His body data type is defined by the heading fields ***Content-Type*** and ***Content-Encoding***. He define two-level encoding ordered model: ***entity-body:= Content-Encoding(Content-Type(data)). Content-Type*** defines main data media type (text, image, etc.). ***Content-Encoding*** can be used for indicating any additional content encoding applied to the data (usually for data compressing). ***Content-Encoding*** is a property of requested resource. There is no a default encoding. Within any HTTP/1.1 there is ***entity-body*** content. In the case when media type is not represented by the field ***Content-Type***, a receiver CAN try assume a media type verifying the content and/or extension (extensions) in the URL name used for an identified resource. If media type is remained unrecognized a receiver SHOULD process it as a “***application/octet-stream***” type.

1.1. Example of HTTP-Request

```
GET /default.aspx HTTP/1.1
```

1.2. Example of HTTP-Response

```
HTTP/1.1 200 OK
Date: Wed, 11 Feb 2009 11:20:59 GMT
Server: Apache
X-Powered-By: PHP/5.2.4-2ubuntu5wm1
Last-Modified: Wed, 11 Feb 2009 11:20:59 GMT
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close
(then there is a requested page in HTML)
```

For detailed information about HTTP-protocol use the links:
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
http://tools.ietf.org/html/rfc1945
http://tools.ietf.org/html/rfc2616

2. Classes for Working with the HTTP

.NET platform immensely simplified the interaction with the HTTP-server by means of submitting of several high-level namespace classes System.Net for maintaining the protocol; these classes allow managing the headings and connections, to exercise preliminary authentication (verification a client's personality), encoding, support of working with the proxy-server, pipeline processing:

Table 3. .Net classes for working with the HTTP

	Class	Namespace	Basic class	Description
1	HttpWebRequest	System.Net	WebRequest	HTTP-query
2	HttpWebResponce	System.Net	WebResponce	HTTP-response
3	WebClient	System.Net	Component	Simple methods of receiving and sending data for URI
4	ServicePoint	System.Net	Object	Handling a connection with URI
5	ServicePoint-Manager	System.Net	Object	Manages ServicePoint objects
6	Uri	System	MarshalByRef Object	Light managing URI
7	UriBuilder	System	Object	Creating and modifying URI objects

2.1. Sending Requests Using HttpWebRequest Class

HttpWebRequest represents an opportunity to send the query. Let's create and send the query:

```
HttpWebRequest reqw=(HttpWebRequest)HttpWebRequest.
Create(«http://itstep.org»);
reqw.GetResponse();
```

2.2. Receiving the Replies Using HttpWebResponse Class

However, sending a request without receiving a result is a senseless act. Let's elaborate a program and display on a screen a resource content (HTML-page).

```
HttpWebRequest reqw=(HttpWebRequest)HttpWebRequest.
    Create(«http://itstep.org»);
HttpWebResponse resp= (HttpWebResponse)
    reqw.GetResponse();// create a response object
StreamReader sr=new StreamReader(resp.
    GetResponseStream(),Encoding.Default);//create a
    //stream for reading a response
Console.WriteLine(sr.ReadToEnd()); // to display on a
    //screen everything that can be read
sr.Close();
```

2.3. Setting Up Query Headings

Query headings allow us finely establishing interaction with HTTP-server, for example to set up a default language

```
HttpWebRequest reqw=(HttpWebRequest)HttpWebRequest.
    Create(«http://itstep.org»);
reqw.Headers.Add("Accept-Language: ru-ru");//setting
    //up a default Russian language
```

Besides, in order to set up the values for some headings we can use the properties of the class.

Table 4. Properties of WebRequest class

	Property	Data type	HTTP-heading
1	Accept	String	Accept
2	Connection	string	Connection
3	ContentLength	Long	Content-Length
4	Expect	String	Expect
5	IfModifiedSince	DateTime	If-Modified-Since
6	Referer	String	Referer
7	TransferEncoding	String	Transfer-Encoding
8	ContentType	String	Content-Type
9	UserAgent	String	User-Agent

To setup the headings through reqw.Headers.Add ***is impossible***.

2.4. Reading Reply Headings

Headings reading are available from the Headers collection or from the other properties of HttpWebResponse class indicated within a table.

Table 5. Properties of WebResponse class

	Property	Data type	HTTP-heading
1	ContentEncoding	String	ContentEncoding
2	ContentLength	Long	ContentLength
3	ContentType	String	ContentType
4	LastModified	DateTime	LastModified
5	Server	String	Server

Headings reading from the collection.

```
HttpWebResponse resp= reqw.GetResponse();
foreach(string header in resp.Headers)
Console.WriteLine("{0}:{1}",header,resp.
    Headers[header]);
```

2.5. Use of WebClient Class

WebClient class simplifies data exchange with the server. For receiving the data we use DownloadData() method

```
//creating a web-client object
WebClient client= new WebClient();
//receiving page content
byte[] urlData = client.DownLoadData
    ("http://www.yandex.ru");
//transformation of receivd content into a string
//for displaying within a console
string page = Encoding.ASCII.GetString(urlData);
Console.WriteLine(page);
```

For obtaining the file from a server we use DownloadFile() method

```
//creating a web-client object
WebClient client= new WebClient();
string fileCopy = "c:\\ttt.gif",
    urlString="http://www.yandex.ru/images/point.gif";
//loading of a web-esource into a file with the
//name fileCopy
client.DownloadFile(urlString,fileCopy);
```

for reading the data in parts we use OpenRead() method obtaining a stream available for reading:

```
//creating a web-client object
WebClient client= new WebClient();
string fileCopy = "c:\\ttt.gif",
     urlString="http://www.yandex.ru/images/point.gif";
//connect URL to reading stream
Stream copy=client.OpenRead(urlString);
```

```
//create a stream for storing the downloaded data
FileInfo fi=new FileInfo(fileCopy);
StreamWriter sw=fi.CreateText();
//loading of a web-resource into a file with
//the name fileCopy
sw.WriteLine(copy.ReadToEnd());
sw.Close();
copy.Close();
```

for data transfer to a server we use OpenWrite() method that returns the stream available for record (used for HTTP-transfer POST):

```
//creating a web-client object
WebClient client= new WebClient();
string TextToUpload ="User=Vasia&passwd=okna",
    urlString="http://www.yandex.ru/page22.aspx";
//transform the text into an array of bytes
byte[] uploadData=Encoding.ASCII.
GetBytes(TextToUpload);
//bind URL with a record stream
Stream upload=client.OpenWrite(urlString,"POST");
//upload data to a server
upload.Write(uploadData,0,uploadData.Length);
upload.Close();
```

for data transfer to a server by other HTTP-methods we use UploadData() method

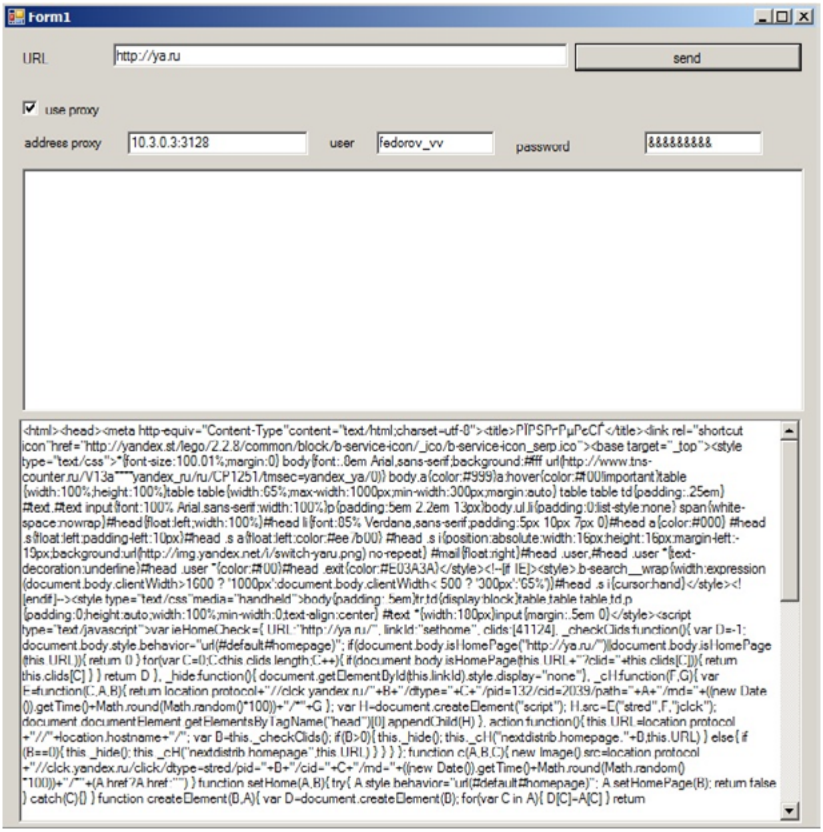
```
//creating a web-client object
WebClient client= new WebClient();
client.Credentials = System.Net.CredentialCashe.
    DefaultCredentials;
//add a HTTP-heading
client.Headers.Add("Content-Type",
    "application/x-www-form-urlencoded");
```

```
string TextToUpload = “User=Vasia&passwd=okna”,
    urlString=”http://www.yandex.ru/page22.aspx”;
//transform a text into an array of bytes
byte[] uploadData=Encoding.ASCII.GetBytes(TextToUpload);
//copy data by means of GET method
byte[] respText=client.
    UploadData(urlString,”GET”,uploadData);
//upload the data onto the server
upload.Write(uploadData,0,uploadData.Length);
upload.Close();
```

Here is an example of the functioning of HttpRequest and HttpResponce classes.

```
namespace HTTP_Client
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void sendButton_Click(object sender,
                EventArgs e)
        {
            HttpWebRequest req = (HttpWebRequest)
                HttpWebRequest.Create(URL.Text);
            req.Method = “GET”;
            if (IfProxy.Checked)
            {
                WebProxy proxy = new
                         WebProxy(proxyAddr.Text);
                proxy.Credentials = new
                         NetworkCredential(proxyUser.
                         Text, proxyPassword.Text);
                req.Proxy = proxy;
            }
            HttpWebResponse rez = (HttpWebResponse)
                req.GetResponse();
            StreamReader sr = new StreamReader(rez.
                GetResponseStream(), Encoding.Default);
            response.Text = sr.ReadToEnd();
        }
    }
}
```

In the result we receive an application that can read an html-page and displaying it in the text editor in the text type:



```
http://ya.ru
[✓] use proxy
address proxy 10.3.0.3:3128   user fedorov_vv   password &&&&&&&&&
```

```
<html><head><meta http-equiv="Content-Type"content="text/html;charset=utf-8"><title>PIPSPrPµPeCЃ</title><link rel="shortcut
icon"href="http://yandex.st/lego/2.2.8/common/block/b-service-icon/_ico/b-service-icon_serp.ico"><base target="_top"><style
type="text/css">*{font-size:100.01%;margin:0} body{font:.8em Arial,sans-serif;background:#fff url(http://www.tns-
counter.ru/V13a****yandex_ru/ru/CP1251/tmsec=yandex_ya/0)} body.a{color:#999}a:hover{color:#f00!important}table
{width:100%;height:100%}table table{width:65%;max-width:1000px;min-width:300px;margin:auto} table table td{padding:.25em}
#text,#text input{font:100% Arial,sans-serif;width:100%}p{padding:5em 2.2em 13px}body.ul.li{padding:0;list-style:none} span{white-
space:nowrap}#head{float:left;width:100%}#head li{font:85% Verdana,sans-serif;padding:5px 10px 7px 0}#head a{color:#000} #head
.s{float:left;padding-left:10px}#head .s a{float:left;color:#ee7b00} #head .s i{position:absolute;width:16px;height:16px;margin-left:-
19px;background:url(http://img.yandex.net/i/switch-yaru.png) no-repeat} #mail{float:right}#head .user,#head .user *{text-
decoration:underline}#head .user *{color:#f00}#head .exit{color:#E03A3A}</style><!--[if IE]><style>.b-search__wrap{width:expression
(document.body.clientWidth>1600 ? '1000px':document.body.clientWidth< 500 ? '300px':'65%')}#head .s i{cursor:hand}</style><!
[endif]--><style type="text/css"media="handheld">body{padding:.5em}tr,td{display:block}table,table table,td,p
{padding:0;height:auto;width:100%;min-width:0;text-align:center} #text *{width:180px}input{margin:.5em 0}</style><script
type="text/javascript">var ieHomeCheck={ URL:"http://ya.ru/", linkId:"sethome", clids:[41124], _checkClids:function(){ var D=-1;
document.body.style.behavior="url(#default#homepage)"; if(document.body.isHomePage("http://ya.ru/")||document.body.isHomePage
(this.URL)){ return 0 } for(var C=0;C<this.clids.length;C++){ if(document.body.isHomePage(this.URL+"?clid="+this.clids[C])){ return
this.clids[C] } } return D }, _hide:function(){ document.getElementById(this.linkId).style.display="none"}, _cH:function(F,G){ var
E=function(C,A,B){ return location.protocol+"//clck.yandex.ru/"+B+"/dtype="+C+"/pid=132/cid=2039/path="+A+"/rnd="+((new Date
()).getTime()+Math.round(Math.random()*100))+"/*"+G }; var H=document.createElement("script"); H.src=E("stred",F,"jclck");
document.documentElement.getElementsByTagName("head")[0].appendChild(H) }, action:function(){ this.URL=location.protocol
+"//"+location.hostname+"/"; var B=this._checkClids(); if(B>0){ this._hide(); this._cH("nextdistrib.homepage."+B,this.URL) } else{ if
(B==0){ this._hide(); this._cH("nextdistrib.homepage",this.URL) } } } }; function c(A,B,C){ new Image().src=location.protocol
+"//clck.yandex.ru/click/dtype=stred/pid="+B+"/cid="+C+"/rnd="+((new Date()).getTime()+Math.round(Math.random()
*100))+"/*"+(A.href?A.href:"") } function setHome(A,B){ try{ A.style.behavior="url(#default#homepage)"; A.setHomePage(B); return false
} catch(C){} } function createElement(B,A){ var D=document.createElement(B); for(var C in A){ D[C]=A[C] } return
```

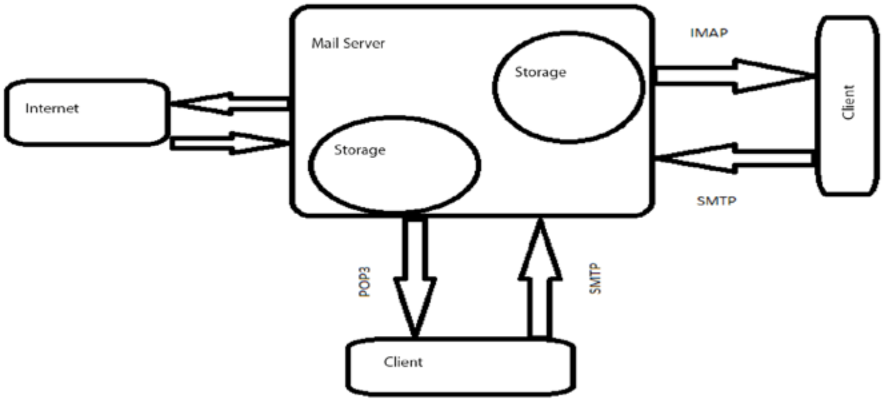
"

3. Working with Electronic Mails

2.1. Overview of Post Office Protocols SMTP, POP3, IMAP

Today sending and receiving electronic mails is performed by means of post office protocols (POP) that use TCP/IP in the capacity of transport protocol. To send electronic mails we use POP SMTP, and we use POP3 for receiving thereof. Existing IMAP protocol cannot manage both tasks, but most often we use it instead of POP3.

Scheme of sending and receiving mails



2.1. SMTP Protocol

Overview

SMTP Protocol (Simple Mail Transport Protocol) defines interaction between the servers transporting electronic mails between a client and a server (RFC 2821). Interaction is carried on by means of transfer of a limited number of commands that are followed by data.

SMTP Terminology

SMTP-server is a program waiting for a client's query for mail sending or for receiving thereof from another SMTP-server. ***SMTP-client*** is program addressing the server to send it a mail. SMTP-server stores a database of ***registered users*** and their passwords. Therefore, sending mail messages from unauthorized users is impossible. ***Mailbox*** is memory area (often a catalogue) on a server that stores received mails designated for a client**.** ***Mail message*** is a text block containing headings, message body and attachment (one or several). Message ***headings*** define a recipient, sender, mail address, content type, availability of attachments, encoding type, text encoding, message type (text-html) and others. ***Attachment*** is an additional file that can have any type content (image, audio file, video, program…).

SMTP Commands

Commands are sent to SMTP-server in the text form. The first one is a text command, then its parameters. Server response is accompanied by an error code that is followed by reason. Exchange session can look as follows:

```
open mail.olgs.gov 25
Trying...Connect to mail.olgs.gov
220 mail.olgs.gov - Server ESMTP (PMDF V4.3-10 #2381)

hello mysite.com
250 mail.olgs.gov OK, mysite.com

mail from:<habitant@mysite.com>
250 Address Ok.

rcpt to:<kravchuk@ mail.olgs.gov>
250 kravchuk@ mail.olgs.gov OK

data
354 Enter mail, end with a single ".".
SUBJECT: E-mail chapter
Ljonja, thanks for the live
.
250 OK

quit
221 Bye received. Goobye
```

Table 5. Overview of some SMTP-server commands

No.	Command (and its 4-letter variant)	Description
1	HELLO (HELO)	SMTP-client identification
2	MAIL (MAIL)	Initiates mail transaction (mail delivery) to mailboxes
3	RECEPIENT (RCPT)	Recipient identification. Several commands per session are allowed
4	DATA (DATA)	Beginning of mail data
5	SEND (SEND)	Initiates mail transaction to the terminals (outdated)

No.	Command (and its 4-letter variant)	Description
6	SEND или MAIL (SOML)	If a recipient is active mails are sent to the terminal, otherwise to a mailbox
7	SEND и MAIL (SAML)	Receives mails on the terminal and mailbox
8	RESET (RSET)	Breaks up a current mail transaction
9	WERIFY (WRFY)	Requites a recipient to verify that its argument is a true name.
10	EXPAND (EXPN)	Command to a SMTP-recipient to verify if an argument is an address of mail-out and if yes it returns a message address
11	HELP (HELP)	Command to SMTP-recipient to return inquiry-message about its commands
12	NOOP (NOOP)	Requires from a sender not to perform any actions, but to give a response OK. It is mainly used for testing
13	QUIT (QUIT)	Requires to give a response OK and close the current connection
14	TURN (TURN)	Command to SMTP-recipient either to give OK and change the roles, i.e. to become a STMP- recipient, or send a re-ject-message and remain in the role of an SMTP-recipient

Using exchange commands we can write an smtp-client on our own. MS.NET Framework represents a convenient set of classes in order to simply build-in a mail-exchange within our applications.

3.3. .Net for Classes for Working with SMTP

.Net contains a set of classes for creating a SMTP-client and does not contain any classes for creating SMTP-server.

At first, we describe an order of message sending using .Net classes.

Then, a message is created and a client class afterwards; a message is attached to a client class and a client sends a mail message. If we need to add some attachments we should create attachment samples that are added to a message sample. Afterwards a received container is added to a client. If we change the attachments, information thereof is to be contained within the header. To perform all the actions with the mails System.Net and System.Net. Mail spaces are required.

System.Net.Mail name space contains 3 classes and 3 enumerations:

Table 6. System.Net.Mail space

	Class	Description
1	MailMessage	Electronic mail message
2	SmtpClient	Realizes sending MailMessage through SMTP
3	MailAttacment	Attachment within a mail message

	Перечисление	Description
1	MailEncoding	Encoding type Base46 or UUEncode
2	MailFormat	Message format Text or HTML
3	MailPriopity	High, Medium, Low – message priority

Let’s review main classes out of System.Net.Mail name space and create a mail message without an attachment and with an attachment.

3.4. MailMessage Class

Main class is for message forming. It contains mail message headings, message body and a collection of attachment files. The headers include recipient’s address, sender’s address, message date, copy address, forward address, hidden reference address, etc.

Table 7. Some member of MailMessage class

No.	Property	Description
1	Attachments	Attachment collection
2	Bcc	List for sending the copies in the form of a mail addresses string divided by _;_ (Blind Carbon Copy)
3	Body	Message body
4	BodyFormat	Electronic mail format MailFormat.Text or MailFormat.Html
5	Cc	List for sending the copies in the form of a mail addresses string divided by _;_ (Carbon Copy)
6	From	Sender's mail address
7	Subject	Message subject
8	To	Recipient mail address
9	Priority	Message priority (MailPriority enumeration)
10	UrlContentBase	HTTP-heading Content-Base. Base for all relative Url
11	UrlContentLocation	HTTP-heading Content-Location
12	Headers	List of non-standard headings sent together with the message

Creating mail message:

```
MailMessage post=new MailMessage();
post.From = «vasily@pupkin.com»;
post.To= «Lusi@pupkin.com»;
post.Subject= «Test message»;
post l.BodyFormat  = MailFormat.Text;
post.Body = «post message»;
```

Such mail message can be sent in fact.

3.5. Attachment Class

This class is designated for creating an attachment entity, which is further to be added to the MailMessage.Attachments collection.

Class has 2 important properties from the viewpoint of mail sending: Attachment.Filename — name of an attached file, Attachment.Encoding — attachment of encoding type (MailEncoding.Base64 and MailEncoding.UUEncode). Example of creating a mail message with an attachment:

```
MailMessage post=new MailMessage();
post.From = «vasily@pupkin.com»;
post.To= «Lusi@pupkin.com»;
post.Subject= «Spring Calndar»;
post l.BodyFormat  = MailFormat.Text;
post.Body = «1 April»;
MailAttachment at=new MailAttachment();
at.Filename= @«C:\MyHohma.Jpg»;
post.Attachments.Add(at);
```

3.6. SmtpClient Class

It is designated for sending mail message to the SMTP-server. It contains the diagnostic tools for successful sending and asynchronous sending of mails. It does not allow learning about what happens to the mail next (whether the mail was delivered to the recipient). There is a message encoding opportunity using the SSL protocol (Secure Socket Level).

Table 8. Main properties and methods of SmtpClient class

	Method	Purpose
	Send	Mail sending
	SendAsync	Non-blocking mail sending
	SendAsyncCancel	Terminating an mail sending non-blocking operation
	Property	Description
	EnableSsl	Encoding use
	Host	String — server address
	Port	Integer — port number
	TimeOut	Command Send terminating timeout
	Событие	Description
	SendCompleted	Terminating an asynchronous mail sending operation

3.7. Example of Mail Sending

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net.Mail;

namespace Mailer
{
    class Program
    {
        string prompt(string query)
        {
            Console.WriteLine(query);
            return Console.ReadLine();
        }
        static void Main(string[] args)
        {
```

```
        Program app = new Program();
        //fill in the fields of mail message
        app.Dialog();
        //attempt to send a message
        app.SendMail();
    }
    string to;
    string from;
    string subject;
    string body;
    string server;

    void Dialog()
    {
        to = prompt(“Input a recipient address:”);
        from = prompt(“Input a sender’s address:”);
        subject = prompt(“Input a subject”);
        body = prompt(“Input message text:”);
        server = prompt(“Input a server address:”);
    }

    public void SendMail()
    {
        MailMessage message = new
            MailMessage(from, to, subject, body);
        SmtpClient client = new
            SmtpClient(server);
        Console.WriteLine(“count to 100”);
        client.Timeout = 10000;//set up TimeOut
            //10000 milliseconds

        client.UseDefaultCredentials = true;
        try
        {
            client.Send(message);
            Console.WriteLine(“Message sent”);
        }
```

```
                catch (SmtpException se)
                {
                    Console.WriteLine("Message not sent
                            due to "+se.Message);
                }
            }
        }
}
```

In case of use a multi-window window application it is not recommended to make a user wait till a message is sent, it's better to offer him to use an asynchronous mail sending (you should change server address, login and password in NetworkCredential):

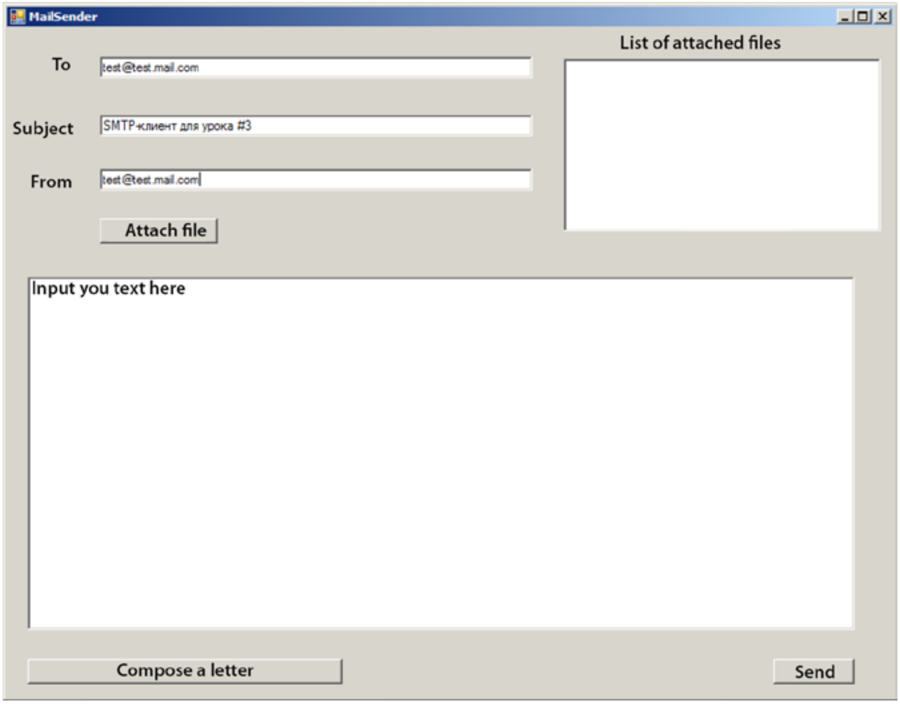
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net.Mail;
using System.Net;

namespace MailSender
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
        }
        string server=”100.30.0.30”;//here we set up
            //a server address

        private void button2_Click(object sender,
            EventArgs e)
        {
           //create message entity
           MailMessage message = new
               MailMessage(fromBox.Text, toBox.Text,
               themeBox.Text,bodyBox.Text);
           //create sending entity
           SmtpClient client = new SmtpClient(server);
           client.Port = 25;//here we set up a server port
           //options for sending mails
           //(login and password
           client.Credentials = new
                NetworkCredential(“test”,”test”);
           //request an asynchronous mail sending
           client.SendAsync(message,”That’s all”);
        }
    }
}
```

This how an application looks like (the very application is within the folder ***source).***



To: test@test.mail.com

Subject: SMTP-клиент для урока #3

From: test@test.mail.com

Input you text here

3.8. POP3 Protocol

Post Office Protocol (POP3) defines interaction between a server and client for receiving mail by a client. Server waits for connection with a client, authorizes him, and performs client’s commands related to his work with the mailbox and finalizing operations (session): «POP3 session consists of several modes. As soon as connection with a server is established and server has sent its invitation, a session transits into the AUTHORIZATION mode. In this mode a client should identify himself on the server. After successful authentication a session transits into TRANSACTION mode (transfer). In this mode a client requests from a server to perform particular commands. When a client sends QUIT command a session transits into an UPDATE mode. In this mode POP3 server releases all occupied resources and terminates its work» (according to the materials from www.codenet.ru) .

Example of Typical POP3 Session

```
S: +OK POP3 server ready

C: USER MonstrVB
S: +OK MonstrVB is a real hoopy frood

C: PASS mymail
S: +OK MonstrVB's maildrop has 2 messages (320
octets)

C: QUIT
S: +OK dewey POP3 server signing off
```

Protocol Commands

Protocol commands

APOP [name] [digest]

A command implying sending user's name and encoded password (digest) to a server.

USER [name]

Sends user's name to a server.

PASS [password]

Sends a mailbox password to a server

DELE [N messages]

Server flags an indicated message for deletion. Flagged for deletion messages are removed only after the transaction closure (transaction closure usually happens after sending QUIT command; besides, for example on the servers transactions closure can happen after some time set up by a server).

LIST [N messages]

If there was an argument sent, then a server gives information about an indicated message. If an argument was not sent the server gives information about all messages that are within a mailbox. Messages flagged for deletion are not enumerated.

NOOP

Server does nothing and responses positively.

RETR [N messages]

Server sends a message with an indicated number.

RSET

This command is used for transaction roll-back inside the session. For example, if a user accidentally flagged any message for deletion, he can take off a flag by sending this command.

STAT

Server returns a number of mails in a mailbox plus a size these mails take within a mailbox

TOP [N messages] [number of strings]

Server returns headings of an indicated message, empty string and indicated number of the first lines of a message body.

QUIT

Session termination

At the moment.Net Framework does not contain any classes for working with POP3-server.

4. Use of FTP

Overview

FTP means a user interface that realizes ARPANET standard file transfer protocol. This program allows user transferring files between two computers interconnected by LAN or WAN. Wherein, computer platforms can be of different types. This is the main feature of FTP within the network.

If your system has an FTP and connected to the Internet, you will get an access to a great number of archives stored within the other systems.

FTP Terminology

While working with FTP we use a client-server model. A client sends queries and receives responses along two ports of TCP-connection (20 — data transfer port and 21 — command transfer port). Remote directory is a file catalogue stored at the server. A deleted file is a file on a server. Local file is a file within a client file system.

Some FTP-commands:

dir [deleted_directory] [local_file]
ls [deleted_directory] [local_file]

Displays a list of files within a directory either for a standard output, or into this file if there is indicated a name of a local file.

get [deleted_file] [local_file]

Requests sending the deleted file copies to your computer.

In case a local file name was not assigned it will coincide with the name of a deleted file.

mget [deleted_files]

For receiving several files

hash

Serves as a switch for indicating each received data block of 1024 bytes; increases the demonstrativeness of the procedure.

cd [deleted_directory]

To change a directory 'cdup' or 'cd' also exist for returning by one or more

lcd

Changes a working directory on a local machine (without an argument — transition into a user home directory)

bin (or binary)

Enables binary file transfer mode

ascii

Switches for a text files transfer mode (usually default).

prompt

Enables interactive hint. Often when we use a command 'mget' it would be good to input 'prompt' so that to avoid multiple verifications.

pwd

Displays a name of a remote working directory.

mkdir [directory_name]

Creates a directory on a remote machine

open host [port]

Establishes a connection with an assigned FTP server

put [local_file] [deleted _file]

Forwards file to remote system. If a name of a remote file is not indicated, then it coincides with the name within a local system.

quit

Synonym for 'bye'

recv [deleted _file] [local _file]

It is a synonym for a command 'get'

reget [deleted _file] [local _file]

"Finish of receiving" a remote file in case a part there of is already on a local machine. Command is especially useful for receiving big files upon various connections break ups.

delete [deleted _file]

Removes a deleted file

close

Breaks FTP session with a remote server and returns to a command interpreter

bye

Terminates the work with FTP server and leads to exit from an interpreter.

4.3. Example of Typical FTP Session

```
220 FTP server ready.
USER ftp //Anonymus
230 Login successful.
PASV
227 Entering Passive Mode (192,168,254,253,233,92)
//Client should open a connection to a transmitted IP
LIST
150 Here comes the directory listing. //Server
    //transmits a list of files within a directory
```

```
226 Directory send OK.
CWD incoming
250 Directory successfully changed.
PASV
227 Entering Passive Mode (192,168,254,253,207,56)
STOR gyuyfotry.avi
150 Ok to send data. //Client transmits a file contents
226 File receive OK.
QUIT
221 Goodbye.
```

4.4. Classes for Working with FTP

```
System.Object
  System.MarshalByRefObject
    System.Net.WebRequest
      System.Net.FtpWebRequest
```

FtpWebRequest is a class for connection with the FTP-server. Using the methods of this class we initiate FTP-commands transmission to a server.

Table 8. Main properties and methods of FtpWebRequest class

No.	Method	Description
1	WebRequest.Create	Static method for creating an object FtpWebRequest
2	GetResponce	Receives a response from FTP-server in a form of FtpWebResponce
3	GetRequestStream	Receives a stream for data upload to a server
4	BeginGetResponce	Beginning of an asynchronous query transfer to a server and receiving a re-sponse
5	EndGetResponce	Termination of asynchronous operation of receiving a server's response

No.	Method	Description
6	BeginGetRequestStream	Asynchronous opening a stream for uploading to a server
7	EndGetRequestStream	Termination of asynchronous operation of receiving a stream for uploading to a server
8	Property	Description
9	Method	Returns or sets up an FTP-command
10	TimeOut	Timeout of asynchronous operation in milliseconds
11	UsePassive	Passive or active transmission mode
12	UseBinary	Data types upon transmission
13	RenameTo	Name of a renaming file
14	ReadWriteTimeOut	Timeout of reading or record
15	Proxy	Description of a proxy-server
16	EnableSsl	Use of encoding upon transmission
17	ContentType	Content type
18	Credentials	Data for connecting a server
19	ContentOffset	Shifting of an uploading file

```
System.Object
  System.MarshalByRefObject
    System.NetWebResponse
      System.Net.FtpWebResponse
```

FtpWebResponse class receives a server response and used for processing the results of this response.

Table 8. Main properties and methods of FtpWebResponse class

No.	Methods	Purpose
1	GetResponseStream	Receives a stream for file reading or uploading to a server
2	Close	Releases the resources
3	Properties	Description

No.	Methods	Purpose
4	ContentType	Content type
5	ContentLength	File length
6	BannerMessage	Preliminary server notice
7	ExitMessage	Message upon termination of FTP-session
8	WelcomeMessage	Server message after connection
9	LastModified	Date of the last file change
10	StatusCode	Current state of FTP-session
11	StatusDescription	Description of the current FTP-session state
12	IsFromCache	Whether this response was received from cash

4.5.Example of FTP Use

File delete on a server

```
string FileName=”ftp://glamurconcurs.com/lucie.jpg”;
//create an FTP- query entity
FtpWebRequest request = (FtpWebRequest)WebRequest.
Create(FileName);
//set up an FTP-command
request.Method = WebRequestMethods.Ftp.DeleteFile;
//execute a query and get a result
FtpWebResponse response = (FtpWebResponse)
                           request.GetResponse();
//process a result
Console.WriteLine(“Delete status: {0}”,response.
StatusDescription);
//close a session
response.Close();
```

5. Exam Tasks

Variants of Examination Home Assignment (select one):

1. Build web-site tree. Restrict the references to other sites by 1 level. Consider the possibility of self-references.
2. Load a web-site. Downloads by the references to the other site are not allowed. Consider the possibility of self-references.
3. Write a chat (message exchange software between several users) on the basis of the sockets with an allocated server
4. Write a chat (message exchange software between several users) on the basis of the datagrams
5. Write a program of network checkers game with a possibility of fan presence.
6. Loading program from FTP-server with a possibility of resume of an interrupted load in a particular time interval of changes of loaded files on a server (updating)
7. Write a remote-access server of several clients for the base MS ACCESS and the simplest client with a SQL-query editor.

Lesson 4

Use of network protocols HTTP, SMTP, FTP