Протокол обмена программы WorkPROG со скважинным прибором

Настройки порта

Установки порта по умолчанию: 8 бит, 1 стоп, без контроля четности. 125000бод по умолчанию.

Формат сообщений

Сообщения разделяются по паузе в линии как в Modbus RTU. Сообщение должно начинаться и заканчиваться интервалом тишины, длительностью не менее 3,5 символов при данной скорости передачи. Во время передачи сообщения не должно быть пауз длительностью более 1,5 символа. Для скоростей 500000 и более бод интервал тишины 128 мкс. Проверка целостности осуществляется с помощью контрольной суммы CRC (длиной в два байта) как в Modbus RTU.

Максимальная длина запроса от персонального компьютера (ПК) к скважинному прибору (СП) составляет 255 байт.

Первый байт сообщения:

7	6	5	4	3	2	1	0
ADR3	ADR2	ADR1	ADR0	CMD3	CMD2	CMD1	CMD0

*старшие 4 бита - адрес устройства в сети, младшие 4 бита – команда для СП.

Адреса устройств:
1-ГГК-П, гамма-гамма плотностной каротаж;
2-Глубиномер;
3-Инклинометр, ГКИ (инклинометр, совмещенный с ГК);
4-ГК гамма-каротаж;
5-ННК нейтронный каротаж;
6-БК, БКС и т.п., электрический каротаж;
7-Акустический профилемер;
8-ИК, индукционный каротаж;
9-АГК, азимутальный ГК;

Зарезервированные адреса:
15-широковещательный запрос
0-зарезервировано для расширенных адресов устройств

Режимы работы СП

Скважинный прибор, как конечный автомат, может находиться в следующих режимах работы:

```
enum
{
	APP_SET_TIME,
	APP_CLEAR_RAM,
	APP_DELAY,
	APP_WORK,
	APP_IDLE,
};
```

Описание:

Нулевой режим. Режим записи задержки в прибор APP_SET_TIME - 0.

Что происходит:

- обнуление таймера тактирования кадров (выполняется синхронизация с другими модулями);
- установка времени задержки СП. Записывается число кадров (выделено 4 байта, знаковое int32). Передается отрицательное число кадров, по которому выставляется задержка включения СП;
- переход на пониженное энергопотребление, по завершению, идет переход к следующему состоянию APP_CLEAR_RAM.
- при получении нулевого числа кадров, СП должен принять это как команду к выключению и перейти в режим спячки APP_IDLE.

Первый режим. Режим очистки внутренней памяти APP_CLEAR_RAM – 1.

Что происходит:

- идет очистка памяти, по окончании переход к следующему режиму отсчета задержки APP_DELAY.

Второй режим. Режим отсчета задержки APP_DELAY – 2.

Что происходит:

- на каждом такте внутренних часов СП, идет инкремент отрицательного номера кадров (значения, полученного в прибор в режиме APP_SET_TIME). При достижении номера кадра равного нулю идет переход в следующий режим APP_WORK;
- установка указателя записи во внешнюю память в начало;
- переход в рабочий режим: полное энергопотребление и включение периферии;
- начало работы (накопление данных за такт).

Третий режим. Основной режим работы и измерений APP_WORK – 3.

Что происходит:

- начало с 1 (первого) такта.
- обработка накопленных за такт данных, запись их в память. Накопление данных для 2 (второго) такта и т.д. вплоть до прихода команды выключения прибора.

Четвертый режим. Режим спячки APP_IDLE – 4.

Что происходит:

- СП переходит в выключенный режим, отключение периферии.
- Ожидание управляющих команд от ПК.

Команды управления СП

СП должен уметь обрабатывать следующие управляющие команды от ПК:

```
// здесь задан адрес устройства в сети
#define ADDRESS (ADDRESS_PROCESSOR << 4)

// 0xF5 установка времени задержки, если получили 0 – то это команда выключения прибора
#define CMD_TIME_SYNC 0xF5

// 0xFA внешняя синхронизация внутренних часов СП во время работы
#define CMD_BEACON 0xFA

// 0xFD режим повышенной скорости приема-передачи данных по UART
```

```
#define CMD_TURBO 0xFD

// 0x01 чтение флэш-памяти
#define CMD_ERAM 0x01 | ADDRESS

// 0x02 чтение информации об устройстве. Так называемые, «метаданные». Описание далее
#define CMD_INFO 0x02 | ADDRESS

// 0x0E обработка ошибок СП
#define CMD_ERR 0x0E | ADDRESS

// 0x05 чтение EEPROM памяти (в EEPROM храним калибровки и т.д.)
#define CMD_READ_EE 0x05 | ADDRESS

// 0x06 запись EEPROM памяти
#define CMD_WRITE_EE 0x06 | ADDRESS

// 0x07 Опрос текущего статуса и данных СП
#define CMD_WORK 0x07 | ADDRESS

// 0x08 bootloader. Для прошивки программного обеспечения микроконтроллера.
#define CMD_BOOT 0x08 | ADDRESS
```

Описание команд

-Команда 0xF5 CMD_TIME_SYNC.

Широковещательная команда без ответа. Установка времени задержки или выключение прибора.

	0	1	2	3	4	5	6
ПК → СП	0xF5	int32L	int32L1	int32H1	int32H	CRCL	CRCH

Тип данных - знаковый int32.

Время задержки передается отрицательным числом кадров. Например, -100, что означает 100 кадров (длительность одного кадра 2.097 сек, по умолчанию) до момента включения СП.

Если получено значение 0, СП должен воспринять это как команду выключения и перейти из любого состояния в режим спячки APP_IDLE.

-установка времени задержки;
-стирание основной флэш-памяти;
-обнуление таймера тактирования кадров;
-если прибор в состоянии больше > APP_DELAY переход в состояние APP_SET_TIME, пониженное энергопотребление.

-Команда 0xFA CMD_BEACON.

Синхронизация времени во время работы в скважине.
Широковещательная команда без ответа. Используется для единой синхронизации всех работающих на линии приборов.

	0	1	2	3	4	5	6
ПК → СП	0xFA	int32L	int32L1	int32H1	int32H	CRCL	CRCH

-Команда 0xFD CMD_TURBO.

Режим повышенной скорости включается при считывании памяти.
Широковещательная команда без ответа.

Если нет сообщений от СП 4-5 кадров, то переход на скорость по умолчанию 125000 бод.

	0	1	2	3
ПК → СП	0xFD	Speed	CRCL	CRCH

*бит Speed – скорость передачи: 1-0.5M, 2-1M, 3-2.25M, 4-4.5M.

-Команда 0x01 CMD_ERAM

Команда чтения памяти. Передается стартовый адрес ячейки памяти и требуемое количество байт из флэш-памяти. Остановка чтения происходит при получении пустого пакета 0xFF. Общий размер памяти, программа WorkProg считывает из метаданных СП.

направ.	0	1...4	5..8	9	10
ПК → СП	0xADR|1	Адрес начала uint32 (4байта)	Длина N uint32 (4байта)	CRCL	CRCH

направ.	0	1..N	N+1	N+2
СП → ПК	0xADR|1	Данные N байт с заданной ячейки флэш-памяти	CRCL	CRCH

-Команда 0x02 CMD_INFO

Команда чтения информации об устройстве.

Вся информация о приборе, о его типе, используемых измерительных каналах, формате используемых переменных хранится в массиве данных, называемые «метаданные». Метаданные генерируется отдельно (см. описание ниже), при помощи специальной утилиты и хранятся в памяти СП в виде массива чисел. Размер метаданных зависит от типа СП. По полученным метаданным программа WorkProg автоматически получает всю информацию о подключенном приборе.

Первый запрос

направ.	0	1	2	3
ПК → СП	0xADR|2	Длина N	CRCL	CRCH
		*требуемое количество байт с массива метаданных. Например, при N=0x03, СП обязан выдать первые 3 байта из массива метаданных, которые информируют WorkProg о размере массива метаданных.		

Второй запрос

направ.	0	1	2	3	4	5
ПК → СП	0xADR|2	Длина N	Адрес начала uint16 (2 байта)		CRCL	CRCH

Ответ СП

направ.	0	N	N+1	N+2
СП → ПК	0xADR|2	Данные N байт	CRCL	CRCH

-Команда 0x05 CMD_READ_EE

Чтение EEPROM устройства. Информацию о размере записанных данных в EEPROM WorkProg считывает с метаданных СП.

направ.	0	1	2	3	4	5
ПК → СП	0xADR|5	Адрес начала uint16 (2 байта)		Длина N	CRCL	CRCH

направ.	0	N	N+1	N+2
СП → ПК	0xADR|5	Данные N байт	CRCL	CRCH

-Команда 0x06 CMD_WRITE_EE

Запись в EEPROM устройства. Используется для записи настроек и калибровок в прибор.

направ.	0	1	2	3..(N+2)	N+3	N+4
ПК → СП	0xADR|6	Адрес начала uint16 (2 байта)		Данные N байт	CRCL	CRCH

Ответ СП:

направ.	0	1	2
СП → ПК	0xADR|6	CRCL	CRCH

-Команда 0x07 CMD_WORK

Команда выполняет опрос статуса и замеренных данных с СП. По заданному протоколу обмена с программой WorkProg, формат выдачи кадра должен быть следующим:

- Первым выдается регистр статуса СП `uint8_t AppState;` // 1 байт

Регистр статуса AppState /// автомат|AU

7bit	6bit	5	4	3	2bit	1bit	0bit
FL_PWR	FL_Error	N/A			MODE_SP		
Флаг питания СП 0 – питание отключено; 1 – питание включено.	Флаг ошибок СП Если во время работы СП были неисправности оборудования или запредельные режимы, то устанавливается данный флаг. Флаг сбрасывается командой 0x0E, которая читает сообщения об ошибках.	не используется			Текущий режим работы СП 0 - APP_SET_TIME; 1 - APP_CLEAR_RAM; 2 - APP_DELAY; 3 - APP_WORK; 4 - APP_IDLE.		

- Вторым выдается регистр текущего времени (кадра) `int32_t time;` // 4 байта

Регистр time

1 Byte	2 Byte	3 Byte	4 Byte
int32L	int32L1	int32H1	int32H

- Последующие данные – измеренные каналы СП (тип данных и последовательность задана в метаданных)

- Контрольная сумма CRC.

Команда CMD_WORK 0x07 может запрашивать только первые два параметра: состояние конечного автомата AppState и время time, в этом случае СП переходит или остается в режиме пониженного энергопотребления (Внимание: это не относится к режиму работы APP_WORK!). Данный режим удобен для контроля состояния СП, например, в момент отсчета задержки.

Если команда CMD_WORK 0x07 запрашивает все данные структуры WRK, то прибор переходит в режим информации, бит питания FL_PWR устанавливается. Данный режим позволяет проверить работоспособность СП в реальном времени. Если запросы не приходят 4-5 кадров, то прибор переходит в спящий режим (пониженного энергопотребления) (Внимание: это не относится к режиму работы APP_WORK!).

При передаваемом кадре меньше 255 байт (задано в метаданных)

направ.	0	1	2	3
ПК → СП	0xADR|7	Длина N	CRCL	CRCH

При передаваемом кадре больше 255 байт (задано в метаданных)

направ.	0	1, 2	3	4
ПК → СП	0xADR|7	Длина N uint16 (2байта)	CRCL	CRCH

Ответ СП:

направ.	0	N	N+1	N+2
СП → ПК	0xADR|7	Данные N байт	CRCL	CRCH

-Команда 0x0E CMD_ERR

Обработка ошибок СП

направ.	0	1	2	3
ПК → СП	0xADR|E	NeedClearErr	CRCL	CRCH

*Если NeedClearErr = 0xA5, то сбрасывается флаг ошибки (очищается буфер ошибок).

направ.	0	1	2	3	4
СП → ПК	0xADR|E	Номер ошибки	Строка описания	CRCL	CRCH

После каждого запроса, выдается очередная ошибка из буфера ошибок

Создание метаданных СП

Создание метаданных об устройстве. В метаданных хранится полная информация о СП (каналы, форматы и тип данных, размер памяти и тд).

Описание замеренных данных, которые выдает прибор, данные записываемые в память в режиме работы, данные хранящиеся в EEPROM – всё это находятся в отдельном хедер файле (Data.h).

Пример Data.h

```
#pragma  once

namespace adxl354gk
{

// описываем свои переменные
typedef struct __attribute__((packed))
{
	uint16_t gk; ///гк
} Gk_t ;


// описывается используемые переменные в EEPROM
typedef struct __attribute__((packed))
{
	Gk_t GR1; /// ГК|GK1
} EepData_t;


// описываем свои переменные
typedef struct __attribute__((packed))
{
	int16_t X;
	int16_t Y;
	int16_t Z;
} Dat_t;

// описываем свои переменные
typedef struct __attribute__((packed))
{
	Dat_t accel;
	Dat_t magnit;
	int16_t T;
```

```
        float Zenit;        /// зенит
        float Azimut;       /// азимут
        float Gtf;          /// отклонитель
        float Mtf;          /// маг_отклон
        int16_t Gtot;       /// амплит_accel
        int16_t Mtot;       /// амплит_magnit
} InclW_t;


// описываем свои переменные
typedef struct __attribute__((packed))
{
        Dat_t accel;
        Dat_t magnit;
        int16_t T;
} InclR_t_old;


// переменные, отображаемые по команде CMD_WORK 0x07.
typedef  struct __attribute__((packed))
{
        uint8_t AppState;  ///  автомат|AU
        int32_t time;      ///  время|WT
        InclW_t dat;       ///  Inclin|INKLGK
        Gk_t gk;           ///  ГК|GK1
} WorkData_t;


// Информация о размере памяти и формате записи каналов.
typedef struct __attribute__((packed))
{
#       define RAM_SIZE 32        /// varRamSize
        int32_t ramtime;           /// время|WT
        InclW_t dat;               /// Inclin|INKLGK
        Gk_t gk;                   /// ГК|GK1
} RamData_t;


// Описываем всю структуру. Внимание тут не изменять ключевые слова.
typedef struct __attribute__((packed))
{
#       define ADDRESS_PROC 3                       /// var_adr
#       define DEV_INFO  "__DATE__ ADXL354 GK"      /// var_info
#       define CHIP_NUMBER 4                        /// varChip
#       define SERIAL_NUMBER 1                      /// varSerial
#       define UART_SPEED_MASK 192                  /// varSupportUartSpeed
    WorkData_t Wrk;        /// WRK
    RamData_t Ram;         /// RAM
    EepData_t Eep;         /// EEP
} AllDataStruct_t;         /// InclGK1
}
```

Правила используемые при формировании Data.h:

1) Структуры должны быть упакованными. Объявление структур распознается только в таком виде:

```
typedef struct
{
……
} RamDat……
```

2) Распознаются только десятичные числа для пользовательских типов данных. Для var_info строка __DATE__ заменяется на текущую дату.

3) Информация после трех слешев /// распознается следующим образом:

- Для переменных:

```
int32_t time;    ///  время|WT
```

time – имя переменной в программе микроконтроллера
время – фактическое имя в метаданных, желательно латиницей. Отображается на экране в WorkProg.
WT - атрибут для форматированного вывода на WorkProg, только латиницей. Используется во встроенном в WorkProg языке программирования LUA.

-Для структур пример:

```
InclW_t dat;  /// Inclin|ADXL354
```

dat - имя структуры для программы прибора
Inclin – фактическое имя в метаданных, тип устройства для метрологии, желательно латиницей.
ADXL354 - атрибут для форматированного вывода на WorkProg, только латиницей. Используется во встроенном в WorkProg языке программирования LUA.

-Для пользовательских данных пример:

```
#     define ADDRESS_PROC 3                /// var_adr
```

ADDRESS_PROC – имя для программы прибора
3 - значение в метаданных и для программы прибора
var_adr - фактическое имя пользовательского типа данных

Утилита h2meta.exe ищет корневую структуру с именем AllDataStruct_t и по ней генерирует метаданные пример:

```
typedef struct
{
#     define ADDRESS_PROC 3                        /// var_adr
#     define DEV_INFO  "__DATE__ ADXL354 GK"       /// var_info
#     define CHIP_NUMBER 4                         /// varChip
#     define SERIAL_NUMBER 1                       /// varSerial
#     define UART_SPEED_MASK 192                   /// varSupportUartSpeed
   WorkData_t Wrk;         /// WRK
   RamData_t Ram;          /// RAM
   EepData_t Eep;          /// EEP
} AllDataStruct_t __attribute__((packed));         /// InclGK1
```

InclGK1 – имя модели прибора

Где WRK – данные прибора выдаваемые в режиме выдачи тестовой информации
RAM - данные записываемые каждый кадр в память в рабочем режиме.
EEP - данные в EEPROM прибора (если есть)

СОГЛАШЕНИЯ ДЛЯ СТРУКТУРЫ WRK
(данные прибора выдаваемые в режиме выдачи тестовой информации).

```
typedef  struct
{
      uint8_t AppState;   ///  автомат|AU – обязательный параметр
      int32_t time;       ///  время|WT – обязательный параметр
      InclW_t dat;        ///  Inclin|ADXL354 – данные модуля прибора
      Gk_t gk;            ///  ГК|GK1 – данные модуля прибора
} Work
```

СОГЛАШЕНИЯ ДЛЯ СТРУКТУРЫ RAM
(данные записываемые каждый кадр в память в рабочем режиме).

```
typedef struct
{
#define RAM_SIZE 32 /// varRamSize или varSSDSize - обязательный параметр объем памяти
int32_t ramtime;    /// время|WT - обязательный параметр номер кадра
InclW_t dat;        /// Inclin|ADXL354 - данные модуля прибора
Gk_t gk;            /// ГК|GK1 - данные модуля прибора
} Ram…
```

Для получения конечного массив метаданных используется утилита h2meta.exe

Данная утилита понимает кодировку ANSI, пытается распознать UTF-8, и преобразовать в ANSI, может распознавать следующие типы данных, массивы этих типов:

```
int8_t
uint8_t
int16_t
uint16_t
int32_t
uint32_t
int64_t
uint64_t
float
double
```

Ключевые слова и пользовательские типы данных:

```
var_info - uint8[]       - текстовая информация о приборе
var_adr - uint8          - адрес прибора
varChip - uint8          - тип процессора для бутлоадера
varSerial - uint16       - серийный номер прибора
varRamSize – uint16      - память в мегабайтах
varSSDSize – uint32      - память в секторах 512 байт для SD карты
varSupportUartSpeed uint16 -битовая маска поддерживаемых устройством протоколов
                           (скорости uart) обмена данными:
                           type ESpeed = (S125K = $80, S500K = $40, S1M = $20, S2_25M =
                           $10, S4_5M = $08, SD = $4000, USB = $8000);
```

Есть два варианта получения метаданных.

1) При использовании компилятора GCC, файл Data.h перед компиляцией преобразуется с помощью утилиты h2meta (пример: c:\AVR\h2meta.exe Data.h MetaData.h) в следующий хедер файл (MetaData.h):

```
#pragma once

const unsigned char __attribute__ ((section(".meta_data"), used)) cmetaAll[] = {
36,138,1,73,110,99,108,51,0,40,3,39,50,53,46,48,57,46,50,48,49,57,32,65,68,88,76,
51,53,52,32,71,75,0,56,4,57,1,0,62,192,0,36,170,0,87,82,75,0,17,224,226,242,238,236,
224,242,124,65,85,0,3,226,240,229,236,255,124,87,84,0,36,127,0,73,110,99,108,105,
110,124,65,68,88,76,51,53,52,0,36,18,0,97,99,99,101,108,0,2,88,0,2,89,0,2,90,0,36,
19,0,109,97,103,110,105,116,0,2,88,0,2,89,0,2,90,0,2,84,0,4,231,229,237,232,242,0,
4,224,231,232,236,243,242,0,4,238,242,234,235,238,237,232,242,229,235,252,0,4,236,
224,227,95,238,242,234,235,238,237,0,2,224,236,239,235,232,242,95,97,99,99,101,108,
0,2,224,236,239,235,232,242,95,109,97,103,110,105,116,0,36,14,0,195,202,124,71,75,
49,0,18,227,234,0,36,161,0,82,65,77,0,43,10,0,3,226,240,229,236,255,124,87,84,0,36,
127,0,73,110,99,108,105,110,124,65,68,88,76,51,53,52,0,36,18,0,97,99,99,101,108,0,
2,88,0,2,89,0,2,90,0,36,19,0,109,97,103,110,105,116,0,2,88,0,2,89,0,2,90,0,2,84,0,
4,231,229,237,232,242,0,4,224,231,232,236,243,242,0,4,238,242,234,235,238,237,232,
242,229,235,252,0,4,236,224,227,95,238,242,234,235,238,237,0,2,224,236,239,235,232,
242,95,97,99,99,101,108,0,2,224,236,239,235,232,242,95,109,97,103,110,105,116,0,36,
14,0,195,202,124,71,75,49,0,18,227,234,0,36,21,0,69,69,80,0,36,14,0,195,202,124,71,
75,49,0,18,227,234,0};
```

2) При использовании сторонних компиляторов, отличных от AVR процессоров, особых настроек проекта и т.д., для генерации файла MetaData.h используем стандартизированный шаблон, например: c:\AVR\h2meta.exe Data.h MetaData.h Patern.h

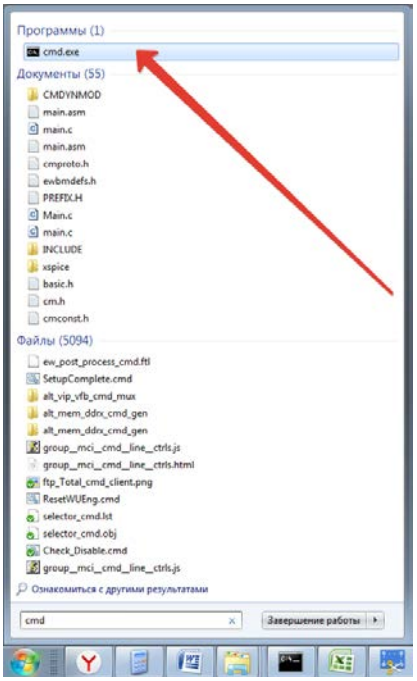
Содержимое файла Patern.h:

```
#pragma once

extern uint8_t ReadMetaData(uint8_t* p, uint8_t n, uint16_t from);

const unsigned char __attribute__ ((section(".meta_data"), used)) cmetaAll[] = {
%s};
```

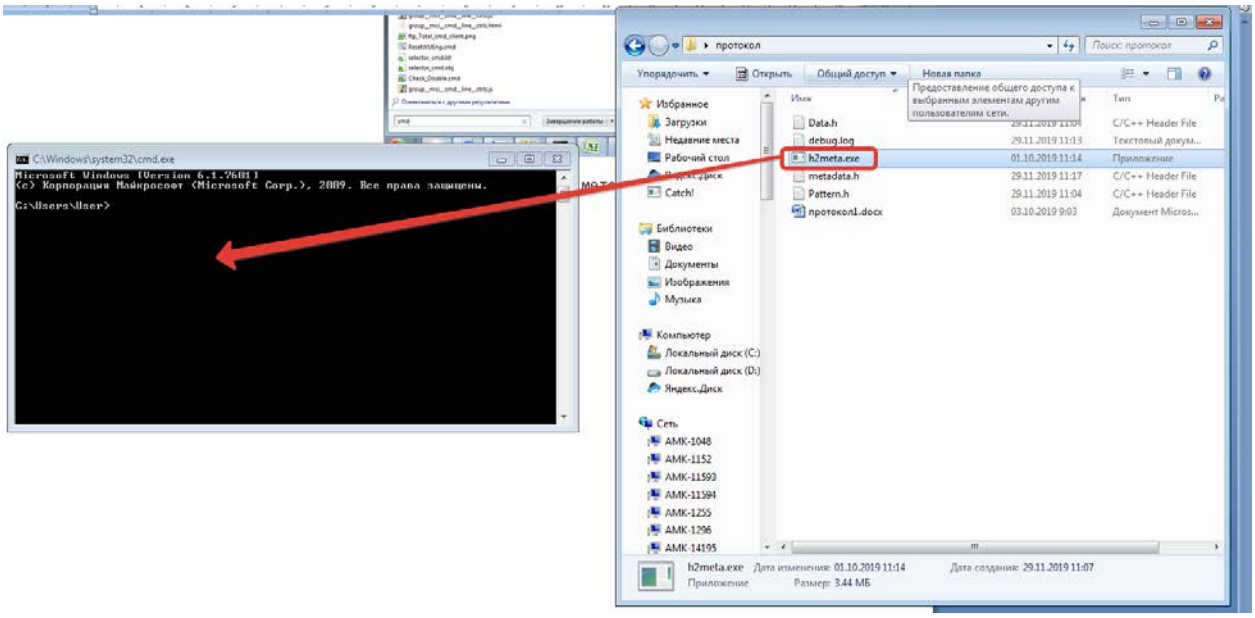
Вместо %s утилитой автоматически подставляется сгенерированный массив метаданных и создается файл MetaData.h.

Пошаговая инструкция:

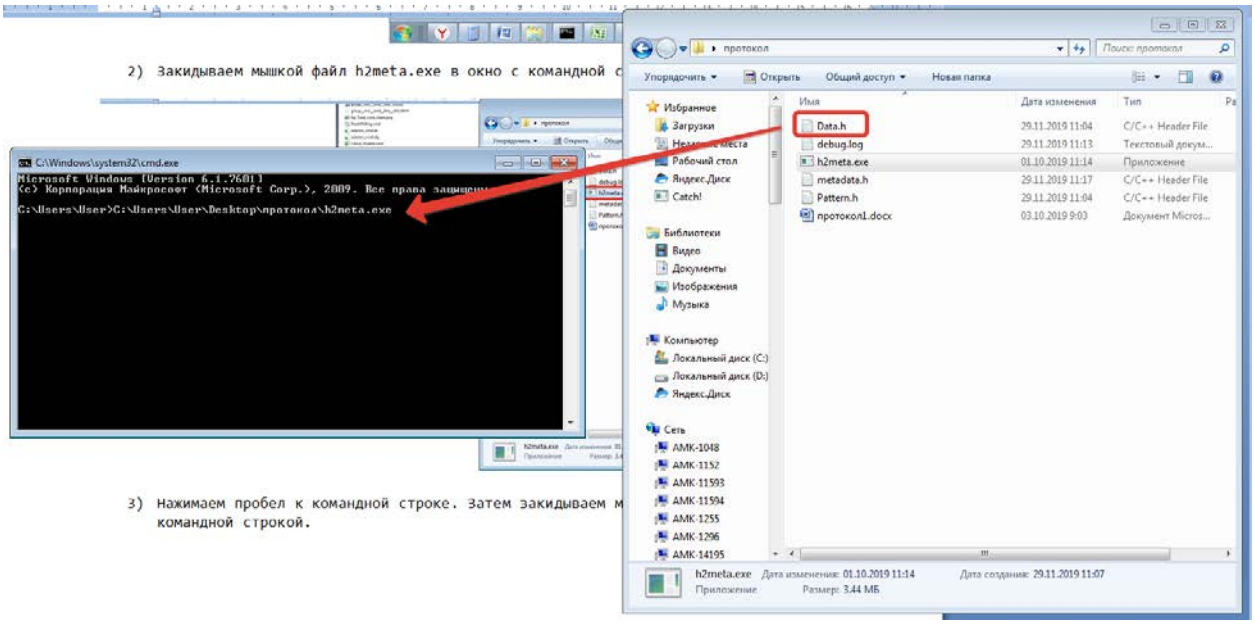
1) Запускаем командую строку. ПУСК → cmd.exe



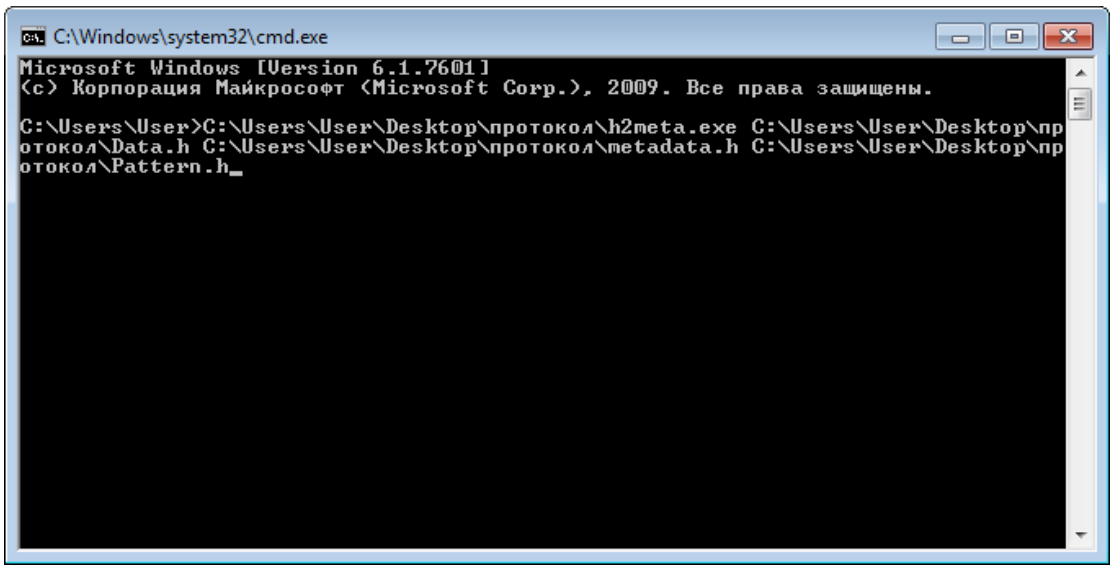
2) Закидываем мышкой файл h2meta.exe в окно с командной строкой методом drag-and-drop



3) Нажимаем пробел в командной строке. Затем закидываем мышкой файл Data.h в окно с командной строкой.



4) Опять пробел в командой строке, потом аналогично закидываем metadata.h (пока пустой) и Pattern.h. В итоге должно получится вот так:



5) После нажимаем Enter, и утилита генерирует metadata.h с нужным нам массивом метаданным. Финиш.