

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ  
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и администрирования  
информационных систем

Разработка и реализация программного комплекса, обеспечивающего  
централизованный контроль движущихся объектов на основе спутниковых  
навигационных систем

Дипломная работа  
по специальности 351500 – Математическое обеспечение и  
администрирование информационных систем,  
специализация 351502 Информационные системы

Допущено к защите в ГАК  
Зав. кафедрой \_\_\_\_\_

*xx.05.2008*  
д. ф. – м.н., проф. Артемов М.А.

Студент \_\_\_\_\_

4 к. 9 гр.                      Алексеев О.В.

Руководитель \_\_\_\_\_

д. ф. – м.н., проф. Артемов М.А.

Воронеж 2008

## **Аннотация**

В работе рассмотрены основные принципы функционирования спутниковых приемников, предназначенных для определения местоположения на примере системы GPS NAVSTAR. Рассмотрены различные подходы к реализации централизованного контроля над удаленными объектами. Реализован программный комплекс, осуществляющий контроль над удаленными объектами.

## Содержание

Аннотация .....	2
Содержание .....	3
Введение .....	5
1. Постановка задачи .....	10
2. Анализ задачи .....	11
2.1. Анализ структуры программного комплекса и его функций .....	11
2.2. Принципы функционирования GPS-приемников .....	13
2.3. Протоколы обмена данными с мобильными GPS-приемниками .....	15
2.4. Протокол NMEA .....	16
2.5. Анализ функций серверной части программного комплекса .....	20
2.5.1. Структуры данных .....	20
2.5.2. Алгоритм вычисления расстояния между точками на земной поверхности .....	21
2.5.3. Функции элементов программного комплекса .....	22
3. Средства реализации .....	24
4. Требования к аппаратному и программному обеспечению .....	25
5. Интерфейс пользователя .....	26
5.1. Интерфейс клиентского приложения .....	26
5.2. Интерфейс серверного приложения .....	28
6. Реализация .....	31
6.1. Структура программного комплекса .....	31
6.2. Описание блоков программного комплекса .....	32
6.3. Структура базы данных .....	33
6.4. Протокол обмена данными между клиентом и сервером .....	34
6.5. Структура клиентской части .....	37
6.5.1. Схема работы клиентского приложения .....	37
6.5.2. Описание основных классов, процедур и функций .....	39
Заключение .....	45

Список литературы .....	46
Приложение 1. Описание таблиц базы данных.....	47
Приложение 2. Sql-скрипт создания базы данных .....	50
Приложение 3. Листинг клиентского приложения.....	54

## Введение

За последние десять лет в области высоких технологий наблюдается рост интереса к навигационной аппаратуре, функционирующей на основе спутниковых навигационных систем (СНС). Это связано со снижением цен на аппаратуру потребительского сегмента этих систем – навигационные приемники.

В настоящее время существуют две такие системы: разработанная американским NASA GPS (Global positioning system) NAVSTAR (в дальнейшем просто GPS) и отечественная система ГЛОНАСС (Глобальная навигационная спутниковая система).

В последнее время системы GPS получили гораздо более широкое распространение, чем ГЛОНАСС. Помимо доступности приемников типа GPS (приемники отечественной системы стоят в среднем в 2-3 раза дороже), их достоинством является то, что несколько лет назад с сигналов спутников убрали искусственные помехи для гражданских пользователей, что позволило довести точность определения координат для конечного потребителя до уровня 10-15 м. Еще одним фактором в пользу GPS до последнего времени является неполнота спутниковой группировки системы ГЛОНАСС.

В общем случае пользователь спутниковой навигационной системы при наличии навигационного приемника может определить координаты только своего местонахождения. Задачи определения местонахождения автомашин, других транспортных средств, ценных грузов и т.п. актуальны как для государственных правоохранительных органов, так и для частных структур безопасности. Такие задачи приходится решать в процессе управления патрульными службами и контроля перемещения подвижных объектов, обеспечения безопасности автомашин и их поиска в случае угона, сопровождении транспортных средств и ценных грузов. Актуальными являются задачи автоматизированного местоопределения подвижных

объектов в составе систем комплексного обеспечения безопасности. Система автоматического определения местоположения движущегося объекта обычно состоит из подсистемы определения местоположения, подсистемы передачи данных и подсистемы управления и обработки данных (рис. 1).

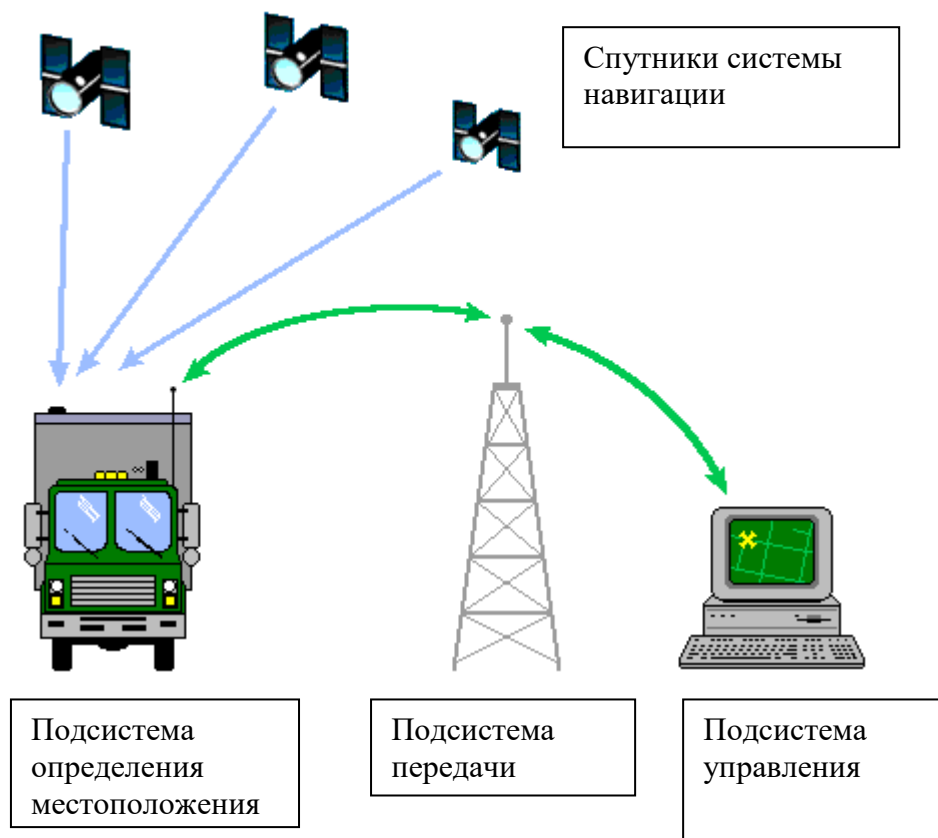


Рис. 1. Система автоматического определения местоположения движущегося объекта

Целью данной работы является создание программного комплекса, реализующего определение местоположения удаленных объектов, а также контроль над их перемещениями.

В данной области существуют более или менее успешные коммерческие разработки, но они не доступны для свободного пользования. Исходя из этого, поставленная задача актуальна и имеет практическую ценность.

## Принципы функционирования GPS-систем

NAVSTAR GPS (англ. NAVigation Satellites providing Time And Range; Global Positioning System – навигационные спутники, обеспечивающие измерение времени и расстояния; глобальная система позиционирования) – спутниковая система навигации, часто именуемая GPS.

Любая спутниковая навигационная система (СНС) включает в себя три сегмента (подсистемы):

- космический;
- наземный;
- сегмент потребителя.

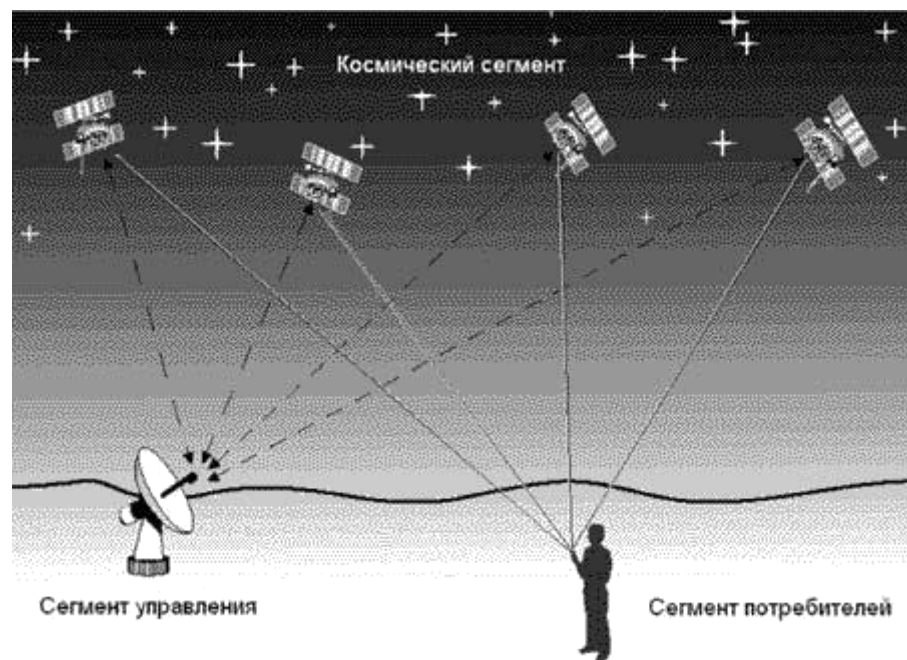


Рис. 2. Сегменты СНС

Космический сегмент представляет собой группу (созвездие) спутников, вращающихся вокруг земли и передающих на Землю радиосигналы, позволяющие потребительскому сегменту определять свои координаты.

Наземный сегмент обеспечивает поддержку архитектуры созвездия спутников, передает на спутники различные корректирующие и управляющие сигналы.

Потребительский сегмент любой СНС включает в себя множество пользователей, обладающих соответствующей навигационной аппаратурой (GPS-приемники).

### **Структура программного комплекса**

Как уже было сказано, целью данной работы является создание программного комплекса, реализующего определение местоположения удаленных объектов, а также контроль над их перемещениями.

Для определения местоположения объекта необходим только GPS-приемник, поэтому работа с СНС GPS будет осуществляться только в потребительском сегменте.

GPS-приемник позволяет получать данные о местоположении объекта, скорости движения и т.п. Необходимо программное обеспечение, реализующее получение данной информации с последующей ее передачей в центр сбора и обработки.

Центр сбора и обработки информации должен получать информацию от множества клиентов и анализировать ее. Возникает необходимость разработки и реализации программного обеспечения, выполняющего функции данного центра.

Очевидно удобство использования в данном случае клиент-серверной архитектуры программного комплекса, где центр сбора и обработки информации является серверным программным обеспечением, а приложение, работающее непосредственно с GPS-приемником, – клиентским.

Поскольку одной из целей данной работы является контроль движущихся объектов, серверное программное обеспечение должно осуществлять анализ полученной информации и делать выводы об отклонениях объектов от их маршрутов.

Схематическое представление данной архитектуры изображено на рис. 3.



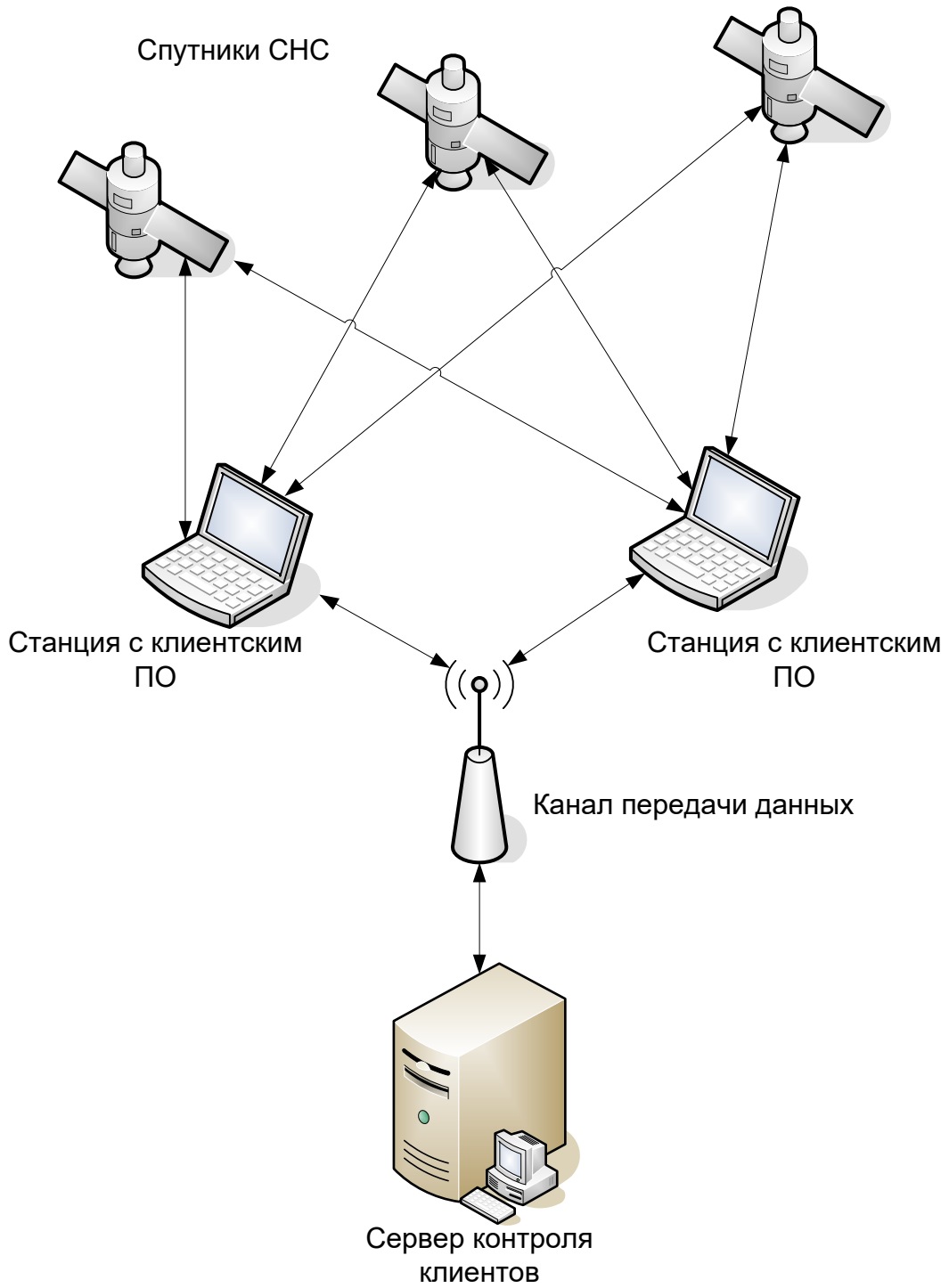


Рис. 3. Структура системы централизованного контроля движущихся объектов

## 1. Постановка задачи

Реализовать программный комплекс, который должен осуществлять централизованный контроль над большим количеством мобильных станций (движущихся объектов).

Программный комплекс должен быть реализован как клиент-серверная система.

Для достижения поставленной цели должны быть решены следующие задачи:

- анализ подходов местоопределения объекта на основе GPS-систем;
- разработка и реализация клиентского программного обеспечения, осуществляющего взаимодействие с мобильным GPS-приемником и передающее данные серверу;
- разработка серверной части программного комплекса, осуществляющего централизованный контроль мобильных клиентов (получение информации от клиентов, ее обработка и анализ).

## 2. Анализ задачи

### 2.1. Анализ структуры программного комплекса и его функций

Для осуществления централизованного контроля над мобильными объектами необходимо разработать клиент-серверный программный комплекс. Множество клиентов должны передавать информацию о своем местоположении, а сервер должен обрабатывать эту информацию и осуществлять контроль над клиентами.

Информация о местоположении клиента получается с помощью мобильного GPS-приемника.

Поскольку клиенты в общем случае находятся в движении, необходимо выбрать беспроводной канал связи со стационарной серверной частью. Наиболее дешевым и доступным решением в данном случае являются сети GSM. GPRS (General Packet Radio Service – пакетная радиосвязь общего пользования) – надстройка над технологией мобильной связи GSM, осуществляющая пакетную передачу данных. Сервисные узлы поддержки GPRS занимаются обработкой пакетной информации и преобразованием кадров данных GSM в форматы, используемые протоколами TCP/IP. Поэтому использование протокола GPRS прозрачно для сетей TCP/IP.

При получении информации от клиента по протоколу TCP/IP, сервер должен ее проанализировать. Если объект отклонился от маршрута, диспетчер должен получить сообщение об этом. Исходя из этого серверному программному обеспечению должна быть доступна информация о возможных маршрутах объекта.

Во время работы сервер должен принимать и обрабатывать непрерывные потоки информации, передаваемой множеством клиентов. В результате данной обработки должны формироваться реальные GPS-треки движущихся объектов. Для хранения этой информации, а также данных о возможных маршрутах, отклонениях объектов от маршрутов и другой сопутствующей информации необходима база данных.

Исходя из вышесказанного, можно составить общую схему функционирования системы централизованного контроля над движущимися объектами (рис. 2.1).

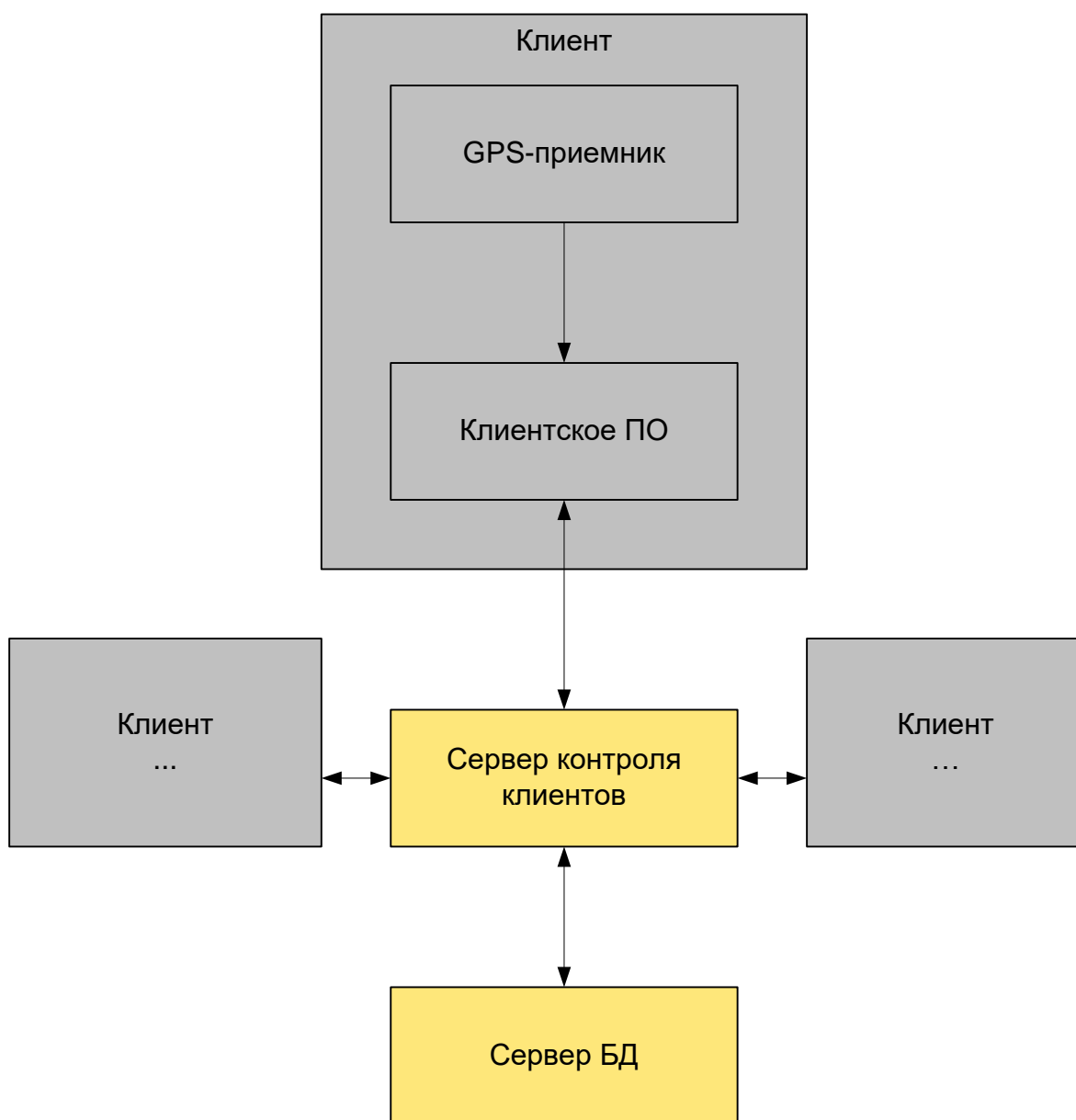


Рис. 2.2. Схема функционирования системы централизованного контроля движущихся объектов

## 2.2. Принципы функционирования GPS-приемников

GPS-приёмник – радиоприёмное устройство для определения географических координат текущего местоположения антенны приёмника, на основе данных о временных задержках прихода радиосигналов, излучаемых спутниками группы NAVSTAR. Максимальная точность измерения составляет 3-5 метров, а при наличии корректирующего сигнала от наземной станции – до 1 мм (обычно 5-10мм) на 1 км расстояния между станциями (дифференциальный метод). Точность коммерческих GPS-навигаторов составляет от 150 метров (у старых моделей при плохой видимости спутников) до 3 метров (у новых моделей на открытом месте).

В приемнике измеряется время распространения сигнала от спутника к приемнику и вычисляется дальность “спутник-приемник”. Поскольку для определения местоположения точки нужно знать три координаты (плоские координаты  $X$ ,  $Y$  и высоту  $Z$ ), то в приемнике должны быть измерены расстояния до трех различных спутников. При его использовании точное определение времени распространения сигнала возможно лишь при наличии синхронизации временных шкал спутника и приемника. Для этого в состав аппаратуры спутника и приемника входят эталонные часы. Бортовые часы всех спутников синхронизированы и привязаны к так называемому “системному времени”. Фактически, в измерениях времени всегда присутствует ошибка, возникающая из-за несовпадения шкал времени спутника и приемника. По этой причине в приемнике вычисляется искаженное значение дальности к спутнику или “псевдодальность”. Измерения расстояний до всех искусственных спутников Земли, с которыми в данный момент работает приемник, происходит одновременно. Следовательно, для всех измерений величину временного несоответствия можно считать постоянной. С математической точки зрения это эквивалентно тому, что неизвестными являются не только координаты  $X$ ,  $Y$  и

Z, но и поправка часов приемника. Для их определения необходимо выполнить измерения псевдодальностей не до трех, а до четырех спутников.

В результате обработки этих измерений в приемнике вычисляются координаты (X, Y и Z) и точное время. Если приемник установлен на движущемся объекте и наряду с псевдодальностями измеряет доплеровские сдвиги частот радиосигналов, то может быть вычислена и скорость объекта. Таким образом, для выполнения необходимых навигационных определений надо обеспечить постоянную видимость с нее, как минимум, четырех спутников. GPS система дает возможность в любой точке Земли наблюдать от 5 до 12 спутников в произвольный момент времени. Современные GPS-приемники имеют от 5 до 12 каналов, т.е. могут одновременно принимать сигналы от такого количества спутников. Избыточные измерения (сверх четырех) позволяют повысить точность определения координат и обеспечить непрерывность решения навигационной задачи.

При использовании GPS-приёмника информация выводится на карманный персональный компьютер, сотовый телефон или персональный компьютер, к которому подключен этот приемник с помощью навигационного программного обеспечения. Как правило, соединение осуществляется через последовательный порт (RS-232, USB, Bluetooth).

Для связи GPS-приёмника с компьютером может использоваться двоичный протокол производителя приёмника (Garmin, Magellan и другие), при этом абсолютное большинство GPS-приёмников поддерживают обмен информацией с помощью текстового протокола NMEA.

### 2.3. Протоколы обмена данными с мобильными GPS-приемниками

Как было сказано ранее, соединение GPS-приемника с компьютерной техникой в большинстве случаев осуществляется через последовательное соединение. Это может быть RS-232, USB, Bluetooth. В операционной системе Windows на более высоком уровне для обмена информацией через последовательное соединение используются COM-порты.

COM-порты или последовательные порты в операционной системе типа Windows – это именованные каналы для передачи данных, называемые обычно COM1, COM2 и т.д. по порядку обнаружения драйверов соответствующих устройств. Например, для обмена информацией через Bluetooth многие драйверы представляются операционной системе как COM-порт, и резервируют похожее имя. COM-порт – это двунаправленный последовательный интерфейс, предназначенный для обмена байтовой информацией. Последовательный потому, что информация через него передается по одному биту, бит за битом (в отличие от параллельного порта).

GPS-приемники независимо от того, какой физический интерфейс подключения они используют, передают данные через COM-порт. В ОС Windows работа с последовательными портами осуществляется посредством Win32 API. Обычно используется асинхронный режим чтения/записи. При работе с последовательным портом необходимо задать параметры конфигурации порта. Основные из них:

- BaudRate – скорость передачи данных. Возможны следующие значения: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000, 256000 бит/сек. GPS-приемники передают данные со скоростью 4800 или 9600 бит/сек.

- **ByteSize** – определяет число информационных бит в передаваемых и принимаемых байтах. Может принимать значение 4, 5, 6, 7, 8. Последнее время используется значение 8.
- **Parity** – определяет выбор схемы контроля четности. Данное поле должно содержать одно из следующих значений:
  - **EVENPARITY** — дополнение до четности;
  - **MARKPARITY** — бит четности всегда равен 1;
  - **NOPARITY** — бит четности отсутствует;
  - **ODDPARITY** — дополнение до нечетности;
  - **SPACEPARITY** — бит четности всегда 0.

Обычно используется значение **NOPARITY**.

- **StopBits** – задает количество стоповых бит. Поле может принимать следующие значения:
  - **ONESTOPBIT** — один стоповый бит;
  - **ONE5STOPBIT** — полтора стоповых бита (практически не используется);
  - **TWOSTOPBIT** — два стоповых бита.

Стандартом является использование одного стопового бита.

При передаче данных через последовательное соединение GPS-приемники используют протокол NMEA.

## 2.4. Протокол NMEA

NMEA («National Marine Electronics Association») – полное название «NMEA 0183» – текстовый протокол связи морского (как правило, навигационного) оборудования между собой. Стал особенно популярен в связи с распространением GPS-приёмников, использующих этот стандарт.



Информация, передаваемая GPS-приемником, состоит из строк следующего формата:

- символ «\$»;
- пятибуквенный идентификатор сообщения, первые две буквы – идентификатор источника сообщения, следующие три буквы – идентификатор формата сообщения, согласно протоколу NMEA 0183 определённой версии;
- список данных (буквы, цифры и точки), разделённых запятыми (если какие-либо данные отсутствуют внутри строки, запятые все равно ставятся (например «,,»); некоторые поля в конце строки могут отсутствовать вовсе);
- символ «\*»;
- двухзначное шестнадцатеричное число – контрольная XOR-сумма всех байт в строке между «\$» и «\*»;
- <CR><LF> (признак конца строки).

Полный набор NMEA-сообщений и команд достаточно велик, однако часто используют единственное сообщение: «рекомендуемый минимум навигационных данных RMC». Если приёмник GPS не настроен иначе, то, как правило, RMC-строки посылаются автоматически с интервалом в 1 секунду.

Формат стандартного RMC сообщения следующий:

```
$GPRMC,hhmmss.ss,A,GGMM.MM,P,gggmm.mm,J,v.v,b.b,ddmmyy,x.x,n,m*hh<CR><LF>
```

Ниже представлено описание каждого поля данного сообщения:

- «RMC» – (Recommended Minimum sentenceC) рекомендуемый минимум данных.
- «hhmmss.ss» – время фиксации местоположения по Гринвичу UTC: «hh» – часы, «mm» – минуты, «ss.ss» – секунды. Длина дробной части секунд варьируется. Лидирующие нули не опускаются.
- «A» – статус: «A» – данные достоверны, «V» — недостоверны.

- «GGMM.MM» – широта. Две цифры градусов («GG»), Две цифры целых минут, точка и дробная часть минут переменной длины. Лидирующие нули не опускаются.
- «P» – «N» для северной широты или «S» для южной.
- «gggmm.mm» – долгота. Три цифры градусов («ggg»), Две цифры целых минут, точка и дробная часть минут переменной длины. Лидирующие нули не опускаются.
- «J» – «E» для восточной долготы или «W» для западной.
- «v.v» – горизонтальная составляющая скорости относительно Земли в узлах. Число с плавающей точкой.
- «b.b» – путевой угол (направление скорости) в градусах. Число с плавающей точкой. Целая и дробная части переменной длины. Значение равное 0 соответствует движению на север, 90 – восток, 180 – юг, 270 – запад.
- «ddmmyy» – дата: день месяца, месяц, последние две цифры года (ведущие нули обязательны).
- «x.x» – магнитное склонение в градусах (часто отсутствует), рассчитанное по некоторой модели. Число с плавающей точкой. Целая и дробная части переменной длины.
- «n» – направление магнитного склонения: если значение «E», то для получения магнитного курса магнитное склонение необходимо вычесть из истинного курса, если «W» – прибавить к истинному курсу.
- «m» – индикатор режима: «A» – автономный, «D» – дифференциальный, «E» – аппроксимация, «N» – недостоверные данные (часто отсутствует, данное поле включая запятую отсутствует в старых версиях NMEA).
- «hh» – контрольная сумма.
- <CR> – байт равен 0x0D.
- <LF> – байт равен 0x0A.

Ниже приведен пример строки RMC:

```
$GPRMC,125504.049,A,5542.2389,N,03741.6063,E,0.06,25.82,200906,,
*3B
```

Этот пример расшифровывается следующим образом: строка формата RMC, время – 12 часов 55 минут 4,049 секунд UTC, данные достоверны, широта 55° 42,2389', северная, долгота 37° 41,6063', восточная, скорость 0,06 узлов, направление движения 25,82 градуса, 20 сентября (20)06 года, контрольная сумма 0х3В.

Существует множество других необязательных команд NMEA:

- GPGGA – данные о последнем определении местоположения;
- GPGLL – координаты, широта/долгота;
- GPGSA – DOP (GPS) и активные спутники;
- GPGSV – наблюдаемые спутники;
- GPWPL – параметры заданной точки;
- GPBOD – азимут одной точки относительно другой;
- GPRMB – рекомендуемый минимум навигационных данных для достижения заданной точки;
- GPRTE – маршруты;
- HCHDG – данные от компаса.

Ниже представлен пример данных, выдаваемый GPS-приемником Globalsat BT-338 каждую секунду.

```
$GPGGA,141742.746,,,,,0,00,,,M,0.0,M,0000*56
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPGSV,3,1,12,25,83,082,,13,78,075,,07,72,223,,27,64,234,*72
$GPGSV,3,2,12,23,39,102,,08,30,231,,16,24,045,,02,18,275,*73
$GPGSV,3,3,12,10,18,315,,04,12,241,,06,07,049,,03,06,091,*74
$GPRMC,141742.746,V,,,,,,,,,220408,,*23
```

GSA-команда (вторая строка) содержит информацию об активных спутниках. Поскольку в ней поля пустые, можно сделать вывод о том, что GPS-приемнику не удалось обнаружить спутники. В последней строке (RMC-команда) поля, несущие информацию о местоположении тоже пусты.

## **2.5. Анализ функций серверной части программного комплекса**

### **2.5.1. Структуры данных**

Клиентское программное обеспечение передает серверу по протоколу TCP/IP NMEA-данные, полученные с GPS-приемника. На основе этой информации можно сформировать текущий маршрут движения объекта.

Под маршрутом можно понимать последовательный набор точек с координатами (широта, долгота, высота).

В GPS-системах существует понятие GPS-трека. GPS-трек – набор данных, содержащий сведения о том, в какой момент времени в какой точке находится движущийся объект. GPS-трек составляется на основе NMEA-данных, получаемых от клиента во время движения.

Можно считать, что объект отклонился от маршрута, если выполняются два условия: во-первых, если объект отклонился от ближайших точек маршрута на определенное заранее расстояние и более; во-вторых, если объект не достигает контрольных точек маршрута вовремя или, наоборот, опережает график на заданное заранее время.

Отклонение по расстоянию можно определить следующим образом:

- 1) определяются две ближайшие точки маршрута;
- 2) найденные точки соединяются прямой;
- 3) определяется расстояние по нормали между прямой и объектом;
- 4) на основе полученных данных делается вывод об отклонении объекта.

## 2.5.2. Алгоритм вычисления расстояния между точками на земной поверхности

При вычислении отклонения объекта по расстоянию возникает проблема определения расстояния между двумя точками на поверхности земли.

Форма Земли отличается от шара и имеет несколько сплюсненную форму, близкую к сфероиду (эллипсоиду вращения), но истинная фигура Земли отличается и от сфероида, и от трехосного эллипсоида и не может быть представлена ни одной из известных математических фигур. Поэтому, говоря о фигуре Земли, имеют в виду не физическую форму земной поверхности, с океанами и материками, с их возвышенностями и впадинами, а, так называемую, поверхность геоида. Любая точка на земной поверхности представляется в геодезической системе координат (геодезическая широта, долгота и высота).

Для решения многих задач навигации и составления карт мелкого масштаба Землю принимают за сферу (шар). Положение точки на земной сфере определяется сферическими координатами: сферической широтой и сферической долготой.

Сферическая широта точки  $A$  – угол  $\varphi_A$  между плоскостью экватора и направлением  $R$  на данную точку из центра земной сферы. Сферическая долгота точки  $A$  – угол  $\lambda_A$ , заключенный между плоскостью нулевого (Гринвичского) меридиана и плоскостью меридиана данной точки. Средний радиус Земли  $R = 6371210$  м.

Законы сферической тригонометрии позволяют рассчитывать расстояния между точками, расположенными на сфере.

Кратчайшее расстояние между двумя точками на земной поверхности (если принять ее за сферу) определяется зависимостью:

$$\cos(d) = \sin(\varphi_A) \sin(\varphi_B) + \cos(\varphi_A) \cos(\varphi_B) \cos(\lambda_A - \lambda_B), \quad (2.1)$$

где  $\varphi_A$  и  $\varphi_B$  – широты,  $\lambda_A$ ,  $\lambda_B$  – долготы данных пунктов,  $d$  – расстояние между пунктами, измеряемое в радианах длиной дуги большого круга земного шара.

Расстояние между пунктами, измеряемое в километрах, определяется по формуле:

$$L = dR, \quad (2.2)$$

где  $R$  – средний радиус земного шара.

Для расчета расстояния между пунктами, расположенными в разных полушариях (северное-южное, восточное-западное), знаки ( $\pm$ ) у соответствующих параметров (широт или долгот) должны быть разными. Для простоты вычислений можно брать северную широту и восточную долготу со знаком плюс, соответственно – южную широту и западную долготу со знаком минус.

### 2.5.3. Функции элементов программного комплекса

Исходя из всего изложенного ранее, можно выделить три функциональных блока, составляющих в совокупности программный комплекс.

- Клиентское программное обеспечение:
  - взаимодействие с GPS-приемником;
  - передача NMEA-данных по протоколу TCP/IP.
- Серверное программное обеспечение:
  - поддержка соединения по протоколу TCP/IP с множеством клиентов;
  - разбор получаемого потока данных и извлечение из него полезной информации (местоположение объекта, время и т.д.);

- анализ полученной информации о текущих GPS-треках и уже имеющейся информации о маршрутах (поиск отклонений GPS-треков от заданных маршрутов);
- обновление информации в базе данных.
- Сервер базы данных:
  - хранение информации о GPS-треках, маршрутах и т.п.

### **3. Средства реализации**

Для хранения информации о маршрутах и треках удобно использовать базу данных. Необходимо выбрать СУБД. Оптимальным выбором здесь является использование СУБД FireBird, т.к. она обеспечивает целостность данных, механизм транзакций, может облегчить реализацию контроля корректности данных и не требует большого количества ресурсов. Не последнюю роль в выборе СУБД играет стоимость ее лицензирования для использования в коммерческих целях. FireBird распространяется по лицензии IPL (InterBase Public License) и полностью бесплатен для использования и распространения.

Средством разработки программного обеспечения выбрана среда Delphi, так как она хорошо подходит для написания программного обеспечения, использующего базы данных, кроме того, позволяет создавать клиент-серверные приложения.



#### **4. Требования к аппаратному и программному обеспечению**

Для работы программного комплекса необходимо следующее аппаратное обеспечение:

1. GPS-приемник Globalsat BT-338 или любой другой, имеющий последовательный интерфейс связи (Bluetooth или RS-232) и передающий данные по стандартному NMEA-протоколу.
2. IBM-совместимый мобильный компьютер (notebook) с возможностью подключения GPS-приемника по последовательному интерфейсу, с постоянным выходом в Интернет (GSM/GPRS). Требования к аппаратному обеспечению обуславливаются требованиями самой операционной системы.
3. IBM-совместимый компьютер для серверной части программного комплекса с постоянным подключением к сети Интернет. Требования к аппаратному обеспечению обуславливаются рекомендуемыми требованиями самой операционной системы.
4. Сервер баз данных FireBird 1.5. Физически СУБД может находиться на том же компьютере, что и сервер данного программного комплекса. Размер необходимого дискового пространства зависит от количества хранимых данных.

Программное обеспечение:

Для функционирования и клиента, и сервера необходимо наличие установленной операционной системы Windows NT/2000/XP/2003/Vista.

Для работы серверной части программного комплекса необходим сервер баз данных FireBird 1.5.

## 5. Интерфейс пользователя

### 5.1. Интерфейс клиентского приложения

При запуске клиентского приложения «ComClient» пользователь видит главное окно программы, представленное на рис. 5.1.

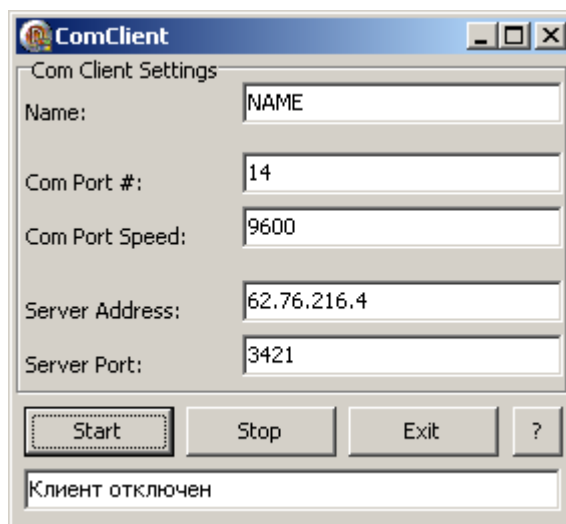


Рис. 5.1. Главное окно программы «ComClient»

Перед началом работы приложения следует произвести необходимые начальные настройки:

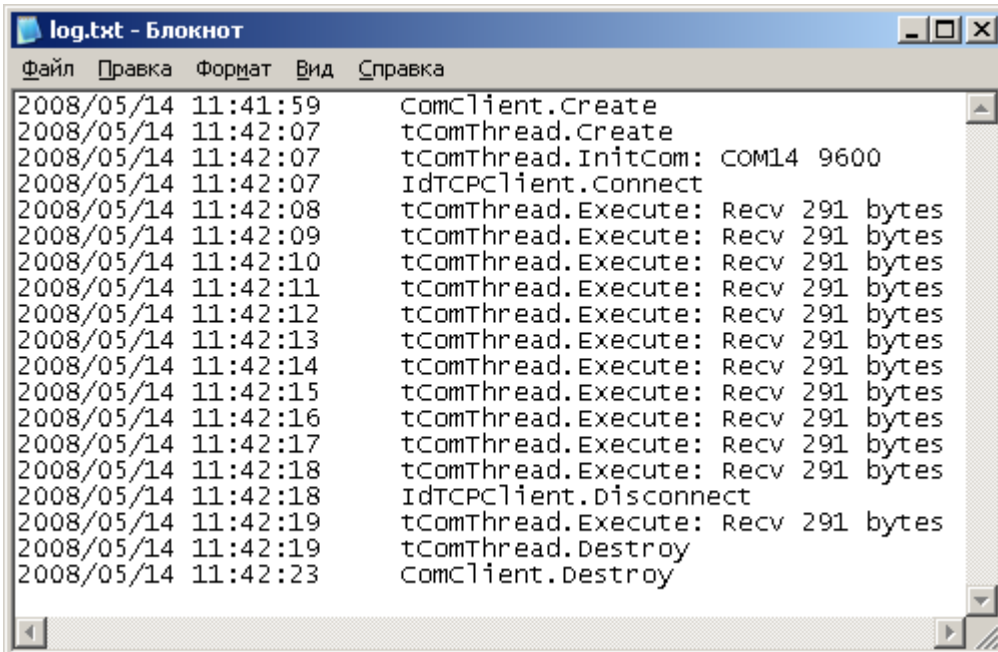
- Name – уникальное имя клиента, должно быть различным у всех клиентов подключаемых к серверу;
- Com Port # – номер Com-порта, зарезервированный для физического порта, к которому подключен GPS-приемник, в системе Windows;
- Com Port Speed – скорость передачи данных Com-порта, к которому подключено устройство.
- Server Address – адрес сервера, принимающего подключение по протоколу TCP/IP;
- Server Port – порт, на который осуществляется подключение к серверу.

Все настройки сохраняются при корректном завершении программы в системном реестре Windows. Поэтому нет необходимости производить настройку программы при следующем запуске.

После завершения начальных настроек приложения необходимо нажать кнопку «Start». При этом выполняется следующая последовательность действий: происходит подключение к Com-порту, потом подключение к серверу и после ответа сервера начинается передача данных. Отключение происходит при нажатии кнопки «Stop».

При нажатии кнопки «Exit» происходит отключение и после происходит завершение работы программы.

Для проведения диагностики работы программы и выявления ошибок удобно использовать лог-файл. Все события, а также сообщения об ошибках регистрируются в текстовом файле «log.txt», находящемся в одном каталоге с самой программой. На рис. 5.2. приведен пример файла «log.txt», полученного после нормального сеанса работы клиентской программы.



```
log.txt - Блокнот
Файл  Правка  Формат  Вид  Справка
2008/05/14 11:41:59 ComClient.Create
2008/05/14 11:42:07 tComThread.Create
2008/05/14 11:42:07 tComThread.InitCom: COM14 9600
2008/05/14 11:42:07 IdTCPClient.Connect
2008/05/14 11:42:08 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:09 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:10 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:11 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:12 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:13 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:14 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:15 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:16 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:17 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:18 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:18 IdTCPClient.Disconnect
2008/05/14 11:42:19 tComThread.Execute: Recv 291 bytes
2008/05/14 11:42:19 tComThread.Destroy
2008/05/14 11:42:23 ComClient.Destroy
```

Рис. 5.2. Пример файла «log.txt»

## 5.2. Интерфейс серверного приложения

При запуске серверного приложения «Navigation Server» пользователь видит главное окно программы, представленное на рис. 5.3.

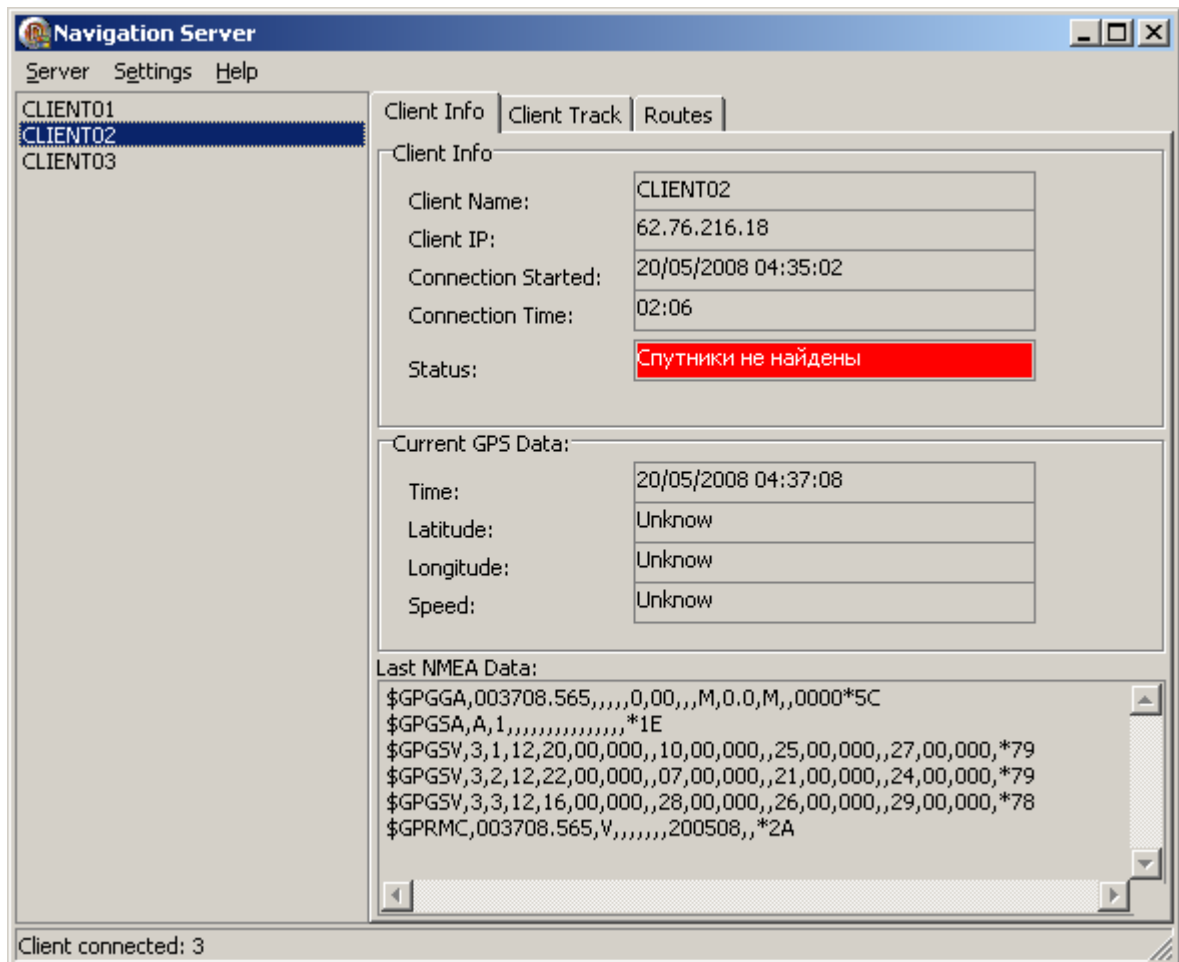


Рис. 5.3. Главное окно приложения «Navigation Server»

Главное меню содержит следующие пункты.

- Server
  - Start Listen – запускает сервер.
  - Disconnect All – отключает всех клиентов.
  - Exit – завершение работы программы.
- Settings – настройки программы. При выборе данного пункта меню, отображается форма, содержащая необходимые для работы сервера настройки (часовой пояс, порт, адрес сервера базы данных, имя пользователя базы данных, пароль). Перед началом

работы необходимо произвести настройки приложения, в дальнейшем нет необходимости это делать, так как все настройки сохраняются в системном реестре Windows.

- Help – вывод справочных сведений о программе.

Главное окно приложения состоит из двух элементов: списка подключенных клиентов и поля вывода информации. При выборе определенного клиента слева, справа выводится о нем информация.

Поле вывода информации состоит из трех вкладок.

- Client Info (рис. 5.3). Здесь отображается информация о подключенном клиенте, данные о его текущем местоположении, а также последние полученные от него NMEA-данные. Поле «Status» отображает текущее состояние объекта, если возникает исключительная ситуация (отклонение от маршрута, потеря спутников) данное поля выделяется красным цветом.
- Client Track (рис. 5.4). Данная вкладка содержит таблицу с данными GPS-трека текущего клиента.

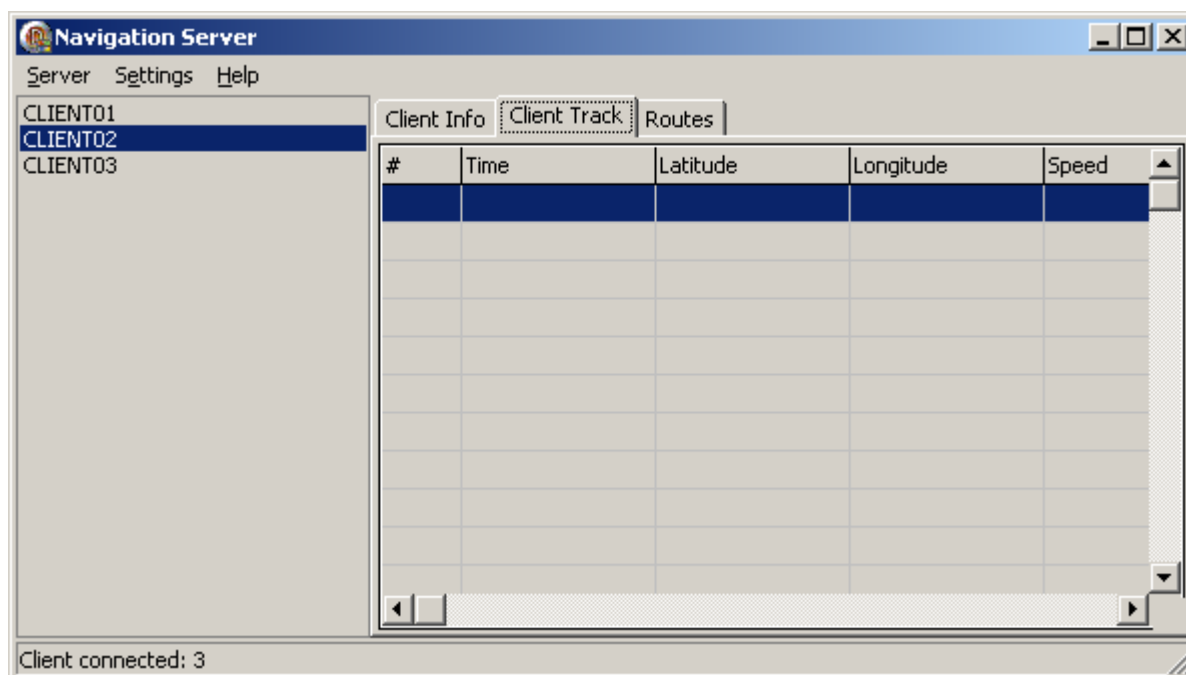


Рис. 5.4. Главное окно приложения «Navigation Server», вкладка «Client Track»

- Routes (рис. 5.5). В данной закладке можно просмотреть все имеющиеся в базе данных маршруты, а также назначить их выбранному клиенту.

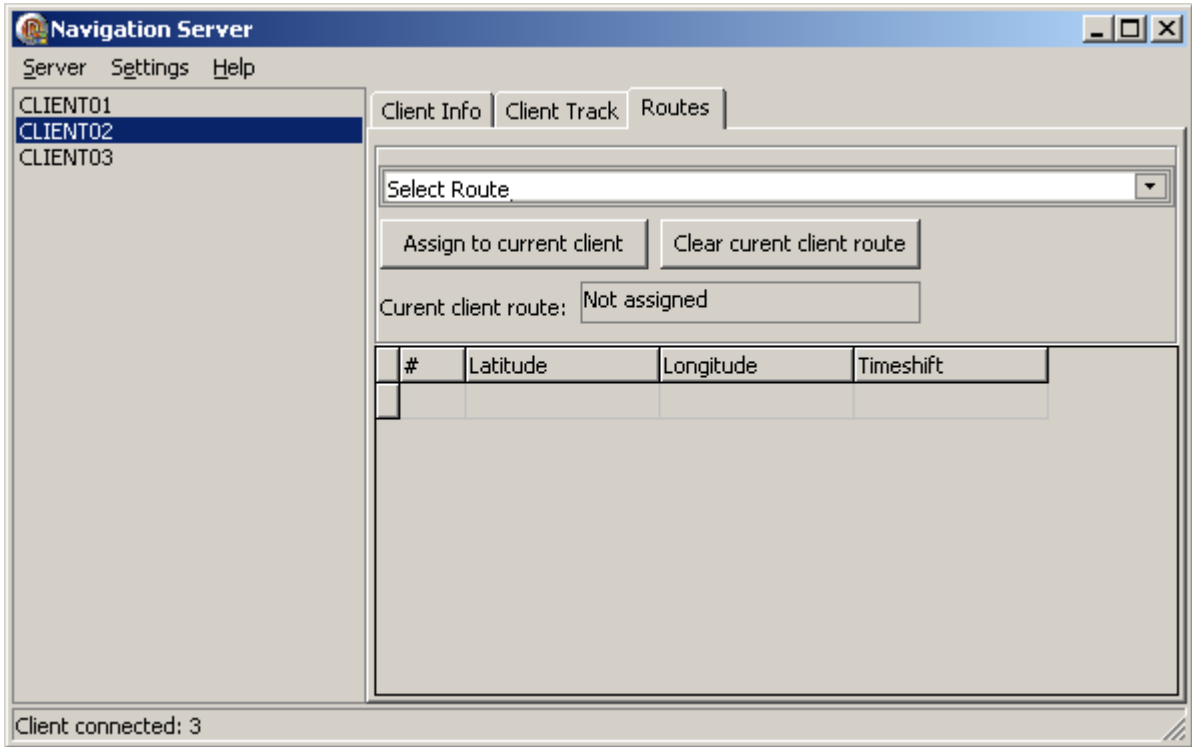


Рис. 5.5. Главное окно приложения «Navigation Server», закладка «Routes»

## 6. Реализация

### 6.1. Структура программного комплекса

В соответствии с поставленными задачами структуру проекта можно представить в виде схемы, изображенной на рис. 6.1.

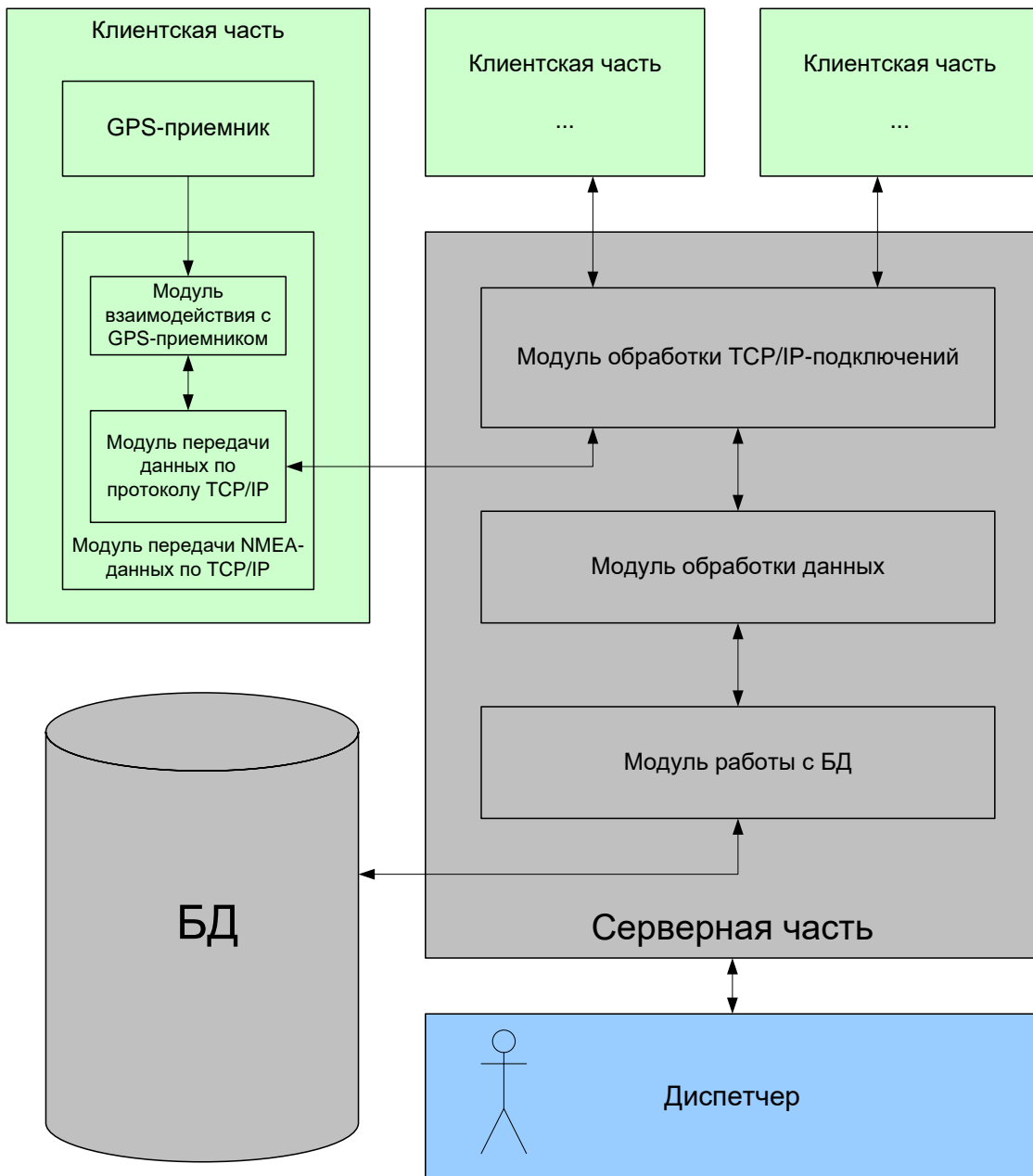


Рис. 6.1. Структура программного комплекса

## **6.2. Описание блоков программного комплекса**

### **1. Блок, реализующий клиентскую часть программного комплекса.**

Программное обеспечение этого блока получает «сырые» NMEA-данные с портативного GPS-приемника, передает их на сервер по протоколу TCP/IP, а также принимает управляющие команды сервера и определенным образом реагирует на них.

### **2. Блок, реализующий серверную часть программного комплекса.**

Серверное программное обеспечение состоит из нескольких модулей. Модуль обработки TCP/IP принимает подключения от множества клиентов и передает полученную информацию блоку обработки данных, а также передает клиентам управляющие команды (приостановка передачи данных, отключение и т.д.). Модуль обработки данных делает разбор полученной от клиентов информации и анализирует ее. При возникновении необходимости модуль обработки данных запрашивает информацию у блока работы с БД. После анализа данных, при возникновении нестандартных ситуаций (отклонение от маршрута), происходит оповещение пользователя (диспетчера). В конце обработки данных блок работы с БД сохраняет информацию в базу данных.

### **3. База данных.**

В базе данных содержится информация о всех допустимых маршрутах, реальных GPS-треках всех объектов, а также информация об исключительных ситуациях (отклонения от маршрута).



### 6.3. Структура базы данных

Модель данных, описывающая структуру базы данных, разработана с помощью ERWin v4.0 и представлена на рис. 6.2.

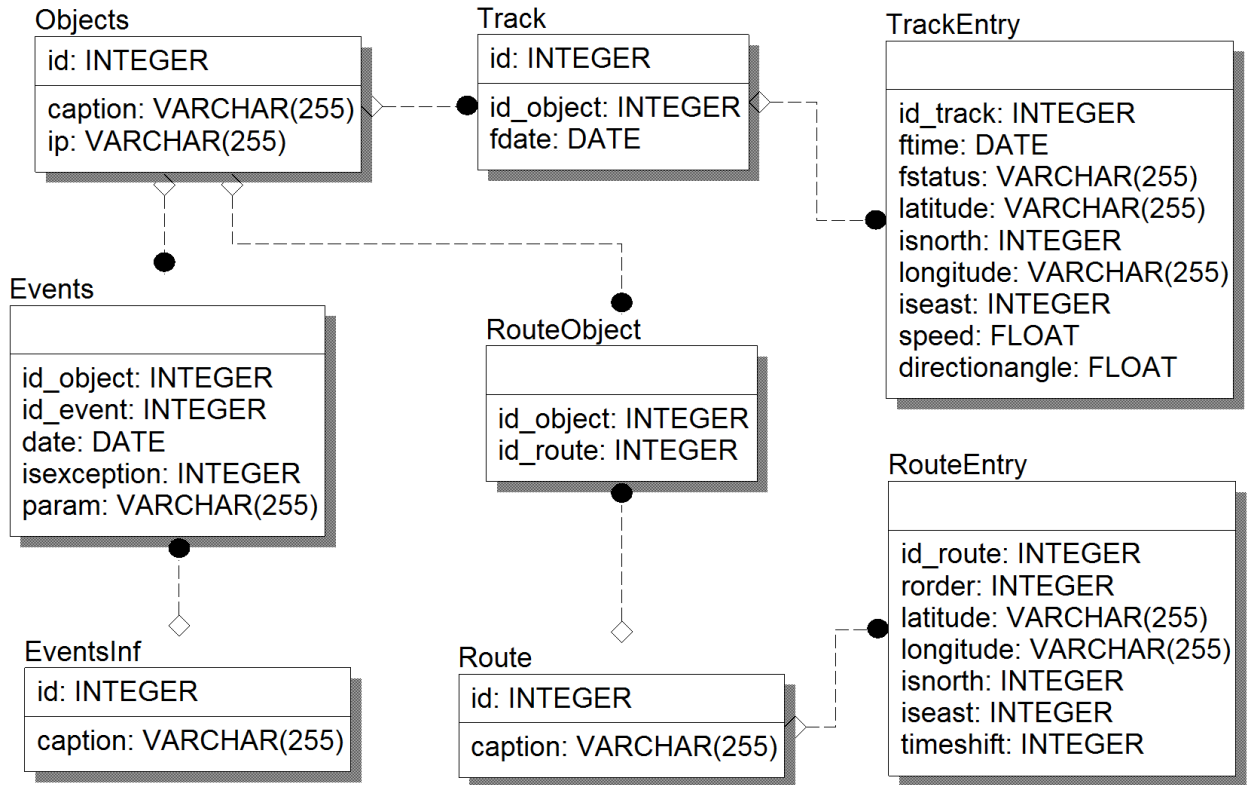


Рис. 6.2. ER-диаграмма базы данных

База данных состоит из следующих таблиц.

Таблицы Track, TrackEntry содержат информацию о GPS-треках движущихся объектах.

Таблицы RouteObject, Route, RouteEntry содержат данные о заранее заданных маршрутах движения объектов.

Таблицы Events, EventsInf содержат информацию о различных событиях и исключительных ситуациях (подключение объекта, отклонения от маршрута, потеря спутника и т.д.).

Таблица Objects включает данные об объектах, за которыми производится или производилось слежение.

Таблица EventsInf информационная таблица, содержащая названия всех событий. Возможны следующие события:

- connect – регистрируется при подключении клиента;
- connecterr – возникает, когда сервер неожиданно теряет соединение с клиентом;
- disconnect – возникает при нормальном отключении клиента;
- satelliteok – регистрируется, когда клиент находит необходимое для работы спутников;
- satelliteerr – регистрируется в случае, если в области видимости клиента находится недостаточное для работы количество спутников;
- routeerr – возникает в случае отклонения объекта от маршрута по расстоянию;
- routetimeerr – регистрируется при отклонении от маршрута по времени;
- routeok – возникает в том случае, если объект после отклонения от маршрута восстанавливает свое движение в соответствии с маршрутом.

В приложении 1 представлено более подробное описание каждой из таблиц.

В приложении 2 представлен sql-скрипт предназначенный для создания описанной базы данных.

#### **6.4. Протокол обмена данными между клиентом и сервером**

Разработан следующий протокол обмена данными между клиентом и сервером. Все данные передаются по сетям TCP/IP и состоят из сообщений.

Сообщения в свою очередь состоят из команды и могут содержать полезную информацию:

```
command [data]
```

Ниже представлен список всех возможных сообщений клиента.

- **Connect:**

```
conn name
```

Данная команда отправляется клиентом серверу при подключении и содержит уникальное имя клиента.

- **Data:**

```
data nmea
```

Эта команда передается клиентом с определенной периодичностью и содержит NMEA-данные, полученные с GPS-приемника.

- **Disconnect:**

```
disconn
```

Данная команда отправляется клиентом при отключении от сервера.

Далее представлены сообщения сервера, передаваемые клиенту:

- **Ready:**

```
ready
```

Отправляется сервером после подключения клиента и говорит о готовности принимать NMEA-данные.

- **Next:**

```
next
```

Отправляется сервером при готовности получить новый пакет данных.

- **Disconnect:**

```
disconn
```

Отправляется сервером при необходимости принудительного отключения клиента.

Сеанс связи между клиентом и сервером проходит по следующему алгоритму.

1. Клиент отправляет при подключении команду `Connect`.
2. Сервер делает необходимые операции и после их выполнения отправляет команду `Ready`.
3. При наличии в буфере неотправленных NMEA-данных клиент их передает в сообщении `Data` с периодичностью не чаще чем раз в 50 миллисекунд. Для передачи следующей последовательности данных необходим ответ сервера в виде сообщения `Next`.
4. При необходимости отключения клиента, сервер отправляет команду `Disconnect`.
5. При завершении работы или при получении команды `Disconnect` от сервера клиент также отправляет команду `Disconnect` и закрывает соединение.
6. При получении от клиента команды `Disconnect` сервер так же закрывает соединение.

## 6.5. Структура клиентской части

### 6.5.1. Схема работы клиентского приложения

Клиентское приложение «ComClient» состоит из восьми модулей. На рис. 6.3 представлена схема взаимодействия модулей.

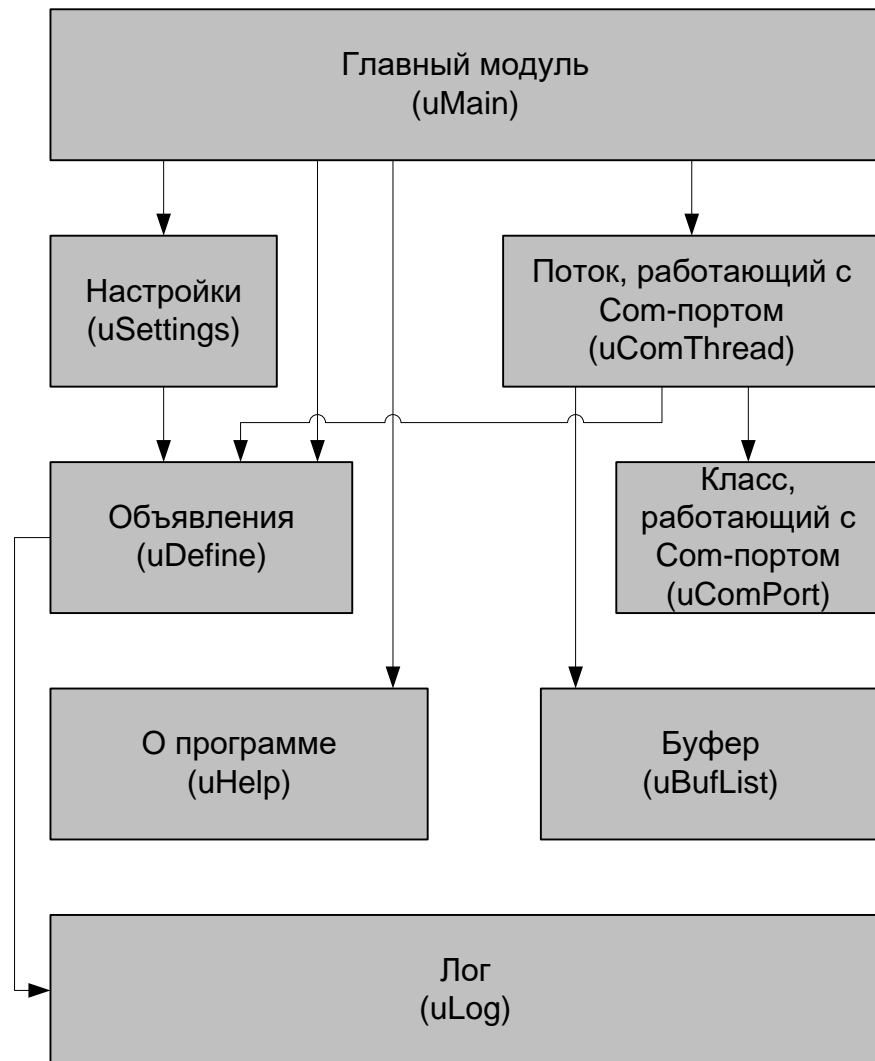


Рис. 6.3. Схема взаимодействия модулей клиентского приложения

#### *Модуль uMain:*

В модуле `uMain` реализован интерфейс пользователя и происходит вызов всех основных процедур программы. Также в данном модуле находится реализация подключения программы к серверу по протоколу TCP/IP.

### *Модуль uSettings:*

В модуле `uSettings` содержится класс, реализующий сохранение настроек программы в системном реестре Windows, а также их чтение при запуске приложения. Сохранение происходит при завершении работы программы.

### *Модуль uComThread или ComPort:*

Модуль `uComThread` содержит класс `tComThread`. Данный класс является потоком и предназначен для работы с Com-портом, к которому подключен GPS-приемник, а также предоставляет модулю `uMain` данные, считанные с приемника. Непосредственную работу с Com-портом осуществляет класс `tComPort`, реализованный в модуле `uComPort`.

### *Модуль uBufList:*

В модуле `uBufList` реализован класс `tBufList`, который является списком из строк. `tBufList` используется классом `tComThread` для сохранения прочитанных из Com-порта данных до их отправления серверу по протоколу TCP/IP.

### *Модуль uDefine:*

Модуль `uDefine` содержит объявления типов данных и переменных, используемых во многих модулях.

В модуле `uLog` реализован класс `tLog`. Данный класс производит регистрацию различных событий и исключений во время работы программы и сохраняет полученные данные в файл `log.txt`.

### *Модуль uHelp:*

`uHelp` содержит описание формы `About`. Данная форма содержит о названии приложения и версии.

Приложение работает следующим образом. При запуске программы происходит инициализация начальных настроек, потом пользователю дается возможность изменить настройки. Далее после нажатия кнопки «Start»

происходит создание потока `tComThread` и инициализация Com-порта. После производится запуск потока и непрерывное чтение данных с GPS-приемника с интервалом не менее 25 мсек. Данное значение задержки чтения является более чем достаточным, поскольку большинство GPS-приемников отправляют данные с интервалом в одну секунду. При успешной инициализации Com-порта осуществляется подключение к серверу и при получении от сервера ответа начинается передача данных по протоколу TCP/IP с интервалом не меньше чем 50 мсек.

Завершение работы приложения происходит в следующей последовательности:

- 1) закрытие сетевого подключения;
- 2) закрытие подключения к Com-порту;
- 3) уничтожение потока `tComThread`;
- 4) сохранение настроек в системном реестре Windows;
- 5) завершение работы программы.

### 6.5.2. Описание основных классов, процедур и функций

Ниже представлено описание интерфейсной части основных компонентов программы «ComClient», полный листинг приведен в приложении 2.

*Основные процедуры модуля `uMain`:*

```
procedure TfmMain.btStartClick(Sender: TObject);  
procedure TfmMain.btStopClick(Sender: TObject);  
procedure TfmMain.tmSendTimer(Sender: TObject);
```

Процедура `btStartClick` является обработчиком события, возникающего при нажатии на кнопку `Start` главного окна программы. Данная процедура производит запуск потока, работающего с Com-портом,

открывает соединение с сервером и запускает непрерывную передачу данных.

Метод `btStopClick` осуществляет обратные действия и вызывается при нажатии на кнопку `Stop` или при принудительном отключении клиента сервером.

Метод `tmSendTimer` запускается раз в 50 мсек при условии успешного запуска `btStartClick`. Данный метод ждет ответ от сервера и при наличии в буфере неотправленных NMEA-данных посылает их серверу.

На рис. 6.4 представлена диаграмма классов, разработанная с помощью Rational Rose v7.5.

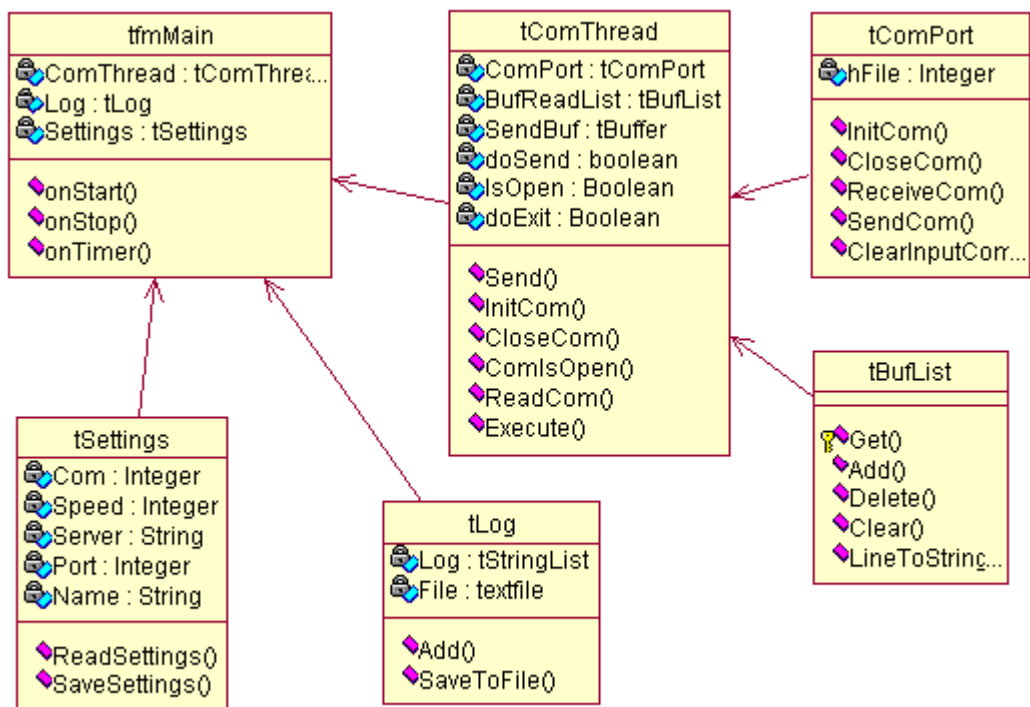


Рис. 6.4. Диаграмма классов клиентского приложения



Рассмотрим более подробно каждый из классов.

*Класс tComThread:*

```
tComThread = class (TThread)
private
  ComPort: tComPort;      // Класс для работы с Com-портом
  BufReadList: tBufList; // Буфер для считанных данных
  SendBuf: tBuffer;      // Буфер для отправляемых данных
  doSend: boolean;      // Есть ли отправленные данные
  IsOpen: boolean;      // Com-порт открыт
  fdoExit: boolean;     // Поток в процессе завершения работы
protected
public
  constructor Create(CreateSuspended: Boolean);
  destructor Destroy; override;
  (* Отправка данных в Com-порт *)
  procedure Send(Buf: tBuffer);
  (* Инициализация Com-порта *)
  function InitCom(BaudRate, PortNo: Integer; Parity: Char):
  Boolean;
  (* Закрытие Com-порта *)
  procedure CloseCom();
  (* Проверка открытости Com-порта *)
  function ComIsOpen: boolean;
  (* Доступ к буферу с прочитанными данными *)
  property ReadCom: tBufList read BufReadList;
  (* Главный метод потока *)
  procedure Execute; override;
  (* Поток находится в режиме завершения работы *)
  property doExit: boolean read fdoExit;
end;
```

Процедура `Send` кладет данные для отправки на Com-порт в буфер `SendBuf` и меняет флаг `doSend`. Сама отправка данных осуществляется в методе `Execute`.

Метод `InitCom` осуществляет начальную инициализацию Com-порта. В параметрах передается скорость, номер порта и схема контроля четности.

Процедура `CloseCom` закрывает Com-порт. Для продолжения работы с портом после вызова данной процедуры необходимо вызвать `InitCom`.

Функция `ComIsOpen` проверяет открыт ли порт и возвращает `true` в случае положительного результата и `false` в случае отрицательного.

Метод `Execute` является основным методом потока. После запуска потока он работает непрерывно в бесконечном цикле и с задержкой не менее чем 25 мсек читает данные из Com-порта.

*Класс tComPort:*

```

TComPort = class
private
  hFile: THandle; // Идентификатор Com-порта
public
  constructor Create;
  destructor Destroy; override;
  (* Инициализация Com-порта *)
  function InitCom(BaudRate, PortNo: Integer; Parity: Char;
    CommTimeOuts: TCommTimeouts): Boolean;
  (* Закрытие порта *)
  procedure CloseCom;
  (* Чтение данных *)
  function ReceiveCom(var Buffer; Size: DWORD): Integer;
  (* Запись данных *)
  function SendCom(var Buffer; Size: DWORD): Integer;
  (* Очистка буфера ввода *)
  function ClearInputCom: Boolean;
end;

```

Все методы данного класса вызываются из потока `tComThread` и осуществляют непосредственную работу с Com-портом.

Метод `InitCom` инициализирует Com-порт. Его следует вызывать перед обращениями ко всем остальным методам класса (кроме конструктора и деструктора).

Метод `CloseCom` закрывает Com-порт. Функции `ReceiveCom` и `SendCom` осуществляют соответственно чтение и запись данных в Com-порт.

*Класс tBufList:*

```

tBufList = class(tList)
protected
  function Get(Index: Integer): tBuffer;
public
  (* Добавление элемента *)
  procedure Delete(Index: integer);
  (* Очитска буфера *)
  procedure Clear();
  (* Добавление элемента *)
  function Add(Item: tBuffer): Integer;
  (* Представление элемента в виде строки *)
  function LineToString(Index: integer): string;
  (* Индексированный доступ к элементам *)
  property Items[Index: Integer]: tBuffer read Get; default;
end;

```

Класс `tBufList` представляет собой буфер для данных, считанных из Com-порта. Класс является реализацией списка, содержащего строки.

Метод `Delete` удаляет элемент с номером `Index`. Процедура производит удаление всех элементов списка. Функция `Add` осуществляет добавление элемента `Item` и возвращает позицию добавленного элемента в списке. Функция `LineToString` возвращает элемент с номером `Index` в виде строки. Свойство `Items` дает индексированный доступ на чтение к элементам списка.

*Класс `tSettings`:*

```
tSettings = class (tRegistry)
private
  fCom: integer;    // Номер Com-порта
  fSpeed: integer; // Скорость Com-порта
  fServer: string; // Сервер
  fPort: integer;  // Номер Порта
  fName: string;  // Имя клиента
public
  constructor Create();
  destructor Destroy(); override;
  (* Чтение настроек *)
  procedure ReadSettings;
  (* Сохранение настроек *)
  procedure SaveSettings;
  (* Свойства доступа к полям с настройками *)
  property Com: integer read fCom write fCom;
  property Speed: integer read fSpeed write fSpeed;
  property Server: string read fServer write fServer;
  property Port: integer read fPort write fPort;
  property Name: string read fName write fName;
end;
```

Данный класс осуществляет сохранение настроек программы (номер Com-порта, скорость, адрес сервера, порт сервера и имя клиента) и их чтение из системного реестра Windows. Методы `ReadSettings` и `SaveSettings` осуществляют соответственно чтение и запись настроек в реестр. Они вызываются при создании и уничтожении объекта данного класса.

Доступ к настройкам на чтение и на запись осуществляется посредством свойств `Com`, `Speed`, `Server`, `Port`, `Name`.

*Класс tLog:*

```

tLog = class
private
  log: tStringList; // Список всех сообщений лога
  f: textfile;      // Лог-файл
public
  constructor Create;
  destructor Destroy; override;
  (* Добавление сообщения в лог *)
  procedure Add(Text: string);
  (* Сохранение лог-файла *)
  procedure SaveToFile(fname: string);
  property GetLog: tStringList read log;
end;

```

Класс tLog осуществляет регистрацию событий и исключений во время работы приложения и сохраняет полученную информацию в файл «log.txt». Сообщения добавляются в лог с помощью метода Add. Метод сразу с добавлением сообщения в список дописывает его в файл «log.txt». Метод SaveToFile позволяет сохранить лог в файл с произвольным именем.

## **Заключение**

В дипломной работе разработан и реализован программный комплекс, осуществляющий централизованный контроль движущихся объектов на основе спутниковых навигационных систем.

В дальнейшем планируется совершенствование программного комплекса. А именно, реализация отображения движения объекта на векторных картах в реальном времени и реализация клиентской части для платформы PocketPC (карманные персональные компьютеры).

## Список литературы

1. Системы спутниковой навигации / Ю. А. Соловьев .– М. : Эко-трендз, 2000 .– 267 с.
2. Global Positioning Systems, Inertial Navigation and Integration / Mohinder S. Grewal, Lawrence R. Weill, Angus P. Andrews – New York : A John Wiley & Sons, Inc. , 2001. – 392 с.
3. Современные компьютерные сети / В. Столингс – СПб.: Питер – Петербург, 2003. – 784 с.
4. Delphi 7 в подлиннике /А.Хомоненко, В.Гофман, Е.Мещеряков и др.; под ред. А.Хомоненко. – СПб.: БХВ – Петербург, 2004. – 1200 с.
5. Объектно- ориентированное программирование: Учебник для вузов / Г.С. Иванова, Т.Н. Ничушкина, Е.К. Пугачев. – 2-е изд., перераб. и доп./ Под ред. Г.С. Ивановой. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2003. – 268 с.
6. Синтаксис регулярных выражений –  
(<http://www.php.net/manual/ru/reference.pcre.pattern.syntax.php>).
7. SQL Statement and Function Reference –  
([http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp\\_60\\_sqlref](http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_60_sqlref)).
8. NMEA – (<http://ru.wikipedia.org/wiki/NMEA>).
9. GPS – (<http://ru.wikipedia.org/wiki/GPS>).
10. The NMEA 0183 Protocol –  
(<http://www.tronico.fi/OH6NT/docs/NMEA0183.pdf>).

## Приложение 1. Описание таблиц базы данных

Таблица 1. Описание таблицы Objects

Название	Тип	Описание
id	integer	Первичный ключ
caption	varchar(255)	Название объекта
ip	varchar(255)	Ip-адрес, с которого последний раз производилось подключение

Таблица 2. Описание таблицы Track

Название	Тип	Описание
id	integer	Первичный ключ
id_object	integer	Внешний ключ, идентификатор объекта
fdate	date	Точная дата и время начала трека (время подключения клиента к серверу)

Таблица 3. Описание таблицы TrackEntry

Название	Тип	Описание
id_track	integer	Внешний ключ, идентификатор трека
ftime	date	Время создания записи
fstatus	varchar(255)	Статус достоверности данных
latitude	varchar(255)	Широта
isnorth	integer	Значение 1, если широта северная, иначе 0
longitude	varchar(255)	Долгота
iseast	integer	Значение 1, если долгота восточная, иначе 0
speed	float	Скорость движения
directionangle	float	Направление скорости

Таблица 4. Описание таблицы RouteObject

Название	Тип	Описание
id_object	integer	Внешний ключ, идентификатор объекта
id_route	integer	Внешний ключ, идентификатор маршрута

Таблица 5. Описание таблицы Route

Название	Тип	Описание
id	integer	Первичный ключ
caption	varchar(255)	Название маршрута

Таблица 6. Описание таблицы RouteEntry

Название	Тип	Описание
id_route	integer	Внешний ключ, идентификатор маршрута
rorder	integer	Номер вхождения для конкретного маршрута
latitude	varchar(255)	Широта
longitude	varchar(255)	Долгота
isnorth	integer	Значение 1, если широта северная, иначе 0
iseast	integer	Значение 1, если долгота восточная, иначе 0
timeshift	integer	время в секундах относительно первой точки маршрута (начало маршрута имеет значение «0»)



Таблица 7. Описание таблицы Events

Название	Тип	Описание
id_object	integer	Внешний ключ, идентификатор объекта
id_event	integer	Внешний ключ, идентификатор события
date	date	Точное время события
isexception	integer	Значение 1, если событие является исключением, иначе 0
param	varchar(255)	Полезная информация, например, адрес клиента при подключении клиента к серверу

Таблица 8. Описание таблицы EventsInf

Название	Тип	Описание
id	integer	Первичный ключ
caption	varchar(255)	Название события

## Приложение 2. Sql-скрипт создания базы данных

```

/*****
**/
/****      Generated by IBExpert 2.5.0.61 12.05.2008 4:22:39
****/
/*****
**/

SET SQL DIALECT 3;

SET NAMES WIN1251;

CREATE DATABASE 'DB.GDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 1024
DEFAULT CHARACTER SET WIN1251;

/*****
**/
/****      Generators
****/
/*****
**/

CREATE GENERATOR GEN_EVENTSINF_ID;
SET GENERATOR GEN_EVENTSINF_ID TO 0;

CREATE GENERATOR GEN_OBJECTS_ID;
SET GENERATOR GEN_OBJECTS_ID TO 0;

CREATE GENERATOR GEN_ROUTE_ID;
SET GENERATOR GEN_ROUTE_ID TO 0;

CREATE GENERATOR GEN_TRACK_ID;
SET GENERATOR GEN_TRACK_ID TO 0;

/*****
**/
/****      Tables
****/
/*****
**/

CREATE TABLE EVENTS (
    ID_OBJECT      INTEGER NOT NULL,
    ID_EVENT       INTEGER NOT NULL,
    FDATE          DATE NOT NULL,
    ISEXCEPTION    INTEGER,
    PARAM          VARCHAR(255)
);

CREATE TABLE EVENTSINF (
    ID              INTEGER NOT NULL,

```

```

    CAPTION  VARCHAR(255) NOT NULL
);

```

```

CREATE TABLE OBJECTS (
    ID          INTEGER NOT NULL,
    CAPTION    VARCHAR(255) NOT NULL,
    IP         VARCHAR(255)
);

```

```

CREATE TABLE ROUTE (
    ID          INTEGER NOT NULL,
    CAPTION    VARCHAR(255)
);

```

```

CREATE TABLE ROUTEENTRY (
    ID_ROUTE   INTEGER NOT NULL,
    RORDER    INTEGER NOT NULL,
    LATITUDE   VARCHAR(255),
    LONGITUDE  VARCHAR(255),
    ISNORTH    INTEGER,
    ISEAST     INTEGER,
    TIMESHIFT  INTEGER
);

```

```

CREATE TABLE ROUTEOBJECT (
    ID_OBJECT  INTEGER NOT NULL,
    ID_ROUTE   INTEGER NOT NULL
);

```

```

CREATE TABLE TRACK (
    ID          INTEGER NOT NULL,
    ID_OBJECT   INTEGER NOT NULL,
    FDATE       DATE NOT NULL
);

```

```

CREATE TABLE TRACKENTRY (
    ID_TRACK   INTEGER NOT NULL,
    FTIME      DATE NOT NULL,
    FSTATUS    VARCHAR(255),
    LATITUDE   VARCHAR(255),
    ISNORTH    INTEGER,
    LONGITUDE  VARCHAR(255),
    ISEAST     INTEGER,
    SPEED      FLOAT,
    DIRECTIONANGLE  FLOAT
);

```

```

/*****
**/
/****
****/
/*****
**/

```

*Unique Constraints*

```
ALTER TABLE ROUTEENTRY ADD CONSTRAINT UNQ_ROUTEENTRY UNIQUE (ID_ROUTE,
RORDER);
ALTER TABLE ROUTEOBJECT ADD CONSTRAINT UNQ_ROUTEOBJECT UNIQUE (ID_OBJECT);
ALTER TABLE TRACKENTRY ADD CONSTRAINT UNQ_TRACKENTRY UNIQUE (ID_TRACK,
FTIME);
```

```

/*****
**/
/****
****/
/*****
**/
```

```
ALTER TABLE EVENTSINF ADD CONSTRAINT PK_EVENTSINF PRIMARY KEY (ID);
ALTER TABLE OBJECTS ADD CONSTRAINT PK_OBJECTS PRIMARY KEY (ID);
ALTER TABLE ROUTE ADD CONSTRAINT PK_ROUTE PRIMARY KEY (ID);
ALTER TABLE TRACK ADD CONSTRAINT PK_TRACK PRIMARY KEY (ID);
```

```

/*****
**/
/****
****/
/*****
**/
```

```
ALTER TABLE EVENTS ADD CONSTRAINT FK_EVENTS_E FOREIGN KEY (ID_EVENT)
REFERENCES EVENTSINF (ID) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE EVENTS ADD CONSTRAINT FK_EVENTS_O FOREIGN KEY (ID_OBJECT)
REFERENCES OBJECTS (ID) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE ROUTEENTRY ADD CONSTRAINT FK_ROUTEENTRY FOREIGN KEY (ID_ROUTE)
REFERENCES ROUTE (ID) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE ROUTEOBJECT ADD CONSTRAINT FK_ROUTEOBJECT_O FOREIGN KEY
(ID_OBJECT) REFERENCES OBJECTS (ID) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE ROUTEOBJECT ADD CONSTRAINT FK_ROUTEOBJECT_R FOREIGN KEY
(ID_ROUTE) REFERENCES ROUTE (ID) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE TRACK ADD CONSTRAINT FK_TRACK FOREIGN KEY (ID_OBJECT) REFERENCES
OBJECTS (ID) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE TRACKENTRY ADD CONSTRAINT FK_TRACKENTRY FOREIGN KEY (ID_TRACK)
REFERENCES TRACK (ID) ON DELETE CASCADE ON UPDATE CASCADE;
```

```

/*****
**/
/****
****/
/*****
**/
```

```
SET TERM ^ ;
```

```

/* Trigger: EVENTSINF_BI */
CREATE TRIGGER EVENTSINF_BI FOR EVENTSINF
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID IS NULL) THEN
    NEW.ID = GEN_ID(GEN_EVENTSINF_ID,1);
END
```

^

```
/* Trigger: OBJECTS_BI */
CREATE TRIGGER OBJECTS_BI FOR OBJECTS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
    IF (NEW.ID IS NULL) THEN
        NEW.ID = GEN_ID(GEN OBJECTS ID,1);
    END
END
^
```

```
/* Trigger: ROUTE_BI */
CREATE TRIGGER ROUTE_BI FOR ROUTE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
    IF (NEW.ID IS NULL) THEN
        NEW.ID = GEN_ID(GEN ROUTE ID,1);
    END
END
^
```

```
/* Trigger: TRACK_BI */
CREATE TRIGGER TRACK_BI FOR TRACK
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
    IF (NEW.ID IS NULL) THEN
        NEW.ID = GEN_ID(GEN TRACK ID,1);
    END
END
^
```

```
SET TERM ; ^
```

### Приложение 3. Листинг клиентского приложения

```

//*****//
//****          ComClient v1.0                      ****//
//*****//

program comclient;

uses
  Forms,
  uBufList in 'uBufList.pas',
  uComPort in 'uComPort.pas',
  uComThread in 'uComThread.pas',
  uDefine in 'uDefine.pas',
  uHelp in 'uHelp.pas' {AboutBox},
  uLog in 'uLog.pas',
  uMain in 'uMain.pas' {fmMain},
  uSettings in 'uSettings.pas';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfmMain, fmMain);
  Application.CreateForm(TAboutBox, AboutBox);
  Application.Run;
end.

//****          End of ComClient                      ****//

//*****//
//****          ComClient v1.0 - uMain                ****//
//*****//

unit uMain;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, uHelp, uSettings,

  uComPort, uLog, uDefine, uComThread, StdCtrls, ComCtrls, Sockets, ExtCtrls,
  IdBaseComponent, IdComponent, IdTCPConnection, IdTCPClient;

//const
//  WM_NOTIFYTRAYICON = WM_USER + 1;

type
  TfmMain = class(TForm)
    TcpClient: TTcpClient;
    tmSend: TTimer;
    gbMain: TGroupBox;
    edCom: TEdit;
    edSpeed: TEdit;
    edServer: TEdit;
    edPort: TEdit;
    lbCom: TLabel;
    lbSpeed: TLabel;
  end;

```

```

lbServer: TLabel;
lbPort: TLabel;
lbName: TLabel;
edName: TEdit;
IdTCPClient: TIdTCPClient;
btStart: TButton;
btStop: TButton;
btExit: TButton;
btHelp: TButton;
edLog: TEdit;
procedure btStopClick(Sender: TObject);
procedure btStartClick(Sender: TObject);
procedure btExitClick(Sender: TObject);
procedure btHelpClick(Sender: TObject);
procedure TcpClientReceive(Sender: TObject; Buf: PAnsiChar;
  var DataLen: Integer);
procedure TcpClientSend(Sender: TObject; Buf: PAnsiChar;
  var DataLen: Integer);
procedure tmSendTimer(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  TrayState: boolean;
  doNext: boolean;
  doStatus: integer; // -1 -- disconnect; 0 -- waitready; 1 -- next; 2 --
doexit;
  procedure ComThreadDead(Sender: TObject); private
    //procedure WMTRAYICONNOTIFY(var Msg: TMessage); message
WM_NOTIFYTRAYICON;
  public

end;

var
  fmMain: TfmMain;

  ComThread: tComThread;
  Settings: tSettings;

implementation

{$R *.dfm}

procedure TfmMain.btExitClick(Sender: TObject);
begin
  btStopClick(btStop);
  Close;
end;

procedure TfmMain.btHelpClick(Sender: TObject);
begin
  uHelp.AboutBox.ShowModal;
end;

procedure TfmMain.btStartClick(Sender: TObject);
var temp: string;
begin
  if not tmSend.Enabled then
    try
      strtoint(edSpeed.Text);
      strtoint(edCom.Text);
      strtoint(edPort.Text);
      ComThread:=tComThread.Create(true);

```

```

    if not ComThread.InitCom(strtoint(edSpeed.Text), strtoint(edCom.Text),
    'N')
    then edLog.Text:='Com-порт недоступен'
    else begin
        IdTCPClient.Host:=edServer.Text;
        IdTCPClient.Port:=strtoint(edPort.Text);
        try
            IdTCPClient.Connect;
            if not {TcpClient.Active} IdTCPClient.Connected then
                edLog.Text:='Сервер недоступен'
            else begin
                Log.Add('IdTCPClient.Connect');
                edLog.Text:='Ожидание ответа сервера';
                ComThread.Resume;
                tmSend.Enabled:=true;
                temp:='conn '+edName.Text;
                doNext:=false;
                IdTCPClient.IOHandler.WriteLine(temp);
            end;
            doStatus:=0;
        except
            on E: Exception do
                begin
                    Log.Add('Exception: ' + E.Message);
                    edLog.Text:='Сервер недоступен';
                    btStopClick(btStop);
                end;
            end;
        except
            on E: Exception do
                begin
                    Log.Add('Exception: ' + E.Message);
                    MessageBox(0, PChar(E.Message), PChar(E.HelpContext), MB_OK or
                    MB_ICONERROR);
                    edLog.Text:='Некорректные настройки';
                end;
            end;
        end;
    end;

procedure TfmMain.btStopClick(Sender: TObject);
begin
    if ComThread <> nil then
        begin
            tmSend.Enabled:=false;
            doStatus:=-1;
            IdTCPClient.IOHandler.WriteLine('disconn');
            IdTCPClient.Disconnect;
            Log.Add('IdTCPClient.Disconnect');
            ComThread.Terminate;
            if not ComThread.Suspended then
                while not ComThread.doExit do
                    sleep(25);
            ComThread.Destroy();
            ComThread:=nil;
            edLog.Text:='Клиент отключен';
        end;
    end;

procedure TfmMain.ComThreadDead(Sender: TObject);
begin
    ComThread.Destroy;
    ComThread:=nil;
end;

```



```

procedure TfmMain.FormCreate(Sender: TObject);
begin
  TrayState:=false;
  doStatus:=-1;
  Mutex:=CreateMutex(nil, true, nil);
  Log:=tLog.Create;
  Settings:=tSettings.Create;
  edCom.Text:=inttostr(Settings.Com);
  edSpeed.Text:=inttostr(Settings.Speed);
  edServer.Text:=Settings.Server;
  edPort.Text:=inttostr(Settings.Port);
  edName.Text:=Settings.Name;
  ComThread:=nil;
  tmSend.Enabled:=false;
  Log.Add('ComClient.Create');
end;

procedure TfmMain.FormDestroy(Sender: TObject);
begin
  Settings.Com:=StrToInt(edCom.Text);
  Settings.Speed:=StrToInt(edSpeed.Text);
  Settings.Server:=edServer.Text;
  Settings.Port:=StrToInt(edPort.Text);
  Settings.Name:=edName.Text;
  Settings.Destroy;
  if ComThread <> nil then
    begin
      tmSend.Enabled:=false;
      IdTCPClient.Disconnect;
      ComThread.Terminate;
      while not ComThread.doExit do
        sleep(25);
      ComThread.Destroy();
      ComThread:=nil;
    end;
  Log.Add('ComClient.Destroy');
  Log.Destroy;
  CloseHandle(Mutex);
end;

procedure TfmMain.TcpClientReceive(Sender: TObject; Buf: PAnsiChar;
  var DataLen: Integer);
var
  Buffer: tBuffer;
  i: integer;
begin
  Log.Add('TcpClient: Recv Ok '+ inttostr(DataLen)+ ' bytes');
  MessageBox(0, '', Buf, 0);
  if Buf = 'ready'
    then begin
      tmSend.Enabled:=true;
      edLog.Text:='Передача данных';
    end;
end;

procedure TfmMain.TcpClientSend(Sender: TObject; Buf: PAnsiChar;
  var DataLen: Integer);
begin
  Log.Add('TcpClient: Send Ok '+ inttostr(DataLen)+ ' bytes');
end;

procedure TfmMain.tmSendTimer(Sender: TObject);
var

```

```

    str: string;
begin
    if (doStatus <> 2) and IdTCPClient.Connected then
    begin
        IdTCPClient.IOHandler.ReadTimeout:=10;
        if (not doNext) then
        begin
            str:=IdTCPClient.IOHandler.ReadLn;
            if (str = 'ready') or (str = 'next') then
                doNext:=true;
        end;
        if (doStatus = 0) and (str = 'ready') then
        begin
            doStatus:=1;
            str:='next';
            edLog.Text:='Передача данных';
            doNext:=true;
        end;
        if (doStatus = 1) and doNext then
        if ComThread <> nil then
            if (ComThread.ReadCom.Count > 0) and {TcpClient.Active}
IdTCPClient.Connected then
            begin
                str:=ComThread.ReadCom.LineToString(0);
                ComThread.ReadCom.Delete(0);
                IdTCPClient.IOHandler.WriteLine('data '+str);
                doNext:=false;
            end;
            if str = 'disconn' then
                doStatus:=2;
        end
        else
            Self.btStopClick(btStop);
    end;

end.

//*****          End of uMain          *****/

//*****
//*****          ComClient v1.0 - uComThread          *****/
//*****

unit uComThread;

interface

uses
    Classes, Windows, SysUtils,
    uDefine, uBufList, uLog, uComPort;

type
    (* Поток, работающий с Com-портом *)
    tComThread = class(TThread)
    private
        ComPort: tComPort;          // Класс для работы с Com-портом
        BufReadList: tBufList;      // Буфер для считанных данных
        SendBuf: tBuffer;           // Буфер для отправляемых на Com-порт данных
        doSend: boolean;           // Есть не отправленные данные
        IsOpen: boolean;           // Com-порт открыт
        fdExit: boolean;           // Поток в процессе завершения работы
    protected

```

```

public
  constructor Create(CreateSuspended: Boolean);
  destructor Destroy; override;
  (* Отправка данных в Com-порт *)
  procedure Send(Buf: tBuffer);
  (* Инициализация Com-порта *)
  function InitCom(BaudRate, PortNo: Integer; Parity: Char): Boolean;
  (* Закрытие Com-порта *)
  procedure CloseCom();
  (* Проверка открытости Com-порта *)
  function ComIsOpen: boolean;
  (* Доступ к буферу с прочитанными данными *)
  property ReadCom: tBufList read BufReadList;
  (* Главный метод потока *)
  procedure Execute; override;
  (* Поток находится в режиме завершения работы *)
  property doExit: boolean read fdoExit;
end;

implementation

{ tComThread }

procedure tComThread.CloseCom;
begin
  //WaitForSingleObject (Mutex, INFINITE);
  ComPort.CloseCom;
  IsOpen:=false;
  Log.Add('tComThread.CloseCom');
  //ReleaseMutex (Mutex);
end;

function tComThread.ComIsOpen: boolean;
begin
  Result:=IsOpen;
end;

constructor tComThread.Create(CreateSuspended: Boolean);
begin
  inherited Create(CreateSuspended);
  BufReadList:=tBufList.Create;
  doSend:=false;
  IsOpen:=false;
  ComPort:=tComPort.Create;
  fdoExit:=false;
  Log.Add('tComThread.Create');
end;

destructor tComThread.Destroy;
begin
  //WaitForSingleObject (Mutex, INFINITE);
  BufReadList.Clear;
  BufReadList.Destroy;
  ComPort.CloseCom;
  ComPort.Destroy;
  IsOpen:=false;
  Log.Add('tComThread.Destroy');
  inherited Destroy;
  //ReleaseMutex (Mutex);
end;

procedure tComThread.Execute;

```

```

var
  Buffer: tBuffer;
  count: integer;
begin
  while not Self.Terminated do
    begin
      if IsOpen then
        begin
          //WaitForSingleObject(Mutex, INFINITE);
          if doSend then
            begin
              try
                count:=ComPort.SendCom(Buffer, BUFFSIZE);
                doSend:=false;
                Log.Add('tComThread.Execute: Send ' + inttostr(count) + '
bytes');
              except
                on E: Exception do
                  begin
                    Log.Add('Exception: '+E.Message);
                    MessageBox(0, PChar(E.Message), PChar(E.HelpContext),
MB_OK or MB_ICONERROR)
                  end;
                end;
            end;
            FillChar(Buffer, BUFFSIZE, #0);
            try
              count:=ComPort.ReceiveCom(Buffer, BUFFSIZE);
              Log.Add('tComThread.Execute: Recv ' + inttostr(count) + '
bytes');
            except
              on E: Exception do
                begin
                  Log.Add('Exception: ' + E.Message);
                  MessageBox(0, PChar(E.Message), PChar(E.HelpContext), MB_OK
or MB_ICONERROR)
                end;
              end;
            //ReleaseMutex(Mutex);
            end;
            Sleep(25);
            end;
            fdoExit:=true;
          end;

function tComThread.InitCom(BaudRate, PortNo: Integer; Parity: Char):
Boolean;
var
  ct: _COMMTIMEOUTS;
begin
  Result:=false;
  ct.ReadIntervalTimeout := 50;
  ct.ReadTotalTimeoutConstant := 50;
  ct.ReadTotalTimeoutMultiplier := 10;
  ct.WriteTotalTimeoutConstant := 50;
  ct.WriteTotalTimeoutMultiplier := 10;
  try
    //WaitForSingleObject(Mutex, INFINITE);
    Result:=ComPort.InitCom(BaudRate, PortNo, Parity, ct);
    if Result then
      begin
        IsOpen:=true;

```

```

        Log.Add('tComThread.InitCom: COM' + inttostr(PortNo) + ' ' +
inttostr(BaudRate));
        end
        else
        Log.Add('tComThread.InitCom: Error COM' + inttostr(PortNo) + ' ' +
inttostr(BaudRate));
        //ReleaseMutex(Mutex);
    except
        on E: Exception do
            begin
                Log.Add('Exception: ' + E.Message);
                MessageBox(0, PChar(E.Message), PChar(E.HelpContext), MB_OK or
MB_ICONERROR);
                IsOpen:=false;
                //ReleaseMutex(Mutex);
            end;
        end;
    end;

procedure tComThread.Send(Buf: tBuffer);
begin
    //WaitForSingleObject(Mutex, INFINITE);
    SendBuf:=Buf;
    doSend:=true;
    //ReleaseMutex(Mutex);
end;

end.

//*****          End of uComThread          *****/

//*****
//*****          ComClient v1.0 - uComPort          *****/
//*****

unit uComPort;

interface

uses
    Windows, SysUtils;

type
    (* Класс, осуществляющий работу с Com-портом *)
    TComPort = class
    private
        hFile: THandle; // Идентификатор Com-порта
    public
        constructor Create;
        destructor Destroy; override;
        (* Инициализация Com-порта *)
        function InitCom(BaudRate, PortNo: Integer; Parity: Char;
CommTimeOuts: TCommTimeouts): Boolean;
        (* Закрытие порта *)
        procedure CloseCom;
        (* Чтение данных *)
        function ReceiveCom(var Buffer; Size: DWORD): Integer;
        (* Запись данных *)
        function SendCom(var Buffer; Size: DWORD): Integer;
        (* Очистка буфера ввода *)
        function ClearInputCom: Boolean;
    end;
end;

```

```

implementation

constructor TComPort.Create;
begin
  inherited Create;
  Self.hFile := INVALID_HANDLE_VALUE;
  CloseCom;
end;

destructor TComPort.Destroy;
begin
  CloseCom;
  inherited;
end;

function TComPort.InitCom(BaudRate, PortNo: Integer; Parity: Char;
  CommTimeOuts: TCommTimeouts): Boolean;
var
  FileName: string;
  DCB: TDCB;
  PortParam: string;
begin
  result := FALSE;
  FileName := '\\.\COM' + IntToStr(PortNo); {имя файла}
  hFile := CreateFile(PChar(FileName),
    GENERIC_READ or GENERIC_WRITE, 0, nil,
    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  if hFile = INVALID_HANDLE_VALUE then
    begin
      raise Exception.Create('TComPort.InitCom: CreateFile
'+PChar(FileName)+' Error. ');
      exit;
    end;

  //установка требуемых параметров
  GetCommState(hFile, DCB); //чтение текущих параметров порта
  PortParam := 'baud=' + IntToStr(BaudRate) + ' parity=' + Parity +
    ' data=8 stop=1 ' +
    'octs=off';
  if BuildCommDCB(PChar(PortParam), DCB) then
    begin
      result := SetCommState(hFile, DCB) and
        SetCommTimeouts(hFile, CommTimeOuts);
    end;
  if not result then
    begin
      raise Exception.Create('TComPort.InitCom: SetCommState Error. ');
      CloseCom;
    end;
end;

procedure TComPort.CloseCom;
begin
  if hFile <> INVALID_HANDLE_VALUE then
    CloseHandle(hFile);
  hFile := INVALID_HANDLE_VALUE;
end;

function TComPort.ReceiveCom(var Buffer; Size: DWORD): Integer;
var
  Received: DWORD;
begin
  if hFile = INVALID_HANDLE_VALUE then

```

```

    raise Exception.Create('TComPort.ReceiveCom: Ошибка чтения Com порта');
if ReadFile(hFile, Buffer, Size, Received, nil) then
begin
    Result := Received;
end
else
    raise Exception.Create('TComPort.ReceiveCom: Ошибка приема данных: ' +
IntToStr(GetLastError));
end;

function TComPort.SendCom(var Buffer; Size: DWORD): Integer;
var
    Sented: DWORD;
begin
    if hFile = INVALID_HANDLE_VALUE then
        raise Exception.Create('TComPort.SendCom: Ошибка записи в Com порт');
    if WriteFile(hFile, Buffer, Size, Sented, nil) then
        begin
            Result := Sented;
        end
    else
        raise Exception.Create('TComPort.SendCom: Ошибка передачи данных: ' +
IntToStr(GetLastError));
    end;
end;

function TComPort.ClearInputCom: Boolean;
begin
    if hFile = INVALID_HANDLE_VALUE then
        raise Exception.Create('TComPort.ClearInputCom: Ошибка Com порта');
    Result := PurgeComm(hFile, PURGE_RXCLEAR);
end;

end.

//****          End of uComPort          ****//

//*****//
//****          ComClient v1.0 - uBufList          ****//
//*****//

unit uBufList;

interface

uses
    Classes,
    uDefine;

type
    (* Буфер для данных          *)
    tBufList = class(tList)
    protected
        function Get(Index: Integer): tBuffer;
    public
        (* Добавление элемента          *)
        procedure Delete(Index: integer);
        (* Очитска буфера          *)
        procedure Clear();
        (* Добавление элемента*)
        function Add(Item: tBuffer): Integer;
        (* Представление элемента в виде строки *)
        function LineToString(Index: integer): string;

```

```

    (* Индексированный доступ к элементам *)
    property Items[Index: Integer]: tBuffer read Get{ write Put}; default;
end;

implementation

{ tBufList }

function tBufList.Add(Item: tBuffer): Integer;
var
    temp: pBuffer;
    i: integer;
begin
    new(temp);
    FillChar(temp^, BUFSIZE, #0);
    temp^:=Item;
    inherited Add(temp);
end;

procedure tBufList.Clear;
begin
    while Self.Count <> 0 do
        Self.Delete(0);
    inherited Clear;
end;

procedure tBufList.Delete(Index: integer);
var
    temp: pBuffer;
begin
    temp:=inherited Get(Index);
    dispose(temp);
    inherited Delete(Index);
end;

function tBufList.Get(Index: Integer): tBuffer;
var
    temp: pBuffer;
    i: integer;
begin
    temp:=inherited Get(Index);
    FillChar(Result, BUFSIZE, #0);
    Result:=temp^;
end;

function tBufList.LineToString(Index: integer): string;
var
    temp: tBuffer;
    i: integer;
begin
    Result:='';
    SetLength(Result, BUFSIZE);
    temp:=Get(Index);
    for i := 0 to BUFSIZE - 1 do
        Result[i + 1]:=chr(temp[i]);
    end;
end.

//****          End of uBufList          ****//

```



```

//*****
//****          ComClient v1.0 - uSettings          ****//
//*****

unit uSettings;

interface

uses Registry, Windows, SysUtils, uDefine;

type
  (* Класс для работы с реестром *)
  tSettings = class(tRegistry)
  private
    fCom: integer;    // Номер Com-порта
    fSpeed: integer; // Скорость Com-порта
    fServer: string; // Сервер
    fPort: integer;  // Номер Порта
    fName: string;   // Имя клиента
  public
    constructor Create();
    destructor Destroy(); override;
    (* Чтение настроек *)
    procedure ReadSettings;
    (* Сохранение настроек *)
    procedure SaveSettings;
    (* Свойства доступа к полям с настройками *)
    property Com: integer read fCom write fCom;
    property Speed: integer read fSpeed write fSpeed;
    property Server: string read fServer write fServer;
    property Port: integer read fPort write fPort;
    property Name: string read fName write fName;
  end;

implementation

{ tSettings }

constructor tSettings.Create;
begin
  inherited Create;
  fCom:=1;
  fSpeed:=9600;
  fServer:='127.0.0.1';
  fPort:=7531;
  ReadSettings;
  fName:='NAME';
end;

destructor tSettings.Destroy;
begin
  SaveSettings;
  inherited Destroy;
end;

procedure tSettings.ReadSettings;
begin
  try
    RootKey := HKEY_CURRENT_USER;
    OpenKey('\Software\ComClient', false);
    fCom:=ReadInteger('Com');
    fSpeed:=ReadInteger('Speed');
    fServer:=ReadString('Server');
    fPort:=ReadInteger('Port');
  
```

```

        fName:=ReadString('Name');
        CloseKey;
    except
        on E: Exception do Log.Add('tSettings.ReadSettings: ' + E.Message);
    end;
end;

procedure tSettings.SaveSettings;
begin
    try
        RootKey := HKEY_CURRENT_USER;
        OpenKey('\Software\ComClient', true);
        WriteInteger('Com', fCom);
        WriteInteger('Speed', fSpeed);
        WriteString('Server', fServer);
        WriteInteger('Port', fPort);
        WriteString('Name', fName);
        CloseKey;
    except
        on E: Exception do Log.Add('tSettings.SaveSettings: ' + E.Message);
    end;
end;

end.

//****          End of uSettins          ****//

//*****//
//****          ComClient v1.0 - uDefine          ****//
//*****//

unit uDefine;

interface

uses
    uLog;

const
    BUFFSIZE = 1024;

type
    tBuffer = array[0..BUFFSIZE - 1] of Byte;
    pBuffer = ^tBuffer;

var
    Log: tLog;
    Mutex: THandle;

implementation

end.

//****          End of uDefine          ****//

//*****//
//****          ComClient v1.0 - uLog          ****//
//*****//

unit uLog;
```

```

interface

uses
  Classes, Windows, SysUtils;

type
  (* Класс для работы с log-файлом *)
  tLog = class
  private
    log: TStringList; // Список всех сообщений лога
    f: textfile;      // Лог-файл
  public
    constructor Create;
    destructor Destroy; override;
    (* Добавление сообщения в лог *)
    procedure Add(Text: string);
    (* Сохранение лог-файла *)
    procedure SaveToFile(fname: string);
    property GetLog: TStringList read log;
  end;

implementation

{ tLog }

procedure tLog.Add(Text: string);
var
  lt: TSYSTEMTIME;
  temp: string;
begin
  GetLocalTime(lt);
  temp:=Format('%4d/%2d/%2d %2d:%2d:%2d', [lt.wYear, lt.wMonth, lt.wDay,
lt.wHour, lt.wMinute, lt.wSecond]);
  log.Add(
    {IntToStr(lt.wYear) + '/' +
    IntToStr(lt.wMonth) + '/' +
    IntToStr(lt.wDay) + ' ' +
    IntToStr(lt.wHour) + ':' +
    IntToStr(lt.wMinute) + ':' +
    IntToStr(lt.wSecond) + ': '}
    temp + #9 +
    Text
  );
  try
    AssignFile(f, 'log.txt');
    Append(f);
    Writeln(f, log[log.Count - 1]);
    CloseFile(f);
  except
    //on E: Exception do MessageBox(0, PChar(E.Message),
PChar(E.HelpContext), MB_OK or MB_ICONERROR);
  end;
end;

constructor tLog.Create;
begin
  inherited Create;
  log:=TStringList.Create();
  AssignFile(f, 'log.txt');
  Rewrite(f);
  CloseFile(f);
end;

```

```

destructor tLog.Destroy;
begin
  log.Destroy;
  inherited;
end;

procedure tLog.SaveToFile(fname: string);
begin
  log.SaveToFile(fname);
end;

end.

//****          End of uLog          ****//

//*****
//****          ComClient v1.0 - uHelp          ****//
//*****

unit uHelp;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TAboutBox = class(TForm)
    Panell: TPanel;
    ProgramIcon: TImage;
    ProductName: TLabel;
    Version: TLabel;
    Copyright: TLabel;
    OKButton: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AboutBox: TAboutBox;

implementation

{$R *.dfm}

end.

//****          End of uHelp          ****//

```