Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

Курсовая работа

по дисциплине «Сети ЭВМ и телекоммуникации»  
«Почтовый клиент IMAP»

**Выполнил:**  
студент группы ПВ-31  
Адаменко И. И.

**Руководитель:**старший преподаватель  
Федотов Е. А.  
  
**Оценка:**

Белгород  
2015

Оглавление

[Введение 3](#_Toc421600153)

[Сравнительный анализ сетевых протоколов 4](#_Toc421600154)

[SMTP 4](#_Toc421600155)

[POP 4](#_Toc421600156)

[IMAP 5](#_Toc421600157)

[Детальный разбор протокола IMAP 8](#_Toc421600158)

[Взаимодействие клиента и сервера 8](#_Toc421600159)

[Команды протокола IMAP 9](#_Toc421600160)

[Состояния сервера IMAP 12](#_Toc421600161)

[Разработка пользовательского интерфейса 14](#_Toc421600162)

[Разработка основных алгоритмов 17](#_Toc421600163)

[Аутентификация 17](#_Toc421600164)

[Получение каталогов с сообщениями 17](#_Toc421600165)

[Выбор рабочего каталога 17](#_Toc421600166)

[Получение списка сообщений 17](#_Toc421600167)

[Получение отдельного сообщения 18](#_Toc421600168)

[Структура программы 19](#_Toc421600169)

[Заключение 20](#_Toc421600170)

[Список литературы 20](#_Toc421600171)

[Приложение 1. Код программы 20](#_Toc421600172)

[MainForm.cs 20](#_Toc421600173)

[Tools.NewMail.cs 20](#_Toc421600174)

[Tools.Settings.cs 20](#_Toc421600175)

[Utils.Base64Converter.cs 20](#_Toc421600176)

[Utils.UTF7Converter.cs 20](#_Toc421600177)

[IMAP.Commands.cs 20](#_Toc421600178)

[IMAP.IMAPClient.cs 20](#_Toc421600179)

[SMTP.SMTPClient.cs 24](#_Toc421600180)

# Введение

Целями данной курсовой работы являются: закрепление знаний и навыков, полученных в рамках курса «Сети ЭВМ и телекоммуникации», и получение практического опыта реализации протокола IMAP при разработке почтового клиента на его основе.

Задача этой курсовой работы состоит в следующем:

1. Изучить протокол IMAP.
2. Разработать пользовательский интерфейс.
3. Разработать алгоритмы взаимодействия с IMAP-сервером, а именно:
   1. аутентификация пользователя;
   2. просмотр почты пользователя с вложениями;
   3. удаление письма;
   4. отправка почты с вложениями (предусмотреть возможность массовой рассылки);
   5. возможность сохранения писем на диск.
4. Программно реализовать разработанные алгоритмы.

# Сравнительный анализ сетевых протоколов

## SMTP

SMTP (Simple Mail Transfer Protocol) — это сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP, причем передача должна быть обязательно инициирована самой передающей системой.

В Интернете для доставки электронной почты машина-источник устанавливает TCP-соединение с портом 25 машины приемника. Этот порт прослушивается почтовым демоном, и их общение происходит с помощью протокола SMTP. Этот демон принимает входящие соединения и копирует сообщения из них в соответствующие почтовые ящики. Если письмо невозможно доставить, отправителю отправляется сообщение об ошибке.

Протокол SMTP представляет собой простой ASCII протокол. Установив ТСР-соединение с портом 25, передающая машина, выступающая в роли клиента, ждет запроса принимающей машины, работающей в режиме сервера. Сервер начинает диалог с того что посылает текстовую строку, содержащую его идентификатор и сообщающую о его готовности (или неготовности) к приему почты. Если сервер не готов, клиент разрывает соединение и продолжает попытку позднее.

Если сервер готов принимать почту, клиент объявляет, от кого поступила почта и кому она предназначается. Если получатель почты существует, сервер дает клиенту добро на пересылку сообщения. Затем клиент посылает сообщение. А сервер подтверждает его получение. Контрольные суммы не проверяются, так как протокол ТСР обеспечивает надежный байтовый поток. Если у отправителя есть еще почта, она также отправляется. После передачи всей почты в обоих направлениях соединение разрывается.

## POP

Post Office Protocol Version 3 — это сетевой протокол, используемый почтовым клиентом для получения сообщений электронной почты с сервера. Обычно используется в паре с протоколом SMTP.

Предыдущие версии протокола (POP, POP2) устарели. Альтернативным протоколом для сбора сообщений с почтового сервера является IMAP. По умолчанию использует TCP-порт 110. Существуют реализации POP3-серверов, поддерживающие TLS и SSL.

После установки соединения протокол РОР3 проходит три последовательных состояния:

1. Авторизация — клиент проходит процедуру аутентификации.
2. Транзакция — клиент получает информацию о состоянии почтового ящика, принимает и удаляет почту.
3. Обновление — сервер удаляет выбранные письма и закрывает соединение.

Несмотря на то, что протокол РОР3 действительно поддерживает возможность получения одного или нескольких писем и оставления их на сервере, большинство программ обработки электронной почты просто скачивают все письма и опустошают почтовый ящик на сервере.

## IMAP

Пользователю, имеющему одну учетную запись у одного провайдера и всегда соединяющегося с провайдером с одной и той же машины, вполне достаточно протокола РОР3. Этот протокол используется повсеместно благодаря его простоте и надежности. Но у многих пользователей есть одна учетная запись в учебном заведении или на работе, но они хотят иметь к ней доступ и из дома, и с места работы (учебы), и во время командировки, т. е. из разных мест. Хотя протокол РОР3 и позволяет разрешить такую ситуацию, проблема в том, что при таком использовании электронной почты вся корреспонденция пользователя очень быстро распространится по случайным машинам, с которых он получал доступ в Интернет, и некоторые из этих компьютеров могут вообще не принадлежать пользователю.

Это неудобство привело к созданию альтернативного протокола для получения почты — IMAP.

IMAP (Internet Message Access Protocol) — интернет-протокол прикладного уровня для доступа к электронной почте.

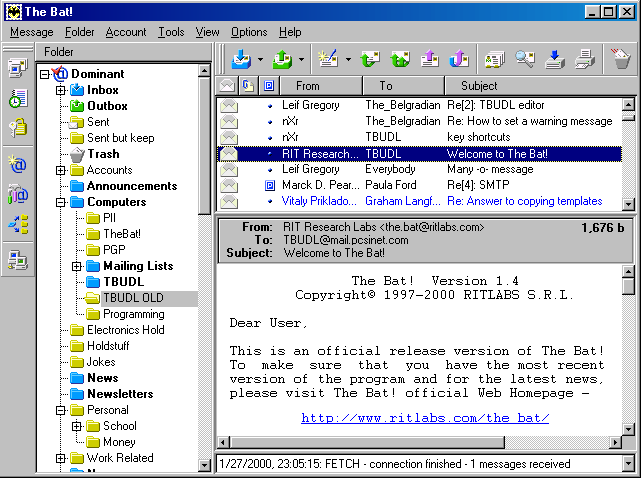
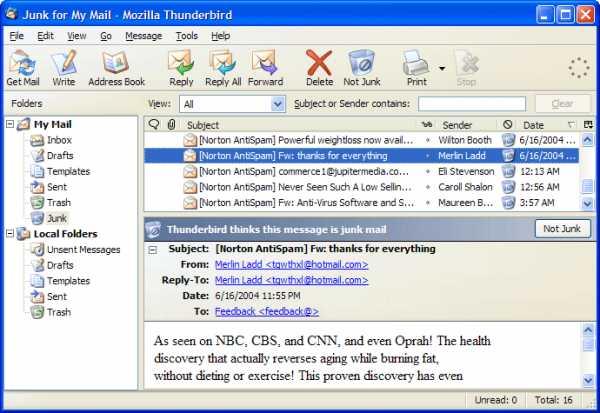
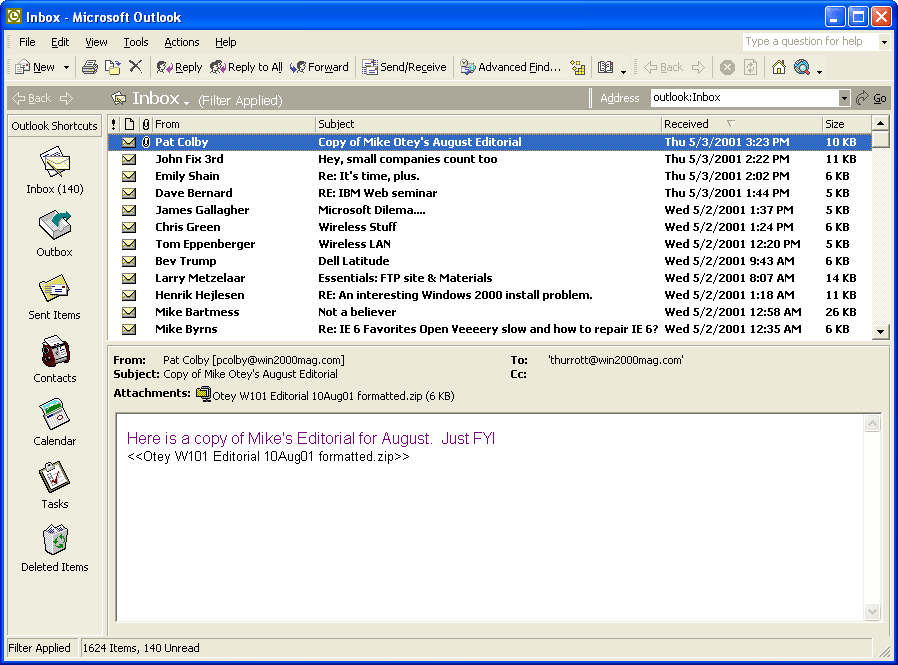
IMAP предоставляет пользователю богатые возможности для работы с почтовыми ящиками, находящимися на центральном сервере. Почтовая программа, использующая этот протокол, получает доступ к хранилищу корреспонденции на сервере так, как будто эта корреспонденция расположена на компьютере получателя. Электронными письмами можно манипулировать с компьютера пользователя (клиента) без необходимости постоянной пересылки с сервера и обратно файлов с полным содержанием писем.

IMAP был разработан для замены более простого протокола POP3 и имеет следующие преимущества по сравнению с последним:

1. Письма хранятся на сервере, а не на клиенте. Возможен доступ к одному и тому же почтовому ящику с разных клиентов. Поддерживается также одновременный доступ нескольких клиентов. В протоколе есть механизмы, с помощью которых клиент может быть проинформирован об изменениях, сделанных другими клиентами.
2. Поддержка нескольких почтовых ящиков (или папок). Клиент может создавать, удалять и переименовывать почтовые ящики на сервере, а также перемещать письма из одного почтового ящика в другой.
3. Возможно создание общих папок, к которым могут иметь доступ несколько пользователей.
4. Информация о состоянии писем хранится на сервере и доступна всем клиентам. Письма могут быть помечены как прочитанные, важные и т. п.
5. Поддержка поиска на сервере. Нет необходимости скачивать с сервера множество сообщений для того чтобы найти одно нужное.
6. Поддержка онлайн-работы. Клиент может поддерживать с сервером постоянное соединение, при этом сервер в реальном времени информирует клиента об изменениях в почтовых ящиках, в том числе о новых письмах.
7. Предусмотрен механизм расширения возможностей протокола.

Текущая версия протокола имеет обозначение IMAP4rev1 (IMAP, версия 4, ревизия 1). Протокол поддерживает передачу пароля пользователя в зашифрованном виде. Кроме того, IMAP-трафик можно зашифровать с помощью SSL.

Стандартные приложения, реализующие IMAP — это, к примеру, Microsoft Outlook, Mozilla Thunderbird и The Bat!



Последняя версия протокола IMAP основывается на спецификации RFC 3501 (<https://tools.ietf.org/html/rfc3501>).

# Детальный разбор протокола IMAP

## Взаимодействие клиента и сервера

Соединение IMAP 4.1 подразумевает установление связи между клиентом и сервером. Клиент посылает серверу команды, сервер клиенту — данные и уведомления о статусе выполнения запроса. Все сообщения, как клиента, так и сервера имеют форму строк, завершающихся специальной последовательностью.

Любая процедура начинается с команды клиента. Любая команда клиента начинается с префикса-идентификатора (обычно короткая буквенно-цифровая строка, например, A0001, A0002 и т. д.), называемого меткой (tag). Для каждой команды клиент генерирует свою метку.

Возможны два случая, когда строка, отправленная клиентом, не представляет собой законченную команду. В первом — аргумент команды снабжается кодом, определяющим число октетов в строке. Во втором — аргументы команды требуют отклика со стороны сервера. В обоих случаях сервер посылает запрос продолжения команды, начинающийся с символа +.

Клиент должен завершить отправку одной команды, прежде чем отправить другую.

Протокольный приемник сервера читает строку команды, пришедшей от клиента, осуществляет её разбор, выделяет параметры и передает серверу данные. По завершении команды сервер посылает отклик.

Данные, передаваемые сервером клиенту, а также статусные отклики, которые не указывают на завершение выполнения команды, имеют префикс \* и называются непомеченными откликами.

Данные могут быть отправлены сервером в ответ на команду клиента или по собственной инициативе. Формат данных не зависит от причины отправки.

Отклик указывает на удачное/неудачное выполнение операции. Он использует ту же метку, что и команда клиента, запустившая процедуру. Таким образом, если осуществляется более чем одна команда, метка сервера указывает на команду, вызвавшую данный отклик. Имеется три вида отклика завершения сервера: OK (успешное выполнение), NO (неудача), BAD (протокольная ошибка, например, не узнана команда или зафиксирована синтаксическая ошибка).

Протокольный приемник клиента IMAP 4.1 читает строку отклика от сервера и предпринимает действия в соответствии с первым символом \* или +.

Клиент должен быть готов принять любой отклик сервера в любое время. Данные сервера должны быть записаны так, чтобы клиент мог их непосредственно использовать, не посылая серверу уточняющих запросов.

## Команды протокола IMAP

* LOGIN  
  Позволяет клиенту при регистрации на сервере IMAP использовать идентификатор пользователя и пароль в обычном текстовом виде. Это не самый лучший метод, но иногда это единственная возможность подключиться к серверу.
* AUTHENTICATE   
  Позволяет клиенту использовать при регистрации на сервере IMAP альтернативные методы проверки подлинности. Индивидуальная проверка подлинности пользователей не является обязательной и поддерживается не всеми серверами IMAP. К тому же реализации такой проверки могут различаться в зависимости от сервера. Когда клиент выдает команду AUTHENTICATE, сервер отвечает на неё строкой вызова в кодировке base64. Далее клиент должен отправить ответ на вызов сервера о проверке подлинности, также закодированный base64. Если на сервере не поддерживается метод проверки подлинности, предложенный клиентом, он включает в свой ответ слово NO. После этого клиент должен продолжить переговоры по согласованию метода проверки подлинности. Если все попытки определить метод проверки подлинности потерпели неудачу, то клиент предпринимает попытку зарегистрироваться на сервере посредством команды LOGIN.
* CLOSE   
  Закрывает почтовый ящик. Когда почтовый ящик закрыт, то все сообщения, помеченные флагом \deleted, физически удаляются из него. Не имеет параметров.
* LOGOUT   
  Завершает сеанс для текущего идентификатора пользователя и закрывает все открытые почтовые ящики. Если какие-либо сообщения были помечены флагом \deleted, то с помощью этой команды они будут физически удалены из почтового ящика.
* CREATE   
  Создаёт новый почтовый ящик. Имя и местоположение новых почтовых ящиков определяются в соответствии с общими спецификациями сервера.
* DELETE   
  Применяется к почтовым ящикам. Сервер IMAP при получении этой команды попытается удалить почтовый ящик с именем, указанным в качестве аргумента команды. Сообщения удаляются вместе с ящиками и восстановлению не подлежат.
* RENAME   
  Изменяет имя почтового ящика. Эта команда имеет два параметра — имя почтового ящика, который требуется переименовать, и новое имя почтового ящика.
* SUBSCRIBE   
  Добавляет почтовый ящик в список активных ящиков клиента. В этой команде используется только один параметр — имя почтового ящика, который нужно внести в список. Почтовый ящик не обязательно должен существовать, чтобы его можно было добавить в список активных ящиков — это позволяет добавлять в список активных ящиков ящики, которые ещё не созданы, или удалять их, если они пусты.
* UNSUBSCRIBE   
  Удаляет почтовые ящики из списка активных. В ней так же используется один параметр — имя почтового ящика, который удаляется из списка активных ящиков клиента. При этом сам по себе почтовый ящик не удаляется.
* LIST   
  Получить список всех почтовых ящиков клиента; имеет два параметра.
* LSUB   
  В отличие от команды LIST используется для получения списка ящиков, активизированных командой SUBSCRIBE. Параметры — такие же, как у LIST.
* STATUS   
  Формирует запрос о текущем состоянии почтового ящика. Первым параметром для этой команды является имя почтового ящика, к которому она применяется. Второй параметр — это список критериев, по которым клиент хочет получить информацию. Команда STATUS может использоваться для получения информации о состоянии почтового ящика без его открытия с помощью команд SELECT или EXAMINE.  
  Пользователь может получить информацию по критериям:
  + MESSAGES — общее число сообщений в почтовом ящике;
  + RECENT — число сообщений с флагом \recent;
  + UIDNEXT — идентификатор UID, который будет назначен новому сообщению;
  + UIDVALIDITY — уникальный идентификатор почтового ящика;
  + UNSEEN — число сообщений без флага \seen.
* APPEND   
  Добавляет сообщение в конец указанного почтового ящика. В качестве аргументов указываются имя ящика, флаги сообщения (не обязательно), метка времени (не обязательно) и само сообщение — заголовок и тело.

Имеются следующие флаги сообщений:

* + \Seen — прочитано;
  + \Answered — написан ответ;
  + \Flagged — срочное;
  + \Deleted — помечено для удаления;
  + \Draft — черновик;
  + \Recent — новое сообщение, оно поступило в почтовый ящик после окончания прошлого сеанса.

Если в команде указаны флаги, то они устанавливаются для добавляемого сообщения. В любом случае для сообщения устанавливается флаг \Recent.

Если в команде задана метка времени, то это время будет установлено в качестве времени создания сообщения, в противном случае за время создания принимается текущее время.

Поскольку сообщение состоит не из одной строки, используются литералы.

* CHECK  
  Устанавливает контрольную точку в почтовом ящике. Любые операции, такие, например, как запись данных из памяти сервера на его жёсткий диск, должны выполняться при соответствующем состоянии почтового ящика. Именно для проверки целостности почтового ящика после дисковых и других подобных им операций и применяется команда CHECK. Эта команда используется без параметров.
* EXPUNGE   
  Удаляет из почтового ящика все сообщения, помеченные флагом \deleted, при этом почтовый ящик не закрывается. Ответ сервера на команду EXPUNGE представляет собой отчёт о новом состоянии почтового ящика.
* SEARCH   
  Поиск сообщений по критериям в активном почтовом ящике с последующим отображением результатов в виде номера сообщения.

Возможен поиск сообщений, в теле которых имеется определённая текстовая строка, или имеющих определённый флаг, или полученных до определённой даты и т. д.

* FETCH   
  Получить текст почтового сообщения. Команда применяется только для отображения сообщений. В отличие от POP3, клиент IMAP не сохраняет копию сообщения на клиентском ПК.
* STORE   
  Изменяет информацию о сообщении.
* COPY   
  Копирует сообщения из одного почтового ящика в другой.
* UID   
  Используется в связке с командами FETCH, COPY, STORE или SEARCH. С её помощью в этих командах можно использовать реальные идентификационные номера UID вместо последовательности чисел из диапазона номеров сообщений.
* CAPABILITY   
  Запрос у сервера IMAP информации о его возможностях.
* NOOP   
  Команда ничего не делает. Она может применяться для поддержки активности во время сеанса для того, чтобы сеанс не прекратился по таймеру интервала ожидания. Ответ сервера на команду NOOP всегда должен быть положительным. Так как сервер часто в ответе возвращает состояние выполнения той или иной команды, то NOOP вполне можно использовать как триггер для периодического запроса о состоянии сервера.

## Состояния сервера IMAP

Сервер IMAP 4.1 находится в одном из четырёх состояний.

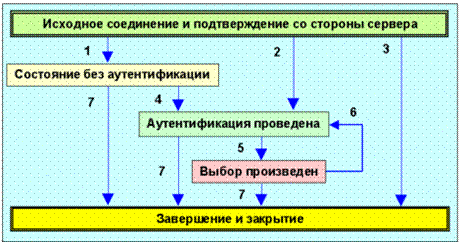
Большинство команд можно использовать только в определенных состояниях.

В состоянии *без аутентификации* клиент должен предоставить имя и пароль, прежде чем ему станет доступно большинство команд. Переход в это состояние производится при установлении соединения без предварительной аутентификации.

В состоянии *аутентификации* клиент идентифицирован и должен выбрать почтовый ящик, после чего ему станут доступны команды для работы с сообщениями. Переход в это состояние происходит при установлении соединение с предварительной аутентификацией, когда выданы все необходимые идентификационные данные или при ошибочном выборе почтового ящика.

В состояние *выбора* система попадает, когда успешно осуществлен выбор почтового ящика.

В состояние *выхода* система попадает при прерывании соединения в результате запроса клиента или вследствие независимого решения сервера.



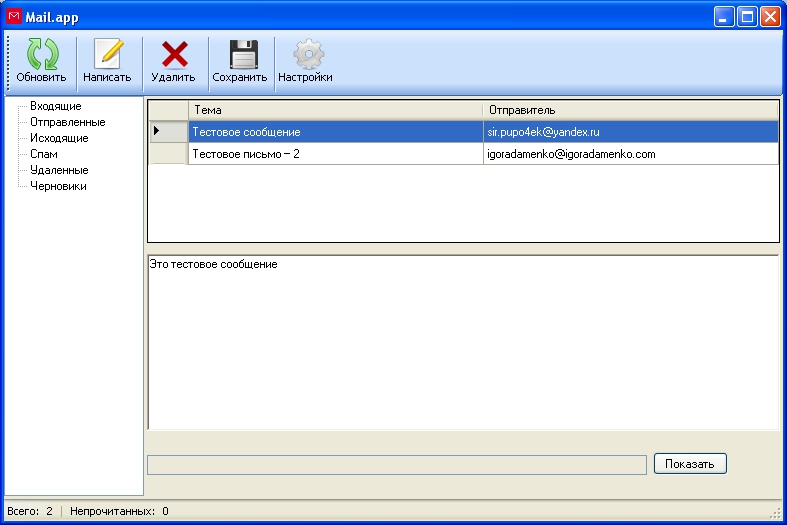
1. Соединение без предварительной аутентификации.
2. Соединение с предварительной аутентификацией.
3. Соединение отвергнуто.
4. Успешное завершение команды LOGIN или AUTHENTICATE.
5. Успешное завершение команды SELECT или EXAMINE.
6. Выполнение команды CLOSE или неудачная команда SELECT или EXAMINE.
7. Выполнение команды LOGOUT, закрытие сервера, или прерывание соединения.

# Разработка пользовательского интерфейса

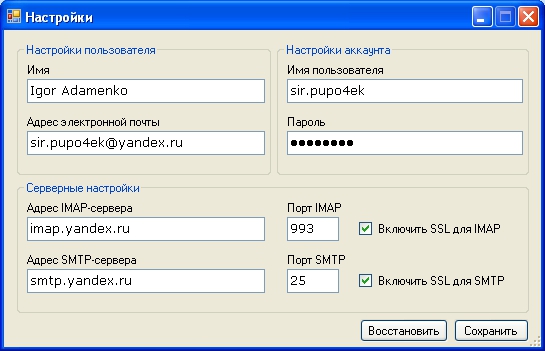
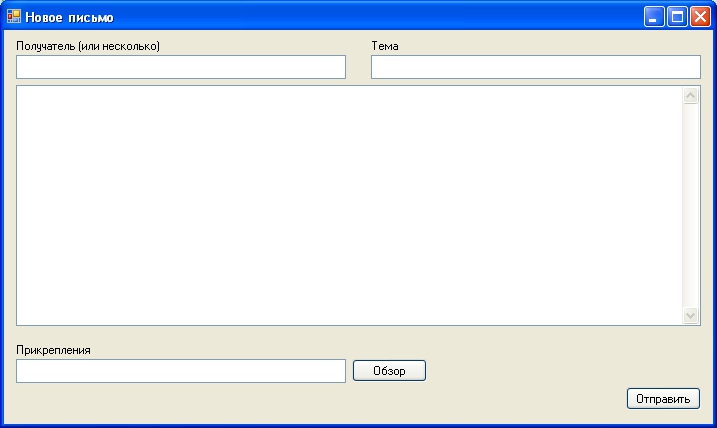
Для разработки пользовательского интерфейса использовалась среда Visual Studio 2010 Professional, предоставляющая возможность создавать приложения типа Windows Forms. Тип этих приложений хорош тем, что его компоненты выглядят так же, как и стандартные приложения Windows XP.

Было разработано три различных окна.

Первое — главное окно. В нём располагаются все основные компоненты программы: панель управления, статус-бар, списки папок и сообщений и тело читаемого сообщения.

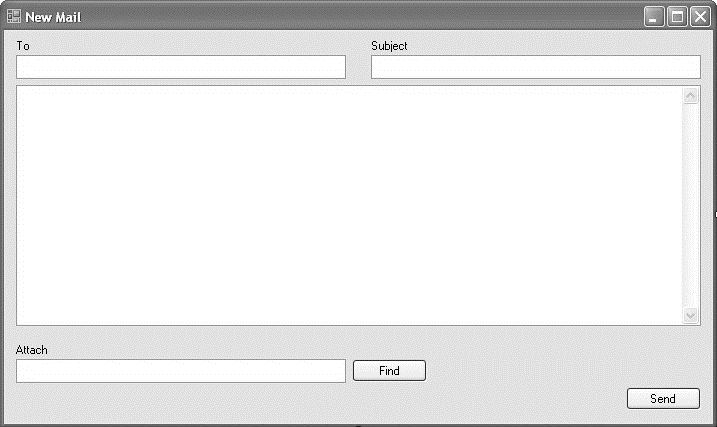


Из этого окна пользователь может перейти в два других: экран написания нового сообщения и экран настроек.



На экране настроек предусмотрен элемент управления, возвращающий введённые данные к начальным, на случай, если пользователь настроил что-то не так, как ему хотелось бы.

Также с помощью встроенных средств Windows Forms предусмотрена локализация приложения на любой язык. Для этого достаточно добавить соответствующий файл ресурсов в папку с программой. Вот так, например, выглядит интерфейс одного из окон программы на английском языке:



# Разработка основных алгоритмов

## Аутентификация

Перед началом работы с сервером IMAP необходимо доказать ему, что данный клиент имеет право работать с запрашиваемыми данными, для этого совершается процедура аутентификации.

Для этого клиент посылает серверу запрос типа LOGIN с двумя параметрами: имя пользователя и пароль. Если данные введены верно, то сервер отвечает OK, иначе — NO.

## Получение каталогов с сообщениями

После прохождения аутентификации, необходимо получить список всех возможных каталогов сообщений на сервере. Для этого клиент посылает серверу команду LIST, с двумя параметрами, первый из которых пустой, а второй — астериск (звёздочка «\*»).

Сервер в ответ последовательно отдаёт хранящиеся у него каталоги сообщений, завершая передачу ответом со статусом OK.

## Выбор рабочего каталога

Перед тем, как начать работать конкретно с письмами, пользователь должен выбрать, с каким именно каталогом он хочет работать. Для этого клиент посылает запрос SELECT с одним параметром — именем каталога.

Если команда выполнена успешна, то все последующие операции над сообщениями будут производиться в этом каталоге, до того момента, пока пользователь его не изменит.

## Получение списка сообщений

Для получения списка сообщений клиент должен отправить серверу команду UID FETCH с двумя параметрами. Первый — диапазон выбираемых сообщений (например: 1:10 — с 1 по 10, 3:5 — с 3 по 5, 1:\* — все), а второй — то, что необходимо выбрать, например FLAGS UID означает, что клиент хочет получить флаги всех сообщений и их уникальные идентификаторы.

## Получение отдельного сообщения

Для выгрузки с сервера отдельного сообщения клиент должен так же использовать команду UID FETCH, только в этот раз первым параметром указать UID выбираемого сообщения, а вторым — какую часть сообщения необходимо получить (например, body[]).

В зависимости от ситуации сервер может присылать данные кодом base64, или же в кодировке UTF7. Соответственно, клиент должен уметь их расшифровывать. Для этого в рамках этой курсовой работы были написаны шифраторы и дешифраторы этих двух ситуаций.

# Структура программы

Программа состоит из следующих классов:

* MainForm — основной класс для работы с главной формой программы;
* Tools.NewMail — класс для работы с формой отправки нового сообщения;
* Tools.Settings — класс для работы с формой настроек;
* Utils.Base64Converter — класс для кодирования и декодирования данных в base64;
* Utils.UTF7Converter — класс для кодирования и декодирования данных в UTF7;
* IMAP.Commands — класс, использующийся для описания всех шаблонов команд протокола IMAP;
* IMAP.IMAPClient — класс, реализующий основную логику работы программы с протоколом IMAP. Содержит такие методы как:
  + Authenticate — осуществляет аутентификацию на сервере;
  + GetFolders — возвращает информацию о каталогах сообщений;
  + SelectFolder — выбирает папку для дальнейшей работы;
  + GetMessageCount — возвращает общее количество сообщений в текущем каталоге;
  + GetUnseenMessageCount — возвращает количество непрочитанных сообщений в текущем каталоге;
  + GetListMessages — возвращает список сообщений в текущем каталоге;
  + GetMessage — возвращает тело выбранного сообщения и вложения;
  + Delete — удаляет выбранное сообщение;
  + SendCommand — общий метод для отправки команд на сервер;
  + GetResponse — общий метод для получения ответа от сервера.
* SMTP.SMTPClient — класс, реализующий основную логику работы программы с протоколом SMTP. Содержит такие методы как:
  + Authenticate — осуществляет аутентификацию на сервере;
  + Send — отправляет сообщение;
  + Close — закрывает созданное ранее соединение.

# Заключение

Как видно из проделанной работы, протокол IMAP (вместе с SMTP) отлично подходит для реализации почтового клиента.

Выполняя курсовую работу, я закрепил знания и навыки, полученные в рамках курса «Сети ЭВМ и телекоммуникации». Также я получил практический опыт реализации протоколов IMAP и SMTP на примере разработки почтового клиента, используя язык программирования высокого уровня.

# Список литературы

1. «Сети электронно-вычислительных машин и телекоммуникации: методические указания к выполнению лабораторных работ» / сост. Е. А. Федотов, А. И. Гарибов. — Белгород: Изд-во БГТУ, 2014 г. — 77 с.
2. «Введение в программирование на языке Visual C#», С. Р. Гуриков. — Форум, Инфра-М, 2013 г. — 448 с.
3. «IMAP — Википедия» [Электронный ресурс] / <https://ru.wikipedia.org/wiki/IMAP> — Режим доступа: свободный.
4. Stack Overflow [Электронный ресурс] / <http://stackoverflow.com/> — Режим доступа: свободный.
5. Understanding the Insides of the SMTP Mail Protocol: Part 1 – CodeProject [Электронный ресурс] / <http://www.codeproject.com/Articles/399207/Understanding-the-insides-of-the-SMTP-Mail-protoco> — Режим доступа: свободный.
6. Understanding the insides of the IMAP mail protocol: Part 3 – CodeProject [Электронный ресурс] / <http://www.codeproject.com/Articles/411018/Understanding-the-insides-of-the-IMAP-mail-protoco> — Режим доступа: свободный.
7. RFC 3501 - INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1 [Электронный ресурс] / <https://tools.ietf.org/html/rfc3501> — Режим доступа: свободный.

# Приложение 1. Код программы

## MainForm.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Windows.Forms;
4. **using** Course.IMAP;
5. **using** Course.Utils;
6. **using** Course.Mail;
7. **using** System.Globalization;
8. **using** System.Diagnostics;
9. **using** System.IO;
11. **namespace** Course
12. {
13. **public** partial **class** MainForm : Form
14. {
15. Logger \_logger = Logger.Instance;
16. IMAPClient imap;
17. Properties.Settings settings = Properties.Settings.Default;
18. **string** folderPath = @"C:\Course\Temp";
20. **public** MainForm()
21. {
22. InitializeComponent();
24. imap = **new** IMAPClient(settings.IMAPServer, settings.IMAPPort,  
     settings.IMAPSSL);
25. imap.Authenticate(settings.Username, settings.Password);
27. UpdateFolders();
28. }
30. **private** **void** UpdateFolders()
31. {
32. var folders = imap.GetFolders();
33. foldersTreeView.BeginUpdate();
34. foldersTreeView.Nodes.Clear();
36. var inbox = (**string**)Properties.Resources.ResourceManager.GetObject("Inbox",  
     CultureInfo.CurrentCulture);
38. **if** (folders.Contains(inbox))
39. {
40. foldersTreeView.Nodes.Add(inbox);
41. folders.Remove(inbox);
42. }
44. **foreach** (var folder **in** folders)
45. {
46. foldersTreeView.Nodes.Add(folder);
47. }
49. foldersTreeView.EndUpdate();
50. }
52. **private** **void** ReloadMessages()
53. {
54. imap.SelectFolder(foldersTreeView.SelectedNode.Text);
56. totalNumberToolStripStatusLabel.Text = imap.GetMessageCount().ToString();
57. unseenNumberToolStripStatusLabel.Text =   
     imap.GetUnseenMessageCount().ToString();
59. var list = imap.GetListMessages();
60. var messages = **new** List<**string**>();
62. **foreach** (var item **in** list)
63. {
64. messages.Add(imap.GetMessage(item.Value));
65. }
67. maillistDataGridView.Rows.Clear();
69. **foreach** (var item **in** messages)
70. {
71. maillistDataGridView.Rows.Add(item.UId.ToString(), item.Subject,  
     item.From.Address);
73. System.IO.Directory.CreateDirectory(folderPath);
75. **foreach** (var file **in** item.Attachments)
76. {
77. file.Download();
78. file.Save(folderPath);
79. }
80. }
82. }
84. #region events
86. **private** **void** foldersTreeView\_AfterSelect(**object** sender, TreeViewEventArgs e)
87. {
88. ReloadMessages();
89. }
91. **private** **void** newMailToolStripButton\_Click(**object** sender, EventArgs e)
92. {
93. var newmail = **new** Course.Tools.NewMail();
94. newmail.Show();
95. }
97. **private** **void** settingsToolStripButton\_Click(**object** sender, EventArgs e)
98. {
99. var settings = **new** Course.Tools.Settings();
100. settings.ShowDialog();
101. }
103. **private** **void** getMailToolStripButton\_Click(**object** sender, EventArgs e)
104. {
105. ReloadMessages();
106. }
108. **private** **void** maillistDataGridView\_SelectionChanged(**object** sender,  
      EventArgs e)
109. {
110. **if** (maillistDataGridView.SelectedRows.Count > 0)
111. {
112. var mail = messages[maillistDataGridView.SelectedRows[0].Index];
114. mailRichTextBox.Text = mail.Body.Text
115. .Split(**new** **string**[] { "IMAP" },  
      StringSplitOptions.None)[0]  
      .TrimEnd(' ', ')', '\r', '\n');
116. attachmentsTextBox.Text = "";
118. **foreach** (var item **in** mail.Attachments)
119. {
120. attachmentsTextBox.Text += item.FileName + " (" +  
      (item.FileSize / 1024).ToString() + " Kb), ";
121. }
123. **if** (attachmentsTextBox.Text.Length > 1)
124. {
125. attachmentsTextBox.Text = attachmentsTextBox.Text  
      .Substring(0, attachmentsTextBox.Text.Length - 2);
126. }
127. }
128. }
130. **private** **void** mailDeleteToolStripButton\_Click(**object** sender, EventArgs e)
131. {
132. imap.Delete(Convert.ToInt32(maillistDataGridView  
      .SelectedRows[0].Cells["UID"].Value.ToString()));
134. messages[maillistDataGridView.SelectedRows[0].Index].Remove();
135. }
137. **private** **void** saveToButton\_Click(**object** sender, EventArgs e)
138. {
139. Process.Start(folderPath);
140. }
142. **private** **void** mailSaveToolStripButton\_Click(**object** sender, EventArgs e)
143. {
144. var mail = messages[maillistDataGridView.SelectedRows[0].Index];
145. var text = (**string**)Properties.Resources.ResourceManager  
      .GetObject("Subject", CultureInfo.CurrentCulture) +
146. mail.Subject + Environment.NewLine +  
      (**string**)Properties.Resources.ResourceManager  
      .GetObject("From", CultureInfo.CurrentCulture) +
147. mail.From.Address + Environment.NewLine +  
      Environment.NewLine + mailRichTextBox.Text;
149. FileStream fcreate = File.Open(folderPath + "\\message.txt",  
      FileMode.Create);
150. StreamWriter sw = **new** StreamWriter(fcreate);
151. sw.Write(text);
152. sw.Flush();
153. fcreate.Close();
155. **string** newfolder = folderPath + "\\message-" +  
      mail.From.Address + mail.UId;

158. **if** (System.IO.Directory.Exists(newfolder))
159. {
160. System.IO.Directory.Move(newfolder, newfolder +  
      DateTime.Now.Millisecond);
161. }
163. System.IO.Directory.CreateDirectory(newfolder);
164. System.IO.File.Move(folderPath + "\\message.txt",  
      newfolder + "\\message.txt");
166. **foreach** (var item **in** mail.Attachments)
167. {
168. System.IO.File.Copy(folderPath + '\\' + item.FileName,  
      newfolder + '\\' + item.FileName);
169. }
171. Process.Start(newfolder);
172. }
174. #endregion
175. }
176. }

## Tools.NewMail.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Linq;
5. **using** System.Windows.Forms;
6. **using** Course.SMTP;
8. **namespace** Course.Tools
9. {
10. **public** partial **class** NewMail : Form
11. {
12. SMTPClient smtp;
13. Properties.Settings settings = Properties.Settings.Default;
15. **public** NewMail()
16. {
17. InitializeComponent();
19. smtp = **new** SMTPClient(settings.SMTPServer, settings.SMTPPort,  
     settings.SMTPSSL);
20. smtp.Authenticate(settings.Username, settings.Password);
21. }
23. **public** NewMail(List<**string**> to, **string** subject)
24. {
25. InitializeComponent();
27. toTextBox.Text = String.Join(",", to.ToArray());
28. subjectTextBox.Text = subject;
30. smtp = **new** SMTPClient(settings.SMTPServer, settings.SMTPPort,  
     settings.SMTPSSL);
31. smtp.Authenticate(settings.Username, settings.Password);
32. }
34. **private** **void** SendMail()
35. {
36. var to = toTextBox.Text.Split(',').ToList<**string**>();
37. var files = filesTextBox.Text.Split(',').ToList<**string**>();
39. smtp.Send(settings.Email, to, subjectTextBox.Text, messageTextBox.Text, files);
40. **this**.Close();
41. }
43. **private** **void** FindFiles()
44. {
45. OpenFileDialog dialog = **new** OpenFileDialog();
46. dialog.FileOk += **new** CancelEventHandler(dialog\_FileOk);
47. dialog.Multiselect = **true**;
48. dialog.ShowDialog();
49. }
51. #region manual events
53. **private** **void** dialog\_FileOk(**object** sender, CancelEventArgs e)
54. {
55. **if** (!e.Cancel)
56. {
57. filesTextBox.Text = String.Join(",", ((OpenFileDialog)sender).FileNames);
58. }
59. }
61. #endregion
63. #region events
65. **private** **void** NewMail\_FormClosing(**object** sender, FormClosingEventArgs e)
66. {
67. smtp.Close();
68. }
70. **private** **void** sendButton\_Click(**object** sender, EventArgs e)
71. {
72. SendMail();
73. }
75. **private** **void** attachButton\_Click(**object** sender, EventArgs e)
76. {
77. FindFiles();
78. }
80. #endregion
81. }
82. }

## Tools.Settings.cs

1. **using** System;
2. **using** System.Text;
3. **using** System.Windows.Forms;
4. **using** Course.Utils;
6. **namespace** Course.Tools
7. {
8. **public** partial **class** Settings : Form
9. {
10. **public** Settings()
11. {
12. InitializeComponent();
13. }
15. #region events
17. **private** **void** saveButton\_Click(**object** sender, EventArgs e)
18. {
19. Properties.Settings.Default.Save();
20. **this**.Close();
21. }
23. **private** **void** restoreButton\_Click(**object** sender, EventArgs e)
24. {
25. Properties.Settings.Default.Reload();
26. }
28. #endregion
29. }
30. }

## Utils.Base64Converter.cs

1. **using** System.Text;
2. **using** System.IO;
3. **using** System.Security.Cryptography;
5. **namespace** Course.Utils
6. {
7. **public** **static** **class** Base64Converter
8. {
9. **public** **static** **string** Encode(**byte**[] data)
10. {
11. var builder = **new** StringBuilder();
12. **using** (var writer = **new** StringWriter(builder))
13. {
14. **using** (var transformation = **new** ToBase64Transform())
15. {
16. var bufferedOutputBytes = **new** **byte**[transformation.OutputBlockSize];
17. **int** i = 0;
18. **int** inputBlockSize = transformation.InputBlockSize;
20. **while** (data.Length - i > inputBlockSize)
21. {
22. transformation.TransformBlock(data, i, data.Length - i,  
     bufferedOutputBytes, 0);
23. i += inputBlockSize;
24. writer.Write(Encoding.UTF8.GetString(bufferedOutputBytes, 0,  
     bufferedOutputBytes.Length));
25. }
27. bufferedOutputBytes = transformation.TransformFinalBlock(data, i,  
     data.Length - i);
28. writer.Write(Encoding.UTF8.GetString(bufferedOutputBytes, 0,  
     bufferedOutputBytes.Length));
29. transformation.Clear();
30. }
32. writer.Close();
33. }
35. **return** builder.ToString();
36. }
38. **public** **static** **byte**[] Decode(**string** s)
39. {
40. **byte**[] bytes;
42. **using** (var writer = **new** MemoryStream())
43. {
44. **byte**[] inputBytes = Encoding.UTF8.GetBytes(s);
46. **using** (var transformation =  
     **new** FromBase64Transform(FromBase64TransformMode.IgnoreWhiteSpaces))
47. {
48. var bufferedOutputBytes = **new** **byte**[transformation.OutputBlockSize];
49. **int** i = 0;
51. **while** (inputBytes.Length - i > 4)
52. {
53. transformation.TransformBlock(inputBytes, i, 4,  
     bufferedOutputBytes, 0);
54. i += 4;
55. writer.Write(bufferedOutputBytes, 0,  
     transformation.OutputBlockSize);
56. }
58. bufferedOutputBytes = transformation.TransformFinalBlock(inputBytes, i,  
     inputBytes.Length - i);
59. writer.Write(bufferedOutputBytes, 0, bufferedOutputBytes.Length);
60. transformation.Clear();
61. }
63. writer.Position = 0;
64. bytes = writer.ToArray();
65. writer.Close();
66. }
68. **return** bytes;
69. }
70. }
71. }

## Utils.UTF7Converter.cs

1. **using** System.Collections.Generic;
2. **using** System.Linq;
3. **using** System.Text;
4. **using** System.Text.RegularExpressions;
6. **namespace** Course.Utils
7. {
8. **public** **class** UTF7Converter
9. {
10. **public** **static** **string** Encode(**string** value)
11. {
12. **if** (**string**.IsNullOrEmpty(value))
13. **return** **string**.Empty;
15. value = value.Replace("&", "&-");
17. var rex = **new** Regex("[^ -~]\*");
18. var encoding = **new** UnicodeEncoding(**true**, **false**);
20. IEnumerable<Match> toEncode =
21. rex.Matches(value).Cast<Match>()  
     .Where(\_ => !**string**.IsNullOrEmpty(\_.Value));
23. **foreach** (Match match **in** toEncode)
24. {
25. **string** b64 = "&" + Base64Converter.Encode(encoding.GetBytes(match.Value))  
     .TrimEnd('=').Replace("/", ",") +
26. "-";
27. value = value.Replace(match.Value, b64);
28. }
30. **return** value;
31. }
33. **public** **static** **string** Decode(**string** name)
34. {
35. **string** result = name.Replace("&-", "&");
36. var encoding = **new** UnicodeEncoding(**true**, **false**);
37. var rex = **new** Regex(@"&[\w|,|\+]\*-");
39. MatchCollection matches = rex.Matches(name);
41. **foreach** (Match m **in** matches)
42. {
43. **string** v = m.Value.TrimStart('&').TrimEnd('-').Replace(",", "/");
44. v += v.Length % 4 == 2 ? "==" : (v.Length % 4 == 3 ? "=" : "");
45. var data = Base64Converter.Decode(v);
46. result = result.Replace(m.Value, encoding.GetString(data, 0, data.Length)  
     .Replace("\0", ""));
47. }
49. **return** result;
50. }
51. }
52. }

## IMAP.Commands.cs

1. **namespace** Course.IMAP
2. {
3. **public** **class** Commands
4. {
5. **public** **const** **string** Login = "LOGIN \"{0}\" \"{1}\"";
7. **public** **const** **string** Logout = "LOGOUT";
9. **public** **const** **string** List = "LIST \"{0}\" {1}";
11. **public** **const** **string** Delete = "DELETE \"{0}\"";
13. **public** **const** **string** Status = "STATUS \"{0}\" ({1})";
15. **public** **const** **string** Search = "UID SEARCH {0}";
17. **public** **const** **string** Fetch = "UID FETCH {0} ({1})";
19. **public** **const** **string** Select = "SELECT \"{0}\"";
21. **public** **const** **string** Noop = "NOOP";
22. }
23. }

## IMAP.IMAPClient.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Net.Sockets;
6. **using** System.IO;
7. **using** Course.Utils;
8. **using** System.Net.Security;
9. **using** System.Text.RegularExpressions;
10. **using** System.Globalization;
12. **namespace** Course.IMAP
13. {
14. **class** IMAPClient
15. {
16. **private** TcpClient \_client;
17. **private** StreamWriter \_writer;
18. **private** StreamReader \_reader;
19. **private** Logger \_logger = Logger.Instance;
20. **private** UInt64 \_prefix = 0;
21. **private** **string** \_selectedFolder;
23. **public** IMAPClient(**string** hostname, **int** port, **bool** ssl)
24. {
25. **try**
26. {
27. \_client = **new** TcpClient(hostname, port);
29. **if** (ssl)
30. {
31. var stream = **new** SslStream(\_client.GetStream());
32. stream.AuthenticateAsClient(hostname);
34. \_writer = **new** StreamWriter(stream);
35. \_reader = **new** StreamReader(stream);
36. }
37. **else**
38. {
39. var stream = \_client.GetStream();
41. \_writer = **new** StreamWriter(stream);
42. \_reader = **new** StreamReader(stream);
43. }
45. **string** greeting = \_reader.ReadLine();
46. }
47. **catch** (Exception e)
48. {
49. \_logger.AddError(e.Message);
50. }
51. }
53. **public** IMAPClient(**string** hostname, **string** sport, **bool** ssl)
54. : **this**(hostname, Convert.ToInt32(sport), ssl)
55. {
56. }
58. **public** **void** Authenticate(**string** username, **string** password)
59. {
60. **this**.SendCommand(Commands.Login, username, password);
61. **this**.GetResponse();
62. }
64. **public** List<**string**> GetFolders()
65. {
66. **this**.SendCommand(Commands.List, "", "\*");
67. **string** response = **this**.GetResponse();
69. **string**[] lines = response.Split(**new** **string**[] { Environment.NewLine },  
     StringSplitOptions.RemoveEmptyEntries);
70. List<**string**> folders = **new** List<**string**>();
72. **foreach** (**string** line **in** lines)
73. {
74. MatchCollection m = Regex.Matches(line, "\\\"(.\*?)\\\"|(INBOX)");
76. **if** (m.Count > 1)
77. {
78. **string** folderName = m[m.Count - 1].ToString().Trim(**new** **char**[] { '"' });
79. **if** (folderName == "INBOX")
80. {
81. folderName = (**string**)Properties.Resources.ResourceManager  
     .GetObject("Inbox", CultureInfo.CurrentCulture);
82. }
84. folders.Add(folderName);
85. }
86. }
88. **return** folders;
89. }
91. **public** **void** SelectFolder(**string** folderName)
92. {
93. **if** (folderName == (**string**)Properties.Resources.ResourceManager  
     .GetObject("Inbox", CultureInfo.CurrentCulture))
94. {
95. folderName = "INBOX";
96. }
98. \_selectedFolder = folderName;
99. **this**.SendCommand(Commands.Select, folderName);
101. **this**.GetResponse();
102. }
104. **public** **int** GetMessageCount()
105. {
106. **this**.SendCommand(Commands.Status, **this**.\_selectedFolder, "messages");
107. **string** response = **this**.GetResponse();
108. Match m = Regex.Match(response, "[0-9]\*[0-9]");
110. **return** Convert.ToInt32(m.ToString());
111. }
113. **public** **int** GetUnseenMessageCount()
114. {
115. **this**.SendCommand(Commands.Status, **this**.\_selectedFolder, "unseen");
116. **string** response = **this**.GetResponse();
117. Match m = Regex.Match(response, "[0-9]\*[0-9]");
119. **return** Convert.ToInt32(m.ToString());
120. }
122. **public** Dictionary<**int**, **string**> GetListMessages()
123. {
124. **this**.SendCommand(Commands.Fetch, "1:\*", "FLAGS UID");
125. **string** response = **this**.GetResponse();
127. **string**[] lines = response.Split(**new** **string**[] { Environment.NewLine },  
      StringSplitOptions.RemoveEmptyEntries);
128. var messages = **new** Dictionary<**int**, **string**>();
130. **foreach** (**string** line **in** lines)
131. {
132. MatchCollection m = Regex.Matches(line, "[0-9]\*[0-9]");
134. **if** (m.Count > 1)
135. {
136. **int** id = Convert.ToInt32(m[0].ToString()  
      .Trim(**new** **char**[] { '"' }));
137. **string** uid = m[1].ToString().Trim(**new** **char**[] { '"' });
139. messages.Add(id, uid);
140. }
141. }
143. **return** messages;
144. }
146. **public** **void** Delete(**int** id)
147. {
148. **this**.SendCommand(Commands.Delete, id.ToString());
149. **this**.GetResponse();
150. }
152. **public** **void** SendCommand(**string** command, **params** **string**[] list)
153. {
154. var cmd = UTF7Converter.Encode(String.Format(command, list));
155. \_writer.WriteLine("A" + (\_prefix++).ToString() + ' ' + cmd);
156. \_writer.Flush();
157. }
159. **private** **string** GetResponse()
160. {
161. **string** response = **string**.Empty;
163. **while** (**true**)
164. {
165. **string** line = UTF7Converter.Decode(\_reader.ReadLine());
166. **string**[] tags = line.Split(**new** **char**[] { ' ' });
167. response += line + Environment.NewLine;
168. **if** (tags[0].Substring(0, 1) == "A"
169. && tags[1].Trim() == "OK"
170. || tags[1].Trim() == "BAD"
171. || tags[1].Trim() == "NO")
172. {
173. **break**;
174. }
176. }
178. **return** response;
179. }
181. **public** **string** GetMessage(**string** uid)
182. {
183. **this**.SendCommand(Commands.Fetch, uid, "body[text]");
184. var message = **this**.\_GetMessage();
186. **return** message;
187. }
189. **private** **string** \_GetMessage()
190. {
191. **string** line = \_reader.ReadLine();
192. MatchCollection m = Regex.Matches(line, "\\{(.\*?)\\}");
194. **if** (m.Count > 0)
195. {
196. **int** length = Convert.ToInt32(m[0].ToString()  
      .Trim(**new** **char**[] { '{', '}' }));
198. **char**[] buffer = **new** **char**[length];
199. **int** read = (length < 128) ? length : 128;
200. **int** remaining = length;
201. **int** offset = 0;
202. **while** (**true**)
203. {
204. read = \_reader.Read(buffer, offset, read);
205. remaining -= read;
206. offset += read;
207. read = (remaining >= 128) ? 128 : remaining;
209. **if** (remaining == 0)
210. {
211. **break**;
212. }
213. }
215. **char**[] needful = buffer.Where(x => (x.ToString() != "\r" &&  
      x.ToString() != "\n")).ToArray();
216. **byte**[] data = Base64Converter.Decode(**new** String(needful));
218. **return** Encoding.UTF8.GetString(data).Replace("\0", "");
219. }
221. **this**.GetResponse();
223. **return** "";
224. }
225. }
226. }

## SMTP.SMTPClient.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Text;
4. **using** System.Net.Mail;
5. **using** System.Net;
6. **using** System.Net.Mime;
7. **using** Course.Utils;
9. **namespace** Course.SMTP
10. {
11. **class** SMTPClient
12. {
13. **private** SmtpClient \_client;
14. **private** Logger \_logger = Logger.Instance;
16. **public** SMTPClient(**string** hostname, **int** port, **bool** ssl)
17. {
18. \_client = **new** SmtpClient(hostname, port);
19. \_client.EnableSsl = ssl;
20. \_client.Timeout = 10000;
21. \_client.DeliveryMethod = SmtpDeliveryMethod.Network;
22. \_client.UseDefaultCredentials = **false**;
23. }
25. **public** SMTPClient(**string** hostname, **string** sport, **bool** ssl)
26. : **this**(hostname, Convert.ToInt32(sport), ssl)
27. {
28. }
30. **public** **void** Authenticate(**string** username, **string** password)
31. {
32. \_client.Credentials = **new** NetworkCredential(username, password);
33. }
35. **public** **void** Send(**string** from, List<**string**> to, **string** subject, **string** body,  
     List<**string**> files)
36. {
37. var message = **new** MailMessage();
38. message.BodyEncoding = Encoding.UTF8;
39. message.HeadersEncoding = Encoding.UTF8;
40. message.From = **new** MailAddress(from);
42. **foreach** (var item **in** to)
43. {
44. message.To.Add(item.Trim());
45. }
47. message.Subject = subject;
48. message.Body = body;
50. **if** (files.Count > 0)
51. {
52. **foreach** (var file **in** files)
53. {
54. **if** (file.Length > 0)
55. {
56. message.Attachments.Add(**new** Attachment(file,  
     MediaTypeNames.Application.Octet));
57. }
58. }
59. }
61. **try**
62. {
63. \_client.Send(message);
64. }
65. **catch** (Exception e)
66. {
67. \_logger.Add(e.Message);
68. }
69. }
71. **public** **void** Close()
72. {
73. \_client.Dispose();
74. }
75. }
76. }