5.1 Разработка ПО верхнего уровня под ОС семейства Windows для связи устройства с ПЭВМ для обмена данными хранящимися в памяти.

Для связи с устройством используется универсальный порт USB.

К порту USB может быть подключено как само устройство (если в устройстве есть в наличии порт USB), так и модуль беспроводной передачи данных HC-12. Алгоритм обмена в обоих вариантах одинаков.

Обмен с устройством происходит посредством драйвера виртуального COM-порта, который позволяет обмениваться посылками в стандартном формате UART. ПО верхнего уровня использует функции WinAPI для работы с виртуальным COM-портом.

Чтение данных из порта:

int HC12::ReadBytes()

{

static int all\_bytes = 0;

static TimeMeterMS meter;

int readed\_bytes = 0;

if (opened\_port != -1)

{

static const int SIZE\_BUFFER = 1000;

unsigned char buffer[SIZE\_BUFFER];

int num\_bytes = RS232\_PollComport(opened\_port, buffer, SIZE\_BUFFER);

if (num\_bytes != 0)

{

ReceivedData::Append(buffer, num\_bytes);

readed\_bytes += num\_bytes;

}

}

all\_bytes += readed\_bytes;

if (meter.ElapsedTime() >= 1000)

{

LOG\_WRITE("%d bytes / sec", all\_bytes);

meter.Reset();

all\_bytes = 0;

}

return readed\_bytes;

}

Далее в главном цикле принятые данные обрабатываеются:

void ReceivedData::Update()

{

while (buffer.GetElementCount() >= SIZE\_MESSAGE)

**{**

if (buffer[0] != 'A')

{

buffer.RemoveFirst(1);

continue;;

}

if (buffer[1] != 'B')

{

buffer.RemoveFirst(2);

continue;;

}

if (buffer[2] != 'C')

{

buffer.RemoveFirst(3);

continue;

}

char bytes[SIZE\_MESSAGE];

buffer.GetData((uint8 \*)bytes, SIZE\_MESSAGE);

static int num\_command = 0;

num\_command++;

if (!ParseCommand(bytes))

{

static int prev\_num = 0;

LOG\_ERROR("Can not parse command %d, parsed %d", num\_command, num\_command - prev\_num);

prev\_num = num\_command;

// \todo удалить

ParseCommandOld(bytes); // Пытаемся распарсить по старому формату передачи данных

}

**}**

}

Данные из входной последовательности извлекаются следующим образом:

bool ReceivedData::ParseCommand(char message[SIZE\_MESSAGE])

{

uint8 type = message[3];

static int num\_types[4] = { 0, 0, 0, 0 };

if (type < 4)

{

num\_types[type]++;

LOG\_WRITE("%d = %d, %d = %d, %d = %d, %d = %d", 0, num\_types[0], 1, num\_types[1], 2, num\_types[2], 3, num\_types[3]);

}

uint id;

*std*::*memcpy*(&id, &message[4], 4); // offset 4

if (type == Measure::Temperature)

{

LOG\_WRITE("Receive temperature from device %08X", id);

}

uint hash = 0;

*std*::*memcpy*(&hash, &message[8], 4); // offset 12

float value = 0.0f;

*std*::*memcpy*(&value, &message[12], 4); // offset 16

if (Math::CalculateHash(&value, 4) == hash)

{

Sensor::Pool::AppendMeasure(id, type, value);

return true;

}

else

{

LOG\_ERROR("Invalid hash for measure %u device %08X", type, id);

}

return false;

}

После этого принятые данные сохраняются для последующего отображения и пересылки на удалённый сервер:

void Sensor::Pool::AppendMeasure(uint id, uint8 type, float value)

**{**

static float values[Measure::Count] =

{

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f,

0.0f

};

if (type == Measure::Temperature)

{

LOG\_WRITE("Receive temperature %f from device %08X", value, id);

}

if (type >= Measure::Count)

{

LOG\_ERROR("Unknown measure type %u", type);

return;

}

if (id == 0xD5E0B863)

{

values[type] = value;

}

auto sensor = pool.*find*(id);

if (sensor == pool.*end*())

{

pool.*try\_emplace*(id, Sensor( id, Pool::ColorForSensor() ));

}

sensor = pool.*find*(id);

if (sensor != pool.*end*())

{

sensor->*second*.AppendMeasure(type, value);

MainWindow::SetMeasure(id, sensor->*second*.GetColor(), type, value);

}

static TimeMeterMS meter;

static bool first = true;

if (values[Measure::Temperature] != 0)

{

if ((meter.ElapsedTime() > (uint)(1000 \* 60 \* SET::NETWORK::time\_http.Get())) || (first && meter.ElapsedTime() > 5000))

{

first = false;

meter.Reset();

if (id == 0xD5E0B863)

{

HTTP::SendPOST(101, values[Measure::Temperature], values[Measure::Humidity], values[Measure::Pressure], values[Measure::DewPoint], values[Measure::Illuminate]);

}

else

{

LOG\_WRITE("Not device");

}

}

}

**}**