

# Руководство пользователя Gyrovert C++ API (Low Level)

Библиотека взаимодействия с инерциальными модулями серий ГКВ и  
МГ

ООО «Лаборатория микроприборов», 2022

## Содержание:

<b>1. Введение. Работа устройства.....</b>	<b>3</b>
<b>2. Структуры пакетов данных. ....</b>	<b>4</b>
<b>3. Методы класса LMP_Device .....</b>	<b>8</b>
<b>3.1. Настройка отправки запросов/команд и обработка принятых данных. ....</b>	<b>8</b>
<b>3.2. Запросы.....</b>	<b>9</b>
<b>3.3. Команды .....</b>	<b>11</b>
<b>3.4. Установка пользовательских callback-функций .....</b>	<b>16</b>
<b>3.5. Специальные команды для настройки параметров интерфейсов (IfProto) .....</b>	<b>23</b>
<b>4. Примеры.....</b>	<b>28</b>
<b>4.1. Консольный пример вывода принимаемых данных с использованием WinAPI (Windows) и библиотеки termios (Linux) .....</b>	<b>28</b>
<b>4.2. Консольный пример установки алгоритма «Данные с датчиков» и вывода принимаемых данных с использованием WinAPI (Windows) и библиотеки termios (Linux) .....</b>	<b>28</b>
<b>4.3. Консольный пример выбора алгоритма пользователем и вывода принимаемых данных с использованием WinAPI (Windows) и библиотеки termios (Linux).....</b>	<b>28</b>
<b>4.4. Пример наследования класса LMP_Device для расширения функционала библиотеки ....</b>	<b>29</b>
<b>4.5. Консольный пример записи данных в бинарный файл с использованием WinAPI (Windows) и библиотеки termios (Linux) .....</b>	<b>29</b>

## **1. Введение. Работа устройства.**

Инерциальные модули серий ГКВ и МГ предназначены для решения задач навигации и ориентации.

Взаимодействие с устройствами осуществляется через последовательный порт по интерфейсу RS-422 с использованием протокола «Gyrovert» ООО «Лаборатория микроприборов».

Библиотека Gyrovert C++ LL предоставляет класс LMP\_Device с набором методов, с помощью которых пользователь может организовать взаимодействие с устройством в программе, написанной на языке C++.

Библиотека не работает напрямую с serial-портом, для задания функции прямой отправки данных на serial-порт используется метод «[SetSendDataFunction\(\)](#)».

Выбранная функция будет использоваться для отправки всех запросов и команд из разделов 3.2 и 3.3. Пример использования представлен в разделе 4.

Обработка принятых данных и распознавание пакетов происходит внутри метода «[ReceiveProcess\(\)](#)». Пример использования представлен в разделе 4.

Библиотека предоставляет возможность задания пользовательских callback-функций, которые будут вызываться при приёме пакета определенного типа. Список методов представлен в разделе 3.4. Примеры представлены в разделе 4.

## 2. Структуры пакетов данных.

Подробное описание протокола информационного взаимодействия приведено в документе «Протокол информационного взаимодействия» ЛМАП.402131.009Д1.

Ниже представлены структуры пакетов, используемых во взаимодействии с устройством серии ГКВ или МГ.

Общая структура всех пакетов. Размер поля data может составлять от 4 до 259 байт и включает в себя одну из нижеперечисленных структур и 4 байта контрольной суммы CRC32.

```
typedef struct __GKV_PacketBase
{
    uint8_t preamble;           /* always 255 */
    uint8_t address;            /* address of sending device */
    uint8_t type;               /* type of the packet */
    uint8_t length;             /* length of data fields (without checksum) */
    uint8_t data[GKV_DATA_LENGTH + 4]; /* all data that packet's containing including checksum */
} GKV_PacketBase;
```

Нижеперечисленные структуры данных при отправке пакета записываются поле data и составляют величину GKV\_DATA\_LENGTH.

Структура поля data (без CRC32) пакета ID устройства (поле type=0x05)

```
typedef struct __GKV_ID
{
    uint16_t bootloader_version; /* version of embended software */
    uint16_t firmware_version;   /* version of changeable software */
    uint32_t production_date;    /* date of device manufacturing */
    char serial_id[16];          /* serial number of device */
    char description[16];        /* device code in ASCII */
    uint8_t mode;                /* used in the manufacturing phase */
    uint16_t status;             /* field for detecting errors, sync and algorithm state */
} GKV_ID;
```

Структура поля data (без CRC32) пакета настроек устройства (поле type=0x07) (для записи параметров используются маски “mode\_mask” и “param\_mask”).

```
typedef struct __GKV_Settings
{
    uint32_t mode_mask;           /* field to allow change data format parameters */
    uint32_t mode;                /* field to change data format parameters when it is allowed */
    uint32_t param_mask;         /* field to allow change settings of data processing and sending */
    uint8_t uart_baud_rate;       /*baudrate of main RS-485 */
    uint8_t uart_address;        /* address of device */
    uint16_t rate_prescaler;      /* basic freq of data packets for GKV = 1000 Hz */
    uint8_t algorithm;           /* type of sensors and GNSS data processing */
    uint8_t gyro_range;
    uint8_t acc_range;
    uint16_t sync_out_prescaler; /*changes the speed of data output by averaging */
    float dcm[9];                /* rotation matrix (3x3) of the measured data */
    uint8_t aux_485_type;         /* when external device is connected, user can receive/process ext. data*/
    uint8_t data_out_skip;        /* number of skipping output packets to reduce frequency */
    uint8_t aux_485_baudrate;     /* baudrate of additional RS-485 from 9600 bit/s to 3 MBit/s */
    uint8_t magnetometer_range;  /* range of three axis magnetic sensor */
    uint8_t ext_sync_mode;        /* type of external sync signal (toggle or pulse) */
}GKV_Settings;
```

Структура поля data (без CRC32) пакета данных с датчиков в виде кодов АЦП (поле type=0x0A)

```
typedef struct __GKV_ADCData
{
    uint16_t sample_cnt;         /* 0-65535 counter to detect number of lost packets */
    uint16_t status;             /* field for detecting errors, sync and algorithm state */
    int32_t a[3];                /* accelerometer non-calibrated X Y Z axis data in 24 bit ADC codes */
    int32_t w[3];                /* rate sensor non-calibrated X Y Z axis data in 24 bit ADC codes */
    int32_t t[4];                /* X Y Z axis and CPU temperature data in 12 bit ADC*/
}GKV_ADCData;
```

Структура поля data (без CRC32) пакета углов ориентации (поле type=0x0C)

```
typedef struct __GKV_GyrovertData
{
    uint16_t sample_cnt;         /* 0-65535 counter to detect number of lost packets */
    uint16_t status;             /* field for detecting errors, sync and algorithm state */
    float pitch;                 /* pitch Euler angle */
    float roll;                  /* roll Euler angle */
    float yaw;                   /* yaw Euler angle */
}GKV_GyrovertData;
```

Структура поля data (без CRC32) пакета углов инклинометра (поле type=0x0D)

```
typedef struct __GKV_InclinometerData
{
    uint16_t sample_cnt;         /* 0-65535 counter to detect number of lost packets */
    uint16_t status;             /* field for detecting errors, sync and algorithm state */
    float alfa;                  /* inclinometer angle alfa (XZ) */
    float beta;                  /* inclinometer angle beta (YZ) */
}GKV_InclinometerData;
```

Структура поля data (без CRC32) пакета навигационных данных (поле type=0x12)

```
typedef struct __GKV_BINSDData
{
    uint16_t sample_cnt; /* 0-65535 counter to detect number of lost packets */
    uint16_t status; /* field for detecting errors, sync and algorithm state */
    float x; /* x axis position */
    float y; /* y axis position */
    float z; /* z axis position */
    float pitch; /* pitch Euler angle */
    float roll; /* roll Euler angle */
    float yaw; /* yaw Euler angle */
    float alfa; /* inclinometer angle alfa XZ */
    float beta; /* inclinometer angle beta YZ */
    float q[4]; /* orientation quaternion q3 q2 q1 q0 */
}GKV_BINSDData;
```

### Структура поля data (без CRC32) пакета данных ГНСС (поле type=0x0E)

```
typedef struct __GKV_GpsData
{
    uint32_t time; /* Coordinated Universal Time (UTC)*/
    double latitude; /* latitude from GNSS */
    double longitude; /* longitude from GNSS */
    double altitude; /* altitude from GNSS */
    uint32_t state_status; /* state of GNSS receiver */
    float TDOP; /* geometry factor of GNSS receiver */
    float HDOP; /* geometry factor of GNSS receiver */
    float VDOP; /* geometry factor of GNSS receiver */
    float velocity; /* horizontal speed */
    float yaw; /* azimuth angle from GNSS */
    float alt_velocity; /* vertical speed */
}GKV_GpsData;
```

### Структура поля data (без CRC32) расширенного пакета данных ГНСС (type=0x0F)

```
typedef struct __GKV_GpsDataExt
{
    double vlat; /* velocity on latitude */
    double vlon; /* velocity on longitude */
    float sig_lat; /* STD of latitude data */
    float sig_lon; /* STD of longitude data */
    float sig_alt; /* STD of altitude data */
    float sig_vlat; /* STD of velocity on latitude */
    float sig_vlon; /* STD of velocity on longitude */
    float sig_valt; /* STD of velocity on altitude */
    uint16_t num_ss; /* number of satellites used in calculation of GNSS data*/
    uint16_t reserved;
}GKV_GpsDataExt;
```

### Структура поля data (без CRC32) наборного пакета данных (поле type=0x13)

```
typedef struct __GKV_CustomData
{
    float parameter[63]; /* value of parameter N from list of 'custom_data_parameters' */
}GKV_CustomData;
```

### Структура поля data пакета параметров наборного пакета (поле type=0x27)

```
typedef struct __GKV_CustomDataParam
{
    uint8_t num; /* number of parameters that device sends in custom packet */
    uint8_t param[63]; /* type of parameter N from list of 'custom_data_parameters' */
}GKV_CustomDataParam;
```

Пакеты запросов всегда имеют величину GKV\_DATA\_LENGTH=0, отличаются только значения поля type.

Описание запроса	#define для заполнения поля type	Значение поля type
Проверка соединения (со стороны пользователя) /подтверждение приёма (со стороны устройства):	GKV_CHECK_PACKET GKV_CONFIRM_PACKET	0x00
Сброс устройства:	GKV_RESET_PACKET	0x01
Запрос ID:	GKV_DEV_ID_REQUEST	0x04
Запрос настроек:	GKV_DEV_SETTINGS_REQUEST	0x06
Запрос данных (в режиме работы устройства «По запросу»)	GKV_DATA_REQUEST	0x17
Запрос списка текущих параметров наборного пакета:	GKV_CUSTOM_PACKET_PARAM_REQUEST	0x26

### 3. Методы класса LMP\_Device

#### 3.1. Настройка отправки запросов/команд и обработка принятых данных.

Метод выбора функции отправки запроса/команды по serial-порту:

```
void SetSendDataFunction(std::function<void(GKV_PacketBase *)>ptrSendPacketFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию отправки массива данных через serial-порт. Эта функция должна иметь вид:

```
void UserSendFunction(GKV_PacketBase *ptrPacket);
```

Здесь ptrPacket – это указатель на объект команды, который будет отправлен на устройство. Список запросов и команд перечислен в разделах 3.2 и 3.3

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetSendDataFunction(std::bind(&UserClass::UserSendFunction, &UserClassObj, std::placeholders::_1));
```

Метод обработки принятых данных:

```
void Receive_Process(uint8_t *buffer_ptr, uint16_t buffer_size);
```

В данном методе работает основной цикл обработки данных.

Метод принимает в качестве аргументов:

- указатель на массив данных, принятых по serial-порту.
- размер массива

Если среди принятых данных обнаружен корректный пакет, вызывается пользовательская callback-функция, в которой может быть произведена обработка принятого пакета. Установить callback-функции можно с помощью сеттеров, перечисленных в разделе 3.4.



## 3.2. Запросы.

**ВНИМАНИЕ:** Для корректной работы всех запросов необходимо до начала взаимодействия задать функцию отправки данных с помощью метода [SetSendDataFunction\(\)](#)» (см. раздел 3.1)

Проверка соединения:

**void CheckConnection ();**

Данный метод отправляет тестовый пакет для проверки соединения с устройством. Ответом будет пакет подтверждения получения запроса. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

Запрос ID:

**void RequestDeviceID();**

Данный метод отправляет запрос ID устройства. Структура ответа на данный запрос представлена в разделе 2. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetIDReceivedCallback();**

Метод описан в разделе 3.4

Запрос настроек устройства:

**void RequestSettings();**

Данный метод отправляет запрос пакета текущих настроек устройства. Структура ответа на данный запрос представлена в разделе 2. Пользовательскую callback-функцию, вызываемую при получении ответа на данный запрос можно задать с помощью метода:

**void SetSettingsReceivedCallback ();**

Метод описан в разделе 3.4

Запрос параметров наборного пакета данных:

**void RequestCustomPacketParams();**

Данный метод отправляет запрос текущих параметров наборного пакета. Структура ответа на данный запрос представлена в разделе 2. Пользовательскую callback-функцию, вызываемую при получении ответа на данный запрос можно задать с помощью метода:

**void SetCustomPacketParamReceivedCallback();**

Метод описан в разделе 3.4

Запрос данных:

**void RequestData();**

Данный метод отправляет запрос данных, если на устройстве установлен режим выдачи данных «По запросу». Ответом на данный запрос будет пакет данных, установленный на данный момент (стандартный пакет алгоритма или наборный пакет данных). Пользовательскую callback-функцию, вызываемую при получении ответа на данный запрос можно задать с помощью методов:

**void SetADCDDataReceivedCallback();** - для стандартного пакета алгоритма работы устройства «Коды АЦП»

**void SetRawDataReceivedCallback();** - для стандартного пакета алгоритма работы устройства «Данные с датчиков»

**void SetGyrovertDataReceivedCallback();** - для стандартного пакета алгоритмов работы устройства «Ориентация фильтр Калмана»/«Ориентация фильтр Махони»

**void SetInclinometerDataReceivedCallback();** - для стандартного пакета алгоритма работы устройства «Инклинометр»

**void SetBINSDataReceivedCallback();** - для стандартного пакета алгоритма работы устройства «Навигация»

**void SetCustomPacketParamReceivedCallback();** – для наборного пакета данных в любом из вышеперечисленных алгоритмов.

Методы описаны в разделе 3.4

### 3.3. Команды

**ВНИМАНИЕ:** Для корректной работы всех команд необходимо до старта взаимодействия задать функцию отправки данных с помощью метода `SetSendDataFunction()`» (см. раздел 3.1)

Установка алгоритма:

**`void SetAlgorithm(uint8_t algorithm_register_value);`**

Данный метод отправляет на устройство команду установки алгоритма в соответствии со значением, указанным в поле “algorithm\_register\_value”. Поле может принимать значения:

**`GKV_ADC_CODES_ALGORITHM`** – алгоритм «Коды АЦП»

**`GKV_SENSORS_DATA_ALGORITHM`** – алгоритм «Данные с датчиков»

**`GKV_ORIENTATION_KALMAN_ALGORITHM`** – алгоритм «Ориентация, фильтр Калмана»

**`GKV_INCLINOMETER_ALGORITHM`** – алгоритм «Инклинометр»

**`GKV_ORIENTATION_MAHONY_ALGORITHM`** – алгоритм «Ориентация, фильтр Mahony»

**`GKV_BINS_NAVIGATON_ALGORITHM`** – алгоритм «Навигация»

**`GKV_CUSTOM_ALGORITHM`** – пользовательский алгоритм (может быть разработан для конкретной задачи по запросу)

Ответом будет пакет подтверждения обработки команды. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**`void SetConfirmPacketReceivedCallback();`**

Метод описан в разделе 3.4

Выбор формата выдачи данных в виде стандартного пакета алгоритма:

**`void SetDefaultAlgorithmPacket();`**

Данный метод отправляет на устройство команду установки формата выдачи данных в виде стандартного пакета выбранного алгоритма.

Ответом будет пакет подтверждения обработки команды. Callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**`void SetConfirmPacketReceivedCallback();`**

Метод описан в разделе 3.4

Выбор формата выдачи данных в виде наборного пакета:

**void SetCustomAlgorithmPacket();**

Данный метод отправляет на устройство команду установки формата выдачи данных в виде наборного пакета данных.

Ответом будет пакет подтверждения обработки команды. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

Установка параметров наборного пакета:

**void SetCustomPacketParam(uint8\_t\* param\_array\_ptr, uint8\_t quantity\_of\_params);**

Данный метод отправляет на устройство массив параметров, которые будут отправляться в наборном пакете. Максимальное количество параметров, которое можно задать таким образом – 64.

В качестве аргументов данный метод принимает:

- указатель на массив номеров параметров, которые, принятых по serial-порту. Подробное описание параметров наборного пакета приведено в документе «Протокол информационного взаимодействия» ЛМАП.402131.009Д1. Список #define для этих параметров можно найти в файле «LMP\_CustomPacket.h»
- количество этих параметров

Ответом будет пакет подтверждения обработки команды. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4.

**ВНИМАНИЕ:** Смена выдаваемых данных произойдет, только если на устройстве выбран тип выдаваемого пакета данных «Наборный». Задать его можно с помощью метода **«SetCustomAlgorithmPacket()»**. В противном случае, если выбран тип выдаваемого пакета данных «Стандартный пакет алгоритма», выдаваемые данные изменятся только после смены типа пакета.

Установка битрейта основного интерфейса RS-422:

**void SetBaudrate(uint8\_t baudrate\_register\_value);**

Данный метод отправляет на устройство команду установки битрейта основного порта RS-422 в соответствии со значением, указанным в поле “baudrate\_register\_value”. Поле может принимать значения:

GKV\_BAUDRATE\_921600

GKV\_BAUDRATE\_460800

GKV\_BAUDRATE\_230400

GKV\_BAUDRATE\_115200

GKV\_BAUDRATE\_1000000

GKV\_BAUDRATE\_2000000

GKV\_BAUDRATE\_3000000

По умолчанию установлен битрейт 921600 бит/с.

Ответом будет пакет подтверждения обработки команды. Callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

Установка единиц измерения ускорения:

**void SetAccelerationUnits(uint8\_t units);**

Данный метод отправляет на устройство команду выбора единиц измерения ускорения в соответствии со значением, указанным в поле “units”. Поле может принимать значения:

GKV\_MS2 – метры в секунду в квадрате

GKV\_G - g

Ответом будет пакет подтверждения обработки команды. Callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

Установка единиц измерения угловой скорости:

**void SetAngularRateUnits(uint8\_t units);**

Данный метод отправляет на устройство команду выбора единиц измерения угловой скорости в соответствии со значением, указанным в поле “units”. Поле может принимать значения:

**GKV\_DEGREES\_PER\_SECOND** – градусы в секунду

**GKV\_RADIANS\_PER\_SECOND** – радианы в секунду

Ответом будет пакет подтверждения обработки команды. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

Установка единиц измерения угловых перемещений:

**void SetAngleUnits(uint8\_t units);**

Данный метод отправляет на устройство команду выбора единиц измерения углов поворота в соответствии со значением, указанным в поле “units”. Поле может принимать значения:

**GKV\_DEGREES** – градусы

**GKV\_RADIANS** – радианы

Ответом будет пакет подтверждения обработки команды. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

Установка предделителя частоты выдачи данных:

**void SetDataRatePrescaler (uint16\_t rate\_prescaler);**

Данный метод отправляет на устройство команду установки предделителя частоты выдачи данных (относительно базовой частоты в 1 кГц) в соответствии со значением, указанным в поле “rate\_prescaler”. Поле может принимать значения от 1 до 1000. При записи значения 0 устанавливается режим выдачи данных «По запросу» и данные могут быть запрошены пользователем с помощью команды “RequestData()”

Ответом будет пакет подтверждения обработки команды. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

Установка типа выходного синхросигнала:

**void SetSyncOutType(uint16\_t type);**

Данный метод отправляет на устройство команду выбора типа синхросигнала (сигнала, отправляемого на контакт SyncOut в момент захвата данных внутренним АЦП устройства) в соответствии со значением, указанным в поле “type”. Поле может принимать значения:

**GKV\_SYNC\_ADC\_PULSE** – импульс в момент захвата данных АЦП

**GKV\_SYNC\_ADC\_TOGGLE** – смена логического уровня в момент захвата данных АЦП

Ответом будет пакет подтверждения обработки команды. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetConfirmPacketReceivedCallback();**

Метод описан в разделе 3.4

### 3.4. Установка пользовательских callback-функций

Метод установки пользовательской callback-функции вызываемой при приёме любого пакета:

```
void SetReceivedPacketCallback(std::function<void(LMP_Device *, GKV_PacketBase *)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при распознавании среди принятых данных любого корректного пакета. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_PacketBase *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetReceivedPacketCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме пакета подтверждения получения команды:

```
void SetConfirmPacketReceivedCallback(std::function<void(LMP_Device*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при распознавании среди принятых данных пакета подтверждения получения команды. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev);
```

Здесь dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetConfirmPacketReceivedCallback(std::bind(&qLMP_Device::ConfirmPacketReceivedCallback, this, std::placeholders::_1));
```



Метод установки пользовательской callback-функции вызываемой при приёме пакета ID устройства:

```
void SetIDReceivedCallback(std::function<void(LMP_Device *, GKV_ID*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при распознавании среди принятых данных пакета ID устройства. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_ID *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetIDReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме текущих настроек устройства:

```
void SetSettingsReceivedCallback(std::function<void(LMP_Device *, GKV_Settings*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при распознавании среди принятых данных пакета текущих настроек. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_Settings *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetSettingsReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме текущих параметров наборного пакета устройства:

```
void SetCustomPacketParamReceivedCallback(std::function<void(LMP_Device *, GKV_CustomDataParam*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при распознавании среди принятых данных списка параметров наборного пакета. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_CustomDataParam *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetCustomPacketParamReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме пакета данных с датчиков в виде кодов АЦП:

```
void SetADCDataReceivedCallback(std::function<void(LMP_Device *, GKV_ADCData*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении пакета данных с датчиков в виде кодов АЦП. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_ADCData *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetADCDataReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме пакета калиброванных данных с датчиков:

```
void SetRawDataReceivedCallback(std::function<void(LMP_Device *, GKV_RawData*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении пакета калиброванных данных с датчиков. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_RawData *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetRawDataReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме пакета ориентации:

```
void SetGyrovertDataReceivedCallback(std::function<void(LMP_Device *, GKV_GyrovertData*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении пакета углов ориентации. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_GyrovertData *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetGyrovertDataReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме пакета углов инклинометра:

```
void SetInclinometerDataReceivedCallback(std::function<void(LMP_Device *, GKV_InclinometerData*)>  
ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении пакета углов инклинометра. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_InclinometerData*ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetInclinometerDataReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме пакета данных БИНС:

```
void SetBINSDataReceivedCallback(std::function<void(LMP_Device *, GKV_BINSDData*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении пакета навигационных данных. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_BINSDData *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetBINSDataReceivedCallback (std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме наборного пакета данных:

```
void SetCustomPacketReceivedCallback(std::function<void(LMP_Device *, GKV_CustomData*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении наборного пакета данных. Эта функция должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_CustomData *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetCustomPacketReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме пакета данных ГНСС:

```
void SetGNSSDataReceivedCallback(std::function<void(LMP_Device *, GKV_GpsData*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении пакета данных ГНСС. Данный пакет отправляется устройством с частотой 10 Гц при подключенном ГНСС приемнике и установке стандартного пакета выбранного алгоритма. Если выбран наборный пакет, ГНСС данные могут быть выбраны в числе параметров, передаваемых в наборном пакете. Функция обратного вызова должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_GpsData*ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetGNSSDataReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме расширенного пакета данных ГНСС:

```
void SetExtGNSSDataReceivedCallback(std::function<void(LMP_Device *, GKV_GpsDataExt*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию вызываемую при получении расширенного пакета данных ГНСС. Данный пакет отправляется устройством с частотой 10 Гц при подключенном ГНСС приемнике и установке стандартного пакета выбранного алгоритма. Если выбран наборный пакет, ГНСС данные могут быть выбраны в числе параметров, передаваемых в наборном пакете. Функция обратного вызова должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_GpsDataExt *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetExtGNSSDataReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

Метод установки пользовательской callback-функции вызываемой при приёме ответа на спецкоманду IfProto (см. раздел 3.5):

```
void SetIfProtoCommandResponseReceivedCallback(std::function<void(IfProtoConfig*)> ptrReceivedPacketProcessingFun);
```

С помощью данного метода задаётся указатель на пользовательскую функцию, вызываемую при получении пакета ответа на команду IfProto. Функция обратного вызова должна иметь вид:

```
void UserCallback(LMP_Device *dev, GKV_GpsDataExt *ptrPacket);
```

Здесь ptrPacket – это указатель на объект принятого пакета, dev – указатель на объект устройства, с которого были приняты данные.

Для привязки метода пользовательского класса может быть использована конструкция вида:

```
SetIfProtoCommandResponseReceivedCallback(std::bind(&UserClass::UserCallback, &UserClassObj, std::placeholders::_1, std::placeholders::_2));
```

### 3.5. Специальные команды для настройки параметров интерфейсов (IfProto)

Инерциальные модули серий ГKB и МГ имеют несколько дополнительных интерфейсов UART/RS-485 и CAN. Количество интерфейсов варьируется в зависимости от конкретной модели изделия.

Каждый из этих интерфейсов может быть настроен с помощью команд настройки интерфейсов IfProto.

Команда настройки режима работы дополнительного UART/RS-485:

```
void SetAUXPortMode(uint8_t port, uint16_t protocol, uint16_t proto_param, uint16_t baudrate, uint16_t mode)
```

Данный метод отправляет на устройство команду IfProto, содержащую настройки дополнительного порта UART/RS-485. Настройка портов обычно осуществляется для взаимодействия с несколькими ГНСС приемниками и/или приёма поправок от базовой станции.

Метод принимает в качестве аргументов:

Port – порт UART/RS-485, который требуется настроить. Может принимать значения:

**EIfFace::IFACE\_UART\_EXT\_MAIN** – основной интерфейс (можно настроить, например, на обработку NMEA сообщения)

**EIfFace::IFACE\_UART\_EXT\_AUX** – внешний UART/RS-485

**EIfFace::IFACE\_UART\_INT\_SNS** – UART, связанный с внутренним приёмником СНС модуля (используется в модулях ГKB-6/7/11/12)

**EIfFace::IFACE\_UART\_INT\_2** – дополнительный внутренний интерфейс (используется в ГKB-1OEM)

Protocol – протокол сторонних устройств, используемый в выбранном интерфейсе. Чаще всего отвечает за выбор типа ГНСС-приёмника, используемого в связке с ИНС. В стандартном исполнении может принимать значения:

**EProto::PROTO\_DISABLED** - интерфейс отключён.

**EProto::PROTO\_PASSTHROUGH\_TO\_MAIN** – проброс принятых данных на основной интерфейс. Все принятые по интерфейсу данные выдаются в поле data пакета типа 0x42 (общую структуру пакетов см. в разделе 2).

**EProto::PROTO\_NMEA\_FROM\_GNSS** – приём сообщений NMEA от ГНСС

В этом же поле может быть задан протокол ГНСС приёмника, с которым будет взаимодействовать устройство по выбранному интерфейсу:

**EProto::PROTO\_GNSS\_<название приёмника/протокола>** - поддерживаются протоколы: NV08C, GEOS3M, NMEA, COMNAV, NOVATEL, MNP, SEPTENTRIO, UBLOX.

Baudrate – битрейт работы выбранного интерфейса. Может принимать значения:

<b>EBaudRate::BAUDRATE_921600</b>	<b>EBaudRate::BAUDRATE_115200</b>	<b>EBaudRate::BAUDRATE_3000000</b>	<b>EBaudRate::BAUDRATE_57600</b>
<b>EBaudRate::BAUDRATE_460800</b>	<b>EBaudRate::BAUDRATE_1000000</b>	<b>EBaudRate::BAUDRATE_4000000</b>	<b>EBaudRate::BAUDRATE_38400</b>
<b>EBaudRate::BAUDRATE_230400</b>	<b>EBaudRate::BAUDRATE_2000000</b>	<b>EBaudRate::BAUDRATE_500000</b>	<b>EBaudRate::BAUDRATE_19200</b>
<b>EBaudRate::BAUDRATE_9600</b>			

Mode - режим интерфейса. Может принимать значения

**EMode::MODE\_UART**

**EMode::MODE\_RS232**

**EMode::MODE\_RS485**

Proto\_param – параметры обработки данных, принимаемых по выбранному протокола. Обычно используется для обработки поправок от базовых станций ГНСС

EProtoParam::PARAM\_USE\_GNSS\_DATA – использовать ГНСС данные полученные по данному интерфейсу  
EProtoParam::PARAM\_USE\_GNSS\_RELPOS – использовать интерфейс в качестве Rover с получением курса.  
EProtoParam::PARAM\_TO\_ESKF\_GNSS\_DATA - использовать ГНСС данные в навигационном решении (при выбранном PARAM\_USE\_GNSS\_DATA)  
EProtoParam::PARAM\_TO\_ESKF\_GNSS\_RELPOS – использовать интерфейс в качестве rover в навигационном решении (при выбранном PARAM\_USE\_GNSS\_RELPOS)  
Предделитель для NMEA сообщений (при выбранном протоколе NMEA):  
EProtoParam::PARAM\_NMEA\_PRESCALER\_1  
EProtoParam::PARAM\_NMEA\_PRESCALER\_5  
EProtoParam::PARAM\_NMEA\_PRESCALER\_10  
EProtoParam::PARAM\_NMEA\_PRESCALER\_50  
EProtoParam::PARAM\_NMEA\_PRESCALER\_100  
EProtoParam::PARAM\_NMEA\_PRESCALER\_500  
EProtoParam::PARAM\_NMEA\_PRESCALER\_1000

Ответом будет копия отправленного пакета IfProto, содержащая записанные настройки. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

```
void SetIfProtoResponseReceivedCallback();
```

Метод описан в разделе 3.4.



## Команда параметров режима проброса дополнительного UART/RS-485

**void SetAUXPPassthroughParams(uint8\_t port, uint16\_t buf\_size, uint8\_t target\_iface)**

Данный метод отправляет на устройство команду IfProto, содержащую настройки дополнительного порта UART/RS-485.

Метод принимает в качестве аргументов:

**Port** – порт UART/RS-485, который требуется настроить. Может принимать значения:

**EIFace::IFACE\_UART\_EXT\_AUX** – внешний UART/RS-485

**EIFace::IFACE\_UART\_INT\_SNS** – UART, связанный с внутренним приёмником ЧС модуля (используется в модулях ГKB-6/7/11/12)

**EIFace::IFACE\_UART\_INT\_2** – дополнительный внутренний интерфейс (используется в ГKB-1ОЕМ)

**Buf\_size** – размер пробрасываемого буфера в байтах.

**Target\_iface** – интерфейс, на который будет передаваться данные, полученные на выбранный порт. Может принимать значения:

**EIFace::IFACE\_UART\_EXT\_MAIN** – основной интерфейс (можно настроить, например, на обработку NMEA сообщения)

**EIFace::IFACE\_UART\_EXT\_AUX** – внешний UART/RS-485

**EIFace::IFACE\_UART\_INT\_SNS** – UART, связанный с внутренним приёмником ЧС модуля (используется в модулях ГKB-6/7/11/12)

**EIFace::IFACE\_UART\_INT\_2** – дополнительный внутренний интерфейс (используется в ГKB-1ОЕМ)

Ответом будет копия отправленного пакета IfProto, содержащая записанные настройки. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

**void SetIfProtoResponseReceivedCallback();**

Метод описан в разделе 3.4.

Команда настройки режима работы CAN-интерфейса:

```
void SetCANPortMode(uint8_t port, uint16_t protocol, uint16_t baudrate)
```

Данный метод отправляет на устройство команду IfProto, содержащую настройки порта CAN. Настройка портов обычно осуществляется для взаимодействия с автомобильными CAN-шинами:

**Port** – порт CAN, который требуется настроить. Может принимать значения:

EIFace::IFACE\_CAN1 – присутствует в модулях серий МГ и ГКВ

EIFace::IFACE\_CAN2 – присутствует в модулях серии ГКВ

**Protocol** – протокол сторонних устройств, используемый в выбранном интерфейсе. Чаще всего отвечает за выбор автомобиля, к шине которого подключается устройство:

EProto::PROTO\_DISABLED - интерфейс отключён.

EProto::PROTO\_RX\_PASSTHROUGHT – проброс принятых данных на основной интерфейс. Все принятые по интерфейсу данные выдаются в поле data пакета типа 0x42 (общую структуру пакетов см. в разделе 2).

EProto::PROTO\_RX\_CARCAN\_KIA\_RIO\_15 – протокол CAN-шины автомобиля KIA RIO 15

EProto::PROTO\_RX\_CARCAN\_RENAULT\_LOGAN2 - протокол CAN-шины Renault Logan

**Baudrate** – битрейт работы выбранного интерфейса. Может принимать значения:

IfCanConfig::BAUD_OFF	IfCanConfig::BAUD_250k	IfCanConfig::BAUD_50k
IfCanConfig::BAUD_1000k	IfCanConfig::BAUD_125k	
IfCanConfig::BAUD_500k	IfCanConfig::BAUD_100k	

Ответом будет копия отправленного пакета IfProto, содержащая записанные настройки. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

```
void SetIfProtoResponseReceivedCallback();
```

Метод описан в разделе 3.4.

Команда настройки сообщения, выдаваемого по CAN-интерфейсу:

```
void SetCANPortMsg(uint8_t port, uint8_t index, uint16_t prescaler, CanId id, float limit)
```

Данный метод отправляет на устройство команду IfProto, содержащую настройки порта CAN. Настройка портов обычно осуществляется для взаимодействия с автомобильными CAN-шинами:

**Port** – порт CAN, который требуется настроить. Может принимать значения:

EIFace::IFACE\_CAN1 – присутствует в модулях серий МГ и ГKB

EIFace::IFACE\_CAN2 – присутствует в модулях серии ГKB

**Index** – индекс сообщения, параметра отправки которого требуется настроить. Может принимать значения:

EIndex:: INDEX\_STATUS\_CNT – сообщение, содержащее состояние устройства и счетчик пакетов (4 байта)

EIndex:: INDEX\_Wb – сообщение, содержащее угловые скорости в формате int16 (6 байт)

EIndex:: INDEX\_Ab – сообщение, содержащее показания акселерометров в формате int16 (6 байт)

EIndex:: INDEX\_Mb – сообщение, содержащее показания магнитометра в формате int16 (6 байт)

EIndex:: INDEX\_BARO - сообщение, содержащее показания барометра в формате uint32 (4 байта)

EIndex:: INDEX\_EULER - сообщение, содержащее углы Эйлера в формате int16(6 байт)

EIndex:: INDEX\_INCLINO - сообщение, содержащее углы инклинометра в формате int16(4 байта)

EIndex:: INDEX\_ALG\_STATUS - сообщение, содержащее статус алгоритма в формате uint8(3 байта)

**Prescaler** – предделитель частоты выдачи данных относительно основного интерфейса UART/RS-485 (по умолчанию 1000 Гц). Может принимать любые значения от 1 до  $(2^{16} - 1)$ . Значение 0 означает отключение отправки сообщения.

**Id** – настраиваемый идентификатор сообщения на CAN шине. Может включать 11 (стандартный id) или 29 байт (расширенный id). Выбор типа id осуществляется методами `set_std(uint16 id)` и `set_ext(uint32 id)` структуры CanId.

**Limit** – значение используется при настройке сообщений, содержащих данные с датчиков (акселерометров, датчиков угловой скорости, магнитометров). Задаёт диапазон показаний датчиков, передаваемый int16. Единицами измерений являются:

g – для акселерометров

°/с – для датчиков угловой скорости.

Гс – для магнитометра

То есть, если при настройке сообщения показаний акселерометра задать величину 1, то, в CAN-сообщении значение  $(2^{15}-1)=32767$  будет соответствовать ускорению в 1g, а значение -32767 будет соответствовать ускорению в -1g.

Ответом на данную команду будет копия отправленного пакета IfProto, содержащая записанные настройки. Пользовательскую callback-функцию, вызываемую при получении ответа можно задать с помощью метода:

```
void SetIfProtoResponseReceivedCallback();
```

Метод описан в разделе 3.4.

## 4. Примеры

Репозиторий включает в себя четыре консольных примера использования библиотеки. В качестве системы сборки проекта используется CMake. Примеры, описанные в пунктах 4.1, 4.2 и 4.3 представляют собой простейшие однопоточные приложения, напрямую использующие класс LMP\_Device.

Пример, описанный в пункте 4.5, использует класс GKV\_FileWriter (см. пункт 4.4), наследующий LMP\_Device и запускающий дополнительные потоки считывания данных и записи их.

*Примечание:* класс GKV\_FileWriter предназначен для ознакомления с механизмом наследования класса LMP\_Device и не является наиболее оптимальным вариантом работы с COM-портом. Для более эффективного использования рекомендуется использовать кроссплатформенные библиотеки, например ASIO или Qt5.

### **4.1. Консольный пример вывода принимаемых данных с использованием WinAPI (Windows) и библиотеки termios (Linux)**

Пример ReceiveDataInConsole представляет собой консольное однопоточное приложение, которое считывает данные с инерциального модуля серий ГКВ/МГ и выводит их в консоль. При старте приложения пользователю необходимо задать serial-порт, к которому подключено устройство.

### **4.2. Консольный пример установки алгоритма «Данные с датчиков» и вывода принимаемых данных с использованием WinAPI (Windows) и библиотеки termios (Linux)**

Пример ReadRawSensorsData представляет собой консольное однопоточное приложение, которое устанавливает алгоритм «Данные с датчиков» и стандартный пакет алгоритма. И в основном цикле программа считывает данные с инерциального модуля серий ГКВ/МГ и выводит их в консоль. При старте приложения пользователю необходимо задать serial-порт, к которому подключено устройство.

### **4.3. Консольный пример выбора алгоритма пользователем и вывода принимаемых данных с использованием WinAPI (Windows) и библиотеки termios (Linux)**

Пример AlgorithmSelectionExample представляет собой консольное однопоточное приложение, которое даёт пользователю возможность выбрать алгоритм работы устройства, после чего устанавливает выбранный алгоритм и

стандартный пакет алгоритма. В основном цикле программа считывает данные с инерциального модуля серий ГКВ/МГ и выводит их в консоль. При старте приложения пользователю необходимо задать serial-порт, к которому подключено устройство.

#### **4.4. Пример наследования класса LMP\_Device для расширения функционала библиотеки**

Класс GKV\_FileWriter расположенный в папке примера FileWriterExample наследует класс LMP\_Device, и запускает два дополнительных потока std::thread.

Поток GyrovertDataReceiveThread работает в качестве цикла для считывания данных с serial-порта.

Поток GyrovertDataFileWriterThread работает в качестве цикла записи данных в bin-файл.

Также в класс вынесена инициализация порта, а также методы записи и чтения данных для Linux и Win32.

*Примечание:* класс GKV\_FileWriter предназначен для ознакомления с механизмом наследования класса LMP\_Device и не является наиболее оптимальным вариантом работы с устройством. Для более эффективного использования рекомендуется использовать оптимизированные библиотеки для вашей платформы или кроссплатформенные библиотеки, например ASIO или Qt5.

#### **4.5. Консольный пример записи данных в бинарный файл с использованием WinAPI (Windows) и библиотеки termios (Linux)**

Пример FileWriterExample представляет собой консольное приложение, которое записывает данные в bin-файл с именем «LogData\_+текущая дата и время», используя класс GKV\_FileWriter. При старте приложения пользователю необходимо задать serial-порт, к которому подключено устройство.