

## Протокол ScoutOpen

- 1 Описание протокола
  - 1.1 О протоколах предыдущих поколений
  - 1.2 Общая информация
  - 1.3 Порядок передачи
  - 1.4 Формат пакета
  - 1.5 Описание отдельных полей
  - 1.6 Описание типа .NET DateTime.Ticks (long)
- 2 Описание формата поля Data
- 3 Декодирование поля Data (дополнительные данные)
- 4 Пример передачи
- 5 Поддержка актуального протокола ScoutOpen оборудованием и ПО СКАУТ

### Описание протокола

#### О протоколах предыдущих поколений

Протоколы ScoutRx и протокол ScoutRx Extended являются устаревшими. Для корректной работы необходимо обновлять поддержку протокола согласно настоящему описанию, переименовав его название в ScoutOpen.

### Общая информация

Передающая сторона (сервер или мобильный терминал) инициирует исходящее соединение на порт TCP, указанный администратором диспетчерского центра. ID объектов, данные по которым ретранслируются (или подключаются напрямую), должны быть предварительно зарегистрированы на принимающей стороне (сервере). Прием пакета с неизвестным ID не приводит к ошибке, соответственно данные по таким объектам будут потеряны.

### Порядок передачи

1. Передается число пакетов, размерность Int32
2. Передаются все пакеты одним потоком, без разделителей. Возможна передача за один раз пакетов, относящихся к нескольким объектам.
3. Принимающая сторона отвечает 0x55, в противном случае передача повторяется.
4. Возврат к пункту 1.

*Примечание – при передаче нулевого количества пакетов пункт 2 пропускается, принимающая сторона сразу отвечает 0x55.*

## Таймауты

Принимающая сторона должна ответить передающей стороне подтверждением **0x55** не позднее **5 секунд**, после приема пакетов. Если принимающая сторона не успевает отправить подтверждение, то передающая сторона (сервер СКАУТ) разорвет соединение и пошлет пакет заново.

*Рекомендуется обработку принятых сообщений делать асинхронно в отдельном потоке, чтобы процедуру общения с передающей стороной сделать без задержек.*

## Формат пакета

Порядок байт – little-endian

|                   |            |           |                                |
|-------------------|------------|-----------|--------------------------------|
| <b>Serial</b>     | String     | n-байт    | Заводской идентификатор        |
| <b>ProtocolId</b> | Int32      | 4 байта   | Тип оборудования               |
| <b>Datetime</b>   | DateTime   | 8 байт    | Время в формате .NET           |
| <b>Lon</b>        | Float      | 4 байта   | Долгота, градусы               |
| <b>Lat</b>        | Float      | 4 байта   | Широта, градусы                |
| <b>Speed</b>      | Float      | 4 байта   | Скорость, км/ч                 |
| <b>Course</b>     | Int16      | 2 байта   | Курс                           |
| <b>Dig_io</b>     | Int16      | 2 байта   | Дискретные входы: биты [0..15] |
| <b>Adc0</b>       | Int16      | 2 байта   | Значение аналогового канала 1  |
| <b>Adc1</b>       | Int16      | 2 байта   | Значение аналогового канала 2  |
| <b>Stat0</b>      | Int16      | 2 байта   | Статус 0 (см. далее)           |
| <b>Stat1</b>      | Int16      | 2 байта   | Статус 1 (см. далее)           |
| <b>Datalen</b>    | Byte/Int16 | 1/2 байта | Длина поля Data                |
| <b>Data</b>       | String     | N байт    | См. описание                   |

## Описание отдельных полей

**Serial** – заводской идентификатор объекта (IMEI, s/n и др.). Первый байт – длина строки (без учета первого байта).

**ProtocolId** – тип оборудования. При передаче и получении данных следует всегда использовать тип 219 = ScoutOpen.

**Datetime** – время в формате dotnet. Один такт соответствует 100 наносекундам или одной десятимиллионной секунды. Значение данного свойства представляет количество 100-наносекундных интервалов, которые прошли с полночи 00:00:00, 1 января 0001, что соответствует значению DateTime.MinValue.

**Передача строк** - строки передаются как массив байт, первый байт – длина строки.

## Описание типа .NET DateTime.Ticks (long)

Один такт соответствует 100 наносекундам или одной десятимиллионной секунды. В миллисекунде 10 000 тактов. Значение данного свойства представляет количество 100-наносекундных интервалов, которые прошли с полночи 00:00:00, 1 января 0001, что соответствует значению DateTime.MinValue.

```
// ??????? ??????? ? Unix
long unixTimeStamp = (DateTime.Ticks - 621355968000000000) / 100000000;
```

Статусные байты:

|                          |                          |
|--------------------------|--------------------------|
| <b>STAT0, биты 15..0</b> | <b>STAT1, биты 15..0</b> |
| 1..0 – зарезервировано   | 4..0 – число спутников   |

|                                                 |                                                |
|-------------------------------------------------|------------------------------------------------|
| <b>2 – если установлен, то питание основное</b> | 5 – не используется                            |
| 15..3 - зарезервировано                         | <b>7..6 – gps fix (фикс есть, если &gt; 0)</b> |
|                                                 | 15..8 – зарезервировано                        |

### Обязательно указание количества спутников и флага GPS fix!

**Datalen** – данное поле кодируется одним или двумя байтами, в зависимости от длины поля дополнительных данных Data.

Алгоритм определения длины поля Data:

1. Читается байт (назовем его DatalenLo) поля DataLength . Если старший бит установлен в “0”, то прочитанное значение и есть длина поля Data, а поле Datalen было закодированно 1 байтом.
2. Если старший бит установлен был в “1”, то длина поля Data закодированно 2 байтами.

Поэтому читается следующий байт (назовем DatalenHi) поля DataLength, а размер определяется по следующей формуле:

$$\text{size} = (\text{DatalenLo} \& 0x7F) + (\text{DatalenHi} \ll 7)$$

**Data** – поле дополнительных данных. Создается при необходимости передать значение больше 2 аналоговых входов. В случае отсутствия в поле **Datalen** передается 0.

### Описание формата поля Data

Формат поля зависит от типа оборудования. Если отдельно не указано, используется следующий формат поля. Дополнительные данные кодируются в строку в шестнадцатеричном виде (HEX-строка), таким образом каждому байту соответствует два байта строки.

Формат исходной строки:

| Len    | Type1    | Len1   | Data1     | ... | TypeN    | LenN   | DataN     |
|--------|----------|--------|-----------|-----|----------|--------|-----------|
| 1 байт | 1 байт   | 1 байт | Len1 байт | ... | 1 байт   | 1 байт | LenN байт |
| Длина  | Датчик 1 |        |           | ... | Датчик N |        |           |

В поле Len передается полная длина декодированной строки (без учета первого байта).

Поле Type – тип датчика:

- 01 – аналоговый
- 02 – частотный
- 03 – импульсный
- 04 – бинарные данные (в разработке, поддерживается не всеми типами серверов)
- 05 – дискретный (в разработке, поддерживается не всеми типами серверов)
- ... - зарезервировано

### Декодирование поля Data (дополнительные данные)

Дополнительные данные кодируются в строку в шестнадцатеричном виде (HEX строка), каждому

байту данных соответствует два символа строки.

Метод декодирования дополнительных данных для ScoutOpen (тип оборудования ScoutOpen = 219)

```
public List<ExtraDataItem> Parse(string hexString)
{
    var paramNames = new[] { "????????? ???? ", "????????? ???? " };
    if (hexString.Length % 2 != 0)
        hexString = hexString.Insert(0, "0");
    var result = new List<ExtraDataItem>();
    if (hexString.Length < 2) return result;
    var numBytes = Convert.ToByte(hexString.Substring(0, 2), 16);

    // ?????? ???????????? ? ?????????? ??????
    var aNum = 2;
    var cNum = 0;
    var i = 0;
    while (i < numBytes)
    {
        // ?????? ? ??????
        var j = (i + 1) * 2;
        var typeCode = (Convert.ToByte(hexString.Substring(j, 2)));
        var type = (ExtraDataItemType) typeCode;
        var paramName = typeCode < paramNames.Length ? paramNames[typeCode - 1] +
            (type == ExtraDataItemType.AnalogData ? aNum++ : cNum++): "";
        var recordBytes = Convert.ToByte(hexString.Substring(j + 2, 2));
        if (type == ExtraDataItemType.AnalogData || type ==
            ExtraDataItemType.CountData)
        {
            var value = recordBytes == 2 ? Convert.ToUInt16(hexString.Substring(j + 4, 4),
                16) : recordBytes == 4 ? Convert.ToInt32(hexString.Substring(j + 4, : 0);
            result.Add(new ExtraDataItem { Name = paramName, Type = type, Value
                = value });
        }
        else
            return result;
        i += (2 + recordBytes);
    }
    return result;
}
```

## Пример передачи

**Передающая сторона:** 00 00 00 00 //число пакетов – 0

**Принимающая сторона:** 55 // подтверждение приема

**Передающая сторона:** 01 00 00 00 //число пакетов – 1

**Передающая сторона:** 05 33 30 31 31 32 db 00 00 00 00 4b 71 00 e7 bc cd 08 2b 98 0a 42 89 06 66 42  
33 33 97 42 41 01 b2 d0 ac 00 12 00 05 00 ca 00 4a 32 34 30 31 30 34 30 30 30 30 32 32 33 30 31 30  
34 30 30 30 30 30 36 37 30 31 30 34 30 30 30 30 33 30 30 30 31 30 34 30 30 30 30 32 30 30 30  
31 30 34 30 30 30 30 30 30 30 31 30 34 30 30 30 30 30 38 42

**Принимающая сторона:** 55 // подтверждение приема

## Содержание пакета:

05 – длина строки ID, 5 байт

33 30 31 31 32 – ID 30112. 33 – это код символа '3', 30 – '0'. Идентификатор терминала - строка.

db 00 00 00 – тип оборудования 219. Данные передаются младшим байтом вперед (little-endian), 0x000000db = 219 (dec)

00 4b 71 00 e7 bc cd 08 – время в формате .NET 30.03.2011 11:44:46. Аналогично, данные передаются младшим байтом вперед. (конвертер ticks to time)

Вот код для чтения на с#:

```
var ticks = br.ReadInt64();  
pack.DateTime = new DateTime(ticks);
```

2b 98 0a 42 – долгота, 34.648602. Представление числа 34.648602, записанное младшим байтом вперед. (конвертер hex to float)

89 06 66 42 – широта, 57.506382. Представление числа 57.506382, записанное младшим байтом вперед.

33 33 97 42 – скорость, 75.6. Представление числа 75.6, записанное младшим байтом вперед.

41 01 – курс, 321 Данные передаются младшим байтом вперед (little-endian), 0x0141 = 321 (dec)

b2 d0 – состояние дискретных входов, 1101000010110010 (нулевой вход (самый первый) имеет состояние "0")

ac 00 – "аналоговый вход 1", значение 172. Данные передаются младшим байтом вперед (little-endian)

12 00 – "аналоговый вход 2", значение 18. Данные передаются младшим байтом вперед (little-endian)

05 00 – основное питание. Stat0 = 0x0005. В бинарном виде это 101. Бит 2 установлен, т.е. питание – основное.

ca 00 – 10 спутников, GPS fix есть. Stat1 = 0x00ca. В бинарном виде это 11001010. Количество спутников 01010, т.е. 10 (dec). Биты 6..7 – 11, т.е. 3 (dec), что больше 0 следовательно фикс есть.

4a – поле data длиной 74 байта

24 010400000223 010400000067 010400000300 010400000200 010400000000 01040000008B

**Длина строки 36 байт, 6 датчиков, все типа 1, длиной 4 байта.**

**Показания датчика 0 – 547**

**Показания датчика 1 – 103**

**Показания датчика 2 – 768**

**Показания датчика 3 – 512**

**Показания датчика 4 – 0**

**Показания датчика 5 – 139**

**Поддержка актуального протокола ScoutOpen оборудованием и ПО СКАУТ**

**СКАУТ 3.5:** начиная со СКАУТ-Сервер 3.5.18.3

**СКАУТ-Платформа:** начиная с версии 1.1.9.2

**Оборудование МТ-600:** начиная с прошивки 8.7