Разработка плагинов для Multi-Protocol MasterOPC

Быстрый старт к Multi-Protocol MasterOPC Server SDK

Руководство пользователя

ОГЛАВЛЕНИЕ

1. *Структура плагина*

1.1. Установка необходимых программ

Для работы Multi-Protocol MasterOPC необходима Windows 7, 8, 8.1, Windows Server 2008, Windows Server 2012.

Для работы необходимо установить следующие программы:

- ***MULTI-PROTOCOL-SDK-MASTEROPC***
- ***SERVERDEVELOP*** – примеры.
- ***Microsoft Visual Studio 2013*** – в нем будет производиться написание кода плагина, отладка.
- Matrikon OPC Explorer или ICONICS OPC DataSpy – для просмотра значений OPC DA тэгов.
- Matrikon OPC HDA Explorer – для просмотра архивов OPC HDA.

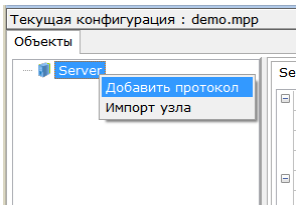
1.2. Первое знакомство

В папке ***C:\Program Files (x86)\InSAT\Multi-Protocol SDK MasterOPC*** находится сам OPC-сервер (***mpssdk.exe***) и конфигуратор протокола (***protocol-maker.exe***).

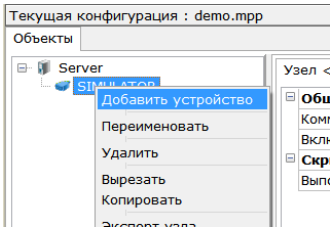
У сервера есть рабочая папка ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server***:

- В папке ***SERVERCFG*** находится конфигурация по умолчанию. Файл ***demo.mpp***.
- В папке ***SERVERDEVELOP*** находятся исходные коды примеров.
- В папке ***SERVERLOGS*** находятся Лог-файлы.
- В папке ***\SERVERPLUGINS\INPOUT*** находятся библиотеки плагинов (файлы с расширением ***spl***).
- В папке ***SERVERPLUGINS\TEMPLATES_PCF*** находятся конфигурации протоколов.

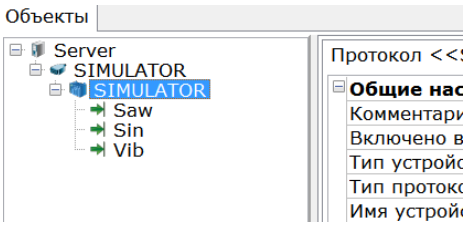
Ознакомимся с работой сервера. Запустите OPC сервер командой ***Пуск – Программы – Insat – Multi-Protocol SDK MasterOPC - Multi-Protocol SDK MasterOPC***. В панели меню открывшегося окна OPC сервера создайте новую конфигурацию. Выделите элемент ***Server***, нажмите правую кнопку мыши и выберите ***Добавить протокол***.



В появившемся окне выберите ***SIMULATOR***, нажмите ***Да*** – в *дерево* добавится ***SIMULATOR***. Добавьте в него ***устройство***.

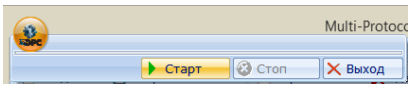


В *устройстве* ***SIMULATOR*** через контекстное меню добавьте по очереди все тэги протокола.

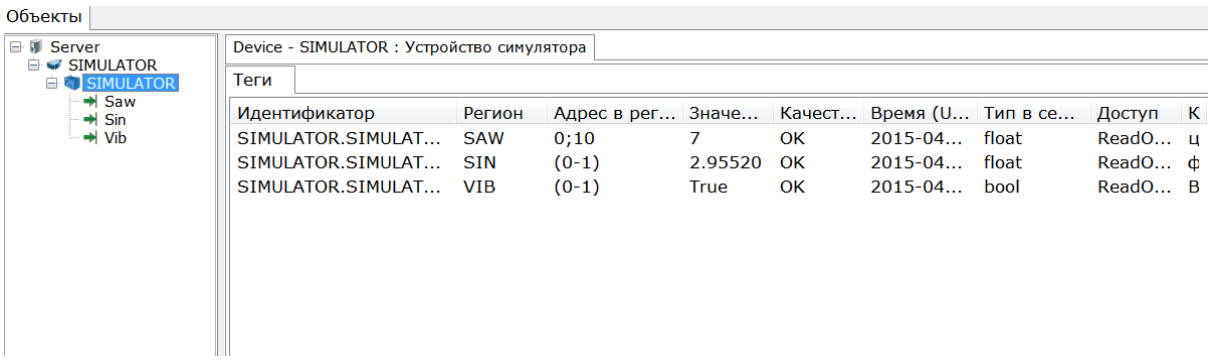


Теперь попробуем наш сервер в действии. В левом верхнем углу найдите пиктограмму

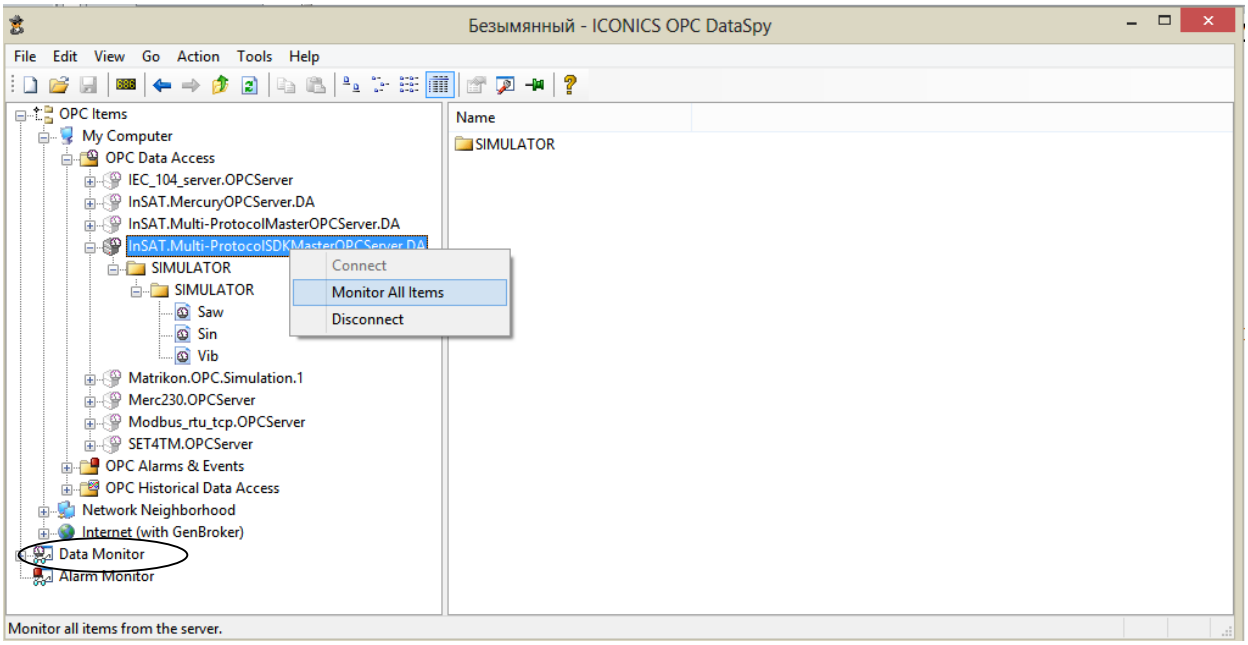
Нажмите на нее и выберите ***Старт***



Теперь сервер перешел в режим исполнения. В правой части мы видим список тэгов с их значениями.



Подключимся к серверу с помощью OPC-клиента. Воспользуемся программой ***ICONICS OPC DataSpy*** – запустим его. В поле справа раскроем дерево и выберем ***InSAT.Multi-ProtocolSDKMasterOPCServer.DA***



Выберем ***Monitor All Items***, щелкнем на ***Data Monitor***. В правом поле увидим значения наших тэгов. Таким образом происходит настройка OPC-клиента для получения данных от OPC-сервера.

2. Структура плагина

Разберем структуру плагина на примере демонстрационного проекта ***mpsplugin-dcondemo***, входящего в комплект OPC сервера. Этот проект показывает пример работы с протоколом ***DCON***.

Проект находится в папке ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERDEVELOP***.

Для открытия проекта необходимо открыть mpsplugin.sln в Visual Studio. В состав проекта входят следующие файлы:

- В файле ***dcon.cpp*** находятся функции работы с портом, отправки данных, приема данных.
- В файле ***plugin-close.cpp*** находится функция закрытия драйвера, которая вызывается сервером один раз для освобождения ресурсов.
- В файле ***plugin-common.cpp*** производится опрос устройств, вызываются процедуры из ***dcon.cpp***.
- В файле ***plugin-create.cpp*** находится функция создания драйвера, которая вызывается один раз при загрузке сервера.
- В файле ***plugin-init.cpp*** находится функция инициализации драйвера, которая вызывается один раз при старте Runtime.
- В файле ***plugin-read.cpp*** находится функция чтения драйвера, которая вызывается циклически из сервера.
- В файле ***plugin-write.cpp*** находится функция записи драйвера, которая вызывается при необходимости записи в теги.

После компиляции проекта формируется файл плагина ***mpsplugin-dcondemo.spl***. Он появляется в папке ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERPLUGINS\INPOUT***.

3. Первый шаг. Пишем Hello Word!!!

Закройте открытые программы ***ICONICS OPC DataSpy*** и ***mpssdk.exe***. В папке ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERDEVELOP\mpsplugin-simulator*** откройте файл ***mpsplugin.sln*** в среде ***Microsoft Visual Studio 2013***. Вы открыли проект плагина, структуру которого мы разбирали в предыдущем пункте.

> ***Примечание.*** *В Microsoft Visual Studio 2013*
>
> *В Microsoft Visual Studio 2013 слева есть панель обозревателя решений. Из этой панели можно открывать различные файлы проекта. Пункт меню «Сборка – Собрать решение» (Ctrl+Shift+B) – создается файл плагина mpsplugin-simulator.spl в папке SERVERPLUGINS\INPOUT. Кнопка «Локальный отладчик Windows» - компилируется проект и запускается mpssdk.exe, можно отлаживать проект, устанавливать точки останова, смотреть значения переменных и т.д.*

Для начала изменим алгоритм работы тэга ***SAW*** – например, сделаем чтобы он показывал нам значение 500. Откроем ***plugin-common.cpp***. Найдем строку `if (strcmp(tagdata->region, "SAW") == 0)`. В данной строчке происходит обращение к тегу со значением свойства ***“SAW”*** (подробнее свойства тегов мы разберем далее). Сотрем внутри {} все лишнее и запишем:

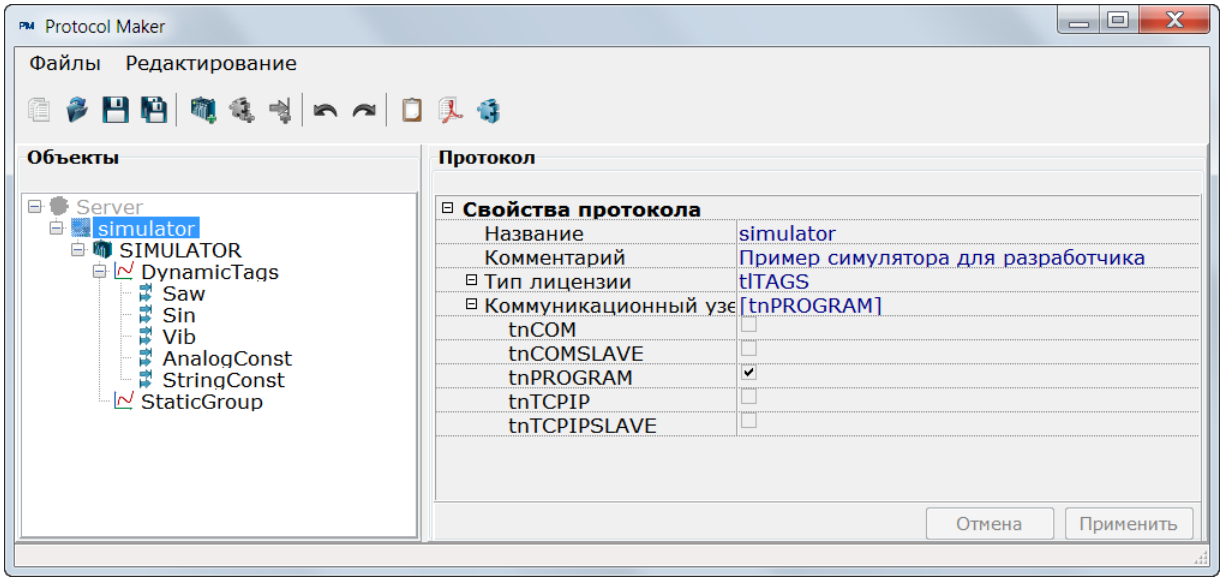
```
if (strcmp(tagdata->region, "SAW") == 0)
{
            mps_setTagAsFloat(tag, 500);
}
```

Процедура `mps_setTagAsFloat(tag, 500);` выводит значение 500 в OPC-тэг `tag`.

Запустим проект, запустим сервер, видим, что тэг SAW равен 500.

Теперь создадим свой собственный тэг.

Для создания макета конфигурации устройства предназначена специальная программа ***Protocol-Maker***. Запустим ее ***Пуск – Программы – Insat – Multi-Protocol SDK MasterOPC Server – Protocol-Maker***. Запустим ее и откроем файл ***mpsplugin-simulator.pcf***.



У устройства есть две группы:

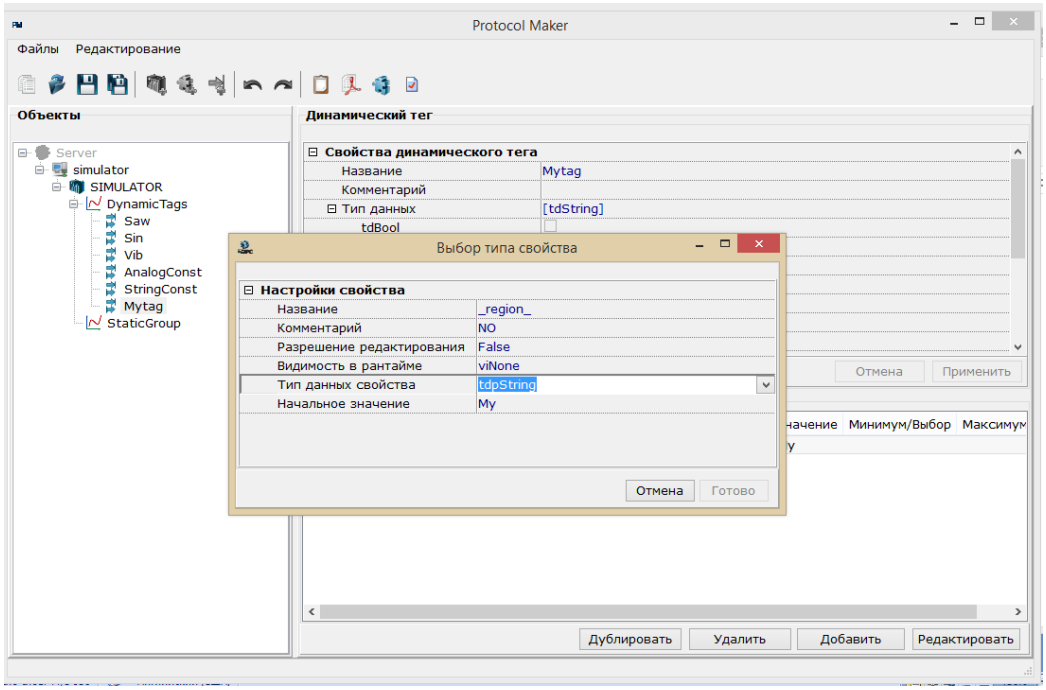
- ***DynamicTags*** – группа динамических тегов. В ***Protocol-Maker*** вы создаете шаблоны динамических тегов с определенными свойствами, а непосредственно сами теги добавляются в устройство пользователем из списка созданных шаблонов.
- ***StaticGroup*** – группа статических тегов. Все тэги этой группы создаются автоматически при добавлении устройства.

В ***DynamicTags*** создадим свой тэг через контекстное меню. Назовем его ***Mytag***. Выберем тип данных *tdString*. Также можно установить тип доступа – ***ReadOnly*** (только чтение), ***WriteOnly*** (только запись) и ***ReadWrite*** (чтение и запись).

Настройки ***Тип редактирования*** определяют какие действия с тегом (или группой) может выполнять пользователь – копировать, удалять, перемещать, переименовывать.

Если предполагается записывать в тег архивные значения, то нужно включить настройку ***Разрешение HDA*** и задать свойства HDA тега.

В правой нижней панели ***Дополнительные свойства*** нажмите на кнопку ***Добавить***. В дополнительных свойствах мы можем сохранить свою информацию о тэге, которую можно будет впоследствии извлечь в своем плагине. Мы добавим новое свойство с названием *_region_*, типом данных *tdpString* (мы будем записывать строку `Hello Word!!`) и начальным значением *My*. Нажмите на кнопку ***Сохранить*** и ***Сохранить в рабочую папку***.



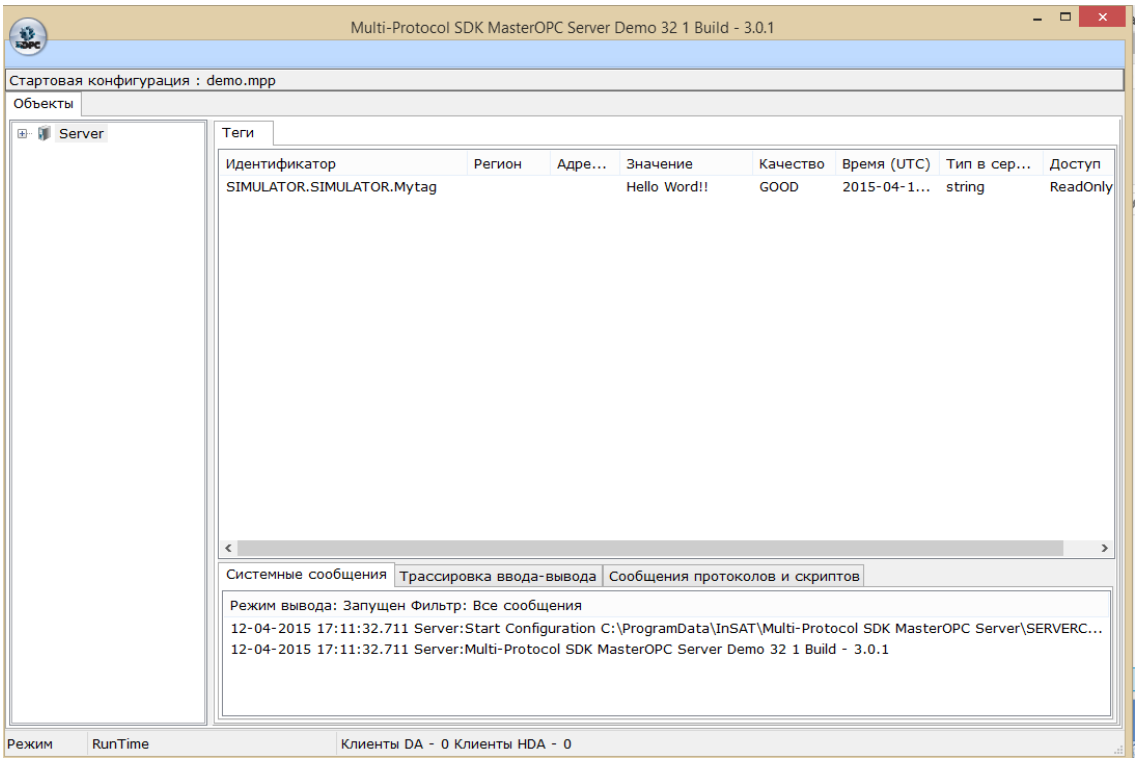
Вернемся к нашему проекту в среду ***Visual Studio***. В файле ***plugin-common.cpp*** мы видим, что все тэги по очереди перебираются в цикле. Кто есть, кто мы узнаем через значение дополнительного свойства с названием `_region_` через `tagdata->region` . Эту информацию плагин получает в ***plugin-init.cpp*** с помощью процедуры `mps_property_as_string` и сохраняет в рабочей структуре.

```
mytagdata->region = mps_property_as_string(propnode, "_region_", "");
```

Таким образом, в ***plugin-common.cpp*** мы найдем свой тэг, когда `tagdata->region` будет равным *My*. Напишем следующее:

```
if (strcmp(tagdata->region, "My") == 0)
{
                mps_setTagAsString(tag, "Hello Word!!");
}
else  ..........
```

Запустим наш проект. Запустим сервер в режим исполнения. Видим, что значение нашего тэга `Hello Word!!`


Multi-Protocol SDK MasterOPC Server Demo 32 1 Build - 3.0.1
Стартовая конфигурация : demo.mpp
Объекты
Server
Теги
Идентификатор
Регион
Адре...
Значение
Качество
Время (UTC)
Тип в сер...
Доступ
SIMULATOR.SIMULATOR.Mytag
Hello Word!!
GOOD
2015-04-1...
string
ReadOnly
Системные сообщения
Трассировка ввода-вывода
Сообщения протоколов и скриптов
Режим вывода: Запущен Фильтр: Все сообщения
12-04-2015 17:11:32.711 Server:Start Configuration C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERC...
12-04-2015 17:11:32.711 Server:Multi-Protocol SDK MasterOPC Server Demo 32 1 Build - 3.0.1
Режим
RunTime
Клиенты DA - 0 Клиенты HDA - 0

4. Этапы создания плагина

Рассмотрим этапы создания плагина:

1. В ***protocol-maker*** создаются тэги, значения которых будем получать из устройства. В каждом тэге создадим дополнительные свойства, чтобы знать, какой тэг к какому параметру устройства относится. При необходимости создадим дополнительные свойства для устройства.

2. За основу нашей разработки мы будем использовать демонстрационный проект ***mpsplugin-dcondemo*** (также находящийся в папке ***SERVERDEVELOP***). Переименуйте его под название своего протокола. После этого измените его под протокол обмена – для этого в файле ***plugin-common.cpp*** удалите все из функции `bool ReadDevice(OpcMP_plugin_t * plugin, USList *readvar)`. Проект ***mpsplugin-dcondemo*** можно использовать на этапе обучения. После обучения рекомендуется создавать свои процедуры.

3. В файле ***plugin-init.cpp*** с помощью процедуры `mps_property_as_(тип данных)` получаем значения дополнительных свойств тэгов и сохраняем в рабочей структуре. Дополнительные свойства можно создавать и для устройства.

4. В файле ***plugin-common.cpp*** найдите уже готовую процедуру `ReadDevice`. В нее добавим структуру с информацией о запросе. `struct Zapros *Zpr = new struct Zapros;`

5. Заполнить структуру `Zpr` в соответствии с вашими задачами.

6. В цикле организовать последовательное чтение данных с устройства. Удобно выносить чтение одного параметра в отдельную процедуру. После считывания параметра с устройства, пересчитать его в инженерное значение, вывести результат в OPC-тэг с помощью процедуры `mps_setTagAsDouble (mps_setTagAsFloat,` или другой, в зависимости от выбранного типа данных тэга в protocol-maker). С помощью процедуры `mps_setTagQwality(opcmp_tag, OPCMP_QUALITY_GOOD);` установить качество тэга. `OPCMP_QUALITY_GOOD` – хорошее качество тэга, устанавливается, если данные считаны успешно. `OPCMP_QUALITY_BAD` - плохое качество тэга, устанавливается, если данные считаны неуспешно. Таблица с константами признаков качества приведена в приложении 7.

7. В конце процедуры `ReadDevice` очистить память. `delete Zpr;`

8. Разработанная библиотека плагина должна иметь следующее имя: ***mpsplugin-<имя протокола>.spl***. Для подключения плагина к ***Multi-Protocol SDK MasterOPC Server*** достаточно поместить скомпилированную библиотеку и соответствующий PCF-файл в папку ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERPLUGINS\INPOUT***

9. Если Вы поместите свой проект плагина в папке ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERDEVELOP***, то плагин ***mpsplugin-<имя протокола>.spl*** после сборки будет оказываться там, где необходимо: в папке ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERPLUGINS\INPOUT***. Если ведете разработку в другом месте, то файл ***mpsplugin-<имя протокола>.spl*** после сборки следует копировать в ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERPLUGINS\INPOUT***.

5. Пример создания OPC DA сервера счетчика электрической энергии

В качестве пример рассмотрим создание плагина OPC DA для счетчиков электроэнергии ***СЭТ-4***. Данные счетчики электроэнергии имеют интерфейсы связи RS-485 и (или) оптопорт, и поддерживают ***MODBUS***-подобный двоичный протокол.

Структура фрейма запроса приведена на рисунке:

Сетевой адрес	Код запроса	Код параметра	Параметры	CRC	
				CRCL	CRCH

CRC рассчитывается как контрольная сумма MODBUS.

Структура фрейма ответа приведена на рисунке:

Сетевой адрес	Поле данных ответа	CRC	
		CRCL	CRCH

5.1. Создание тэгов в программе Protocol-maker

Запустите программу ***Protocol-maker***. Заполните поля ***Свойства протокола***.

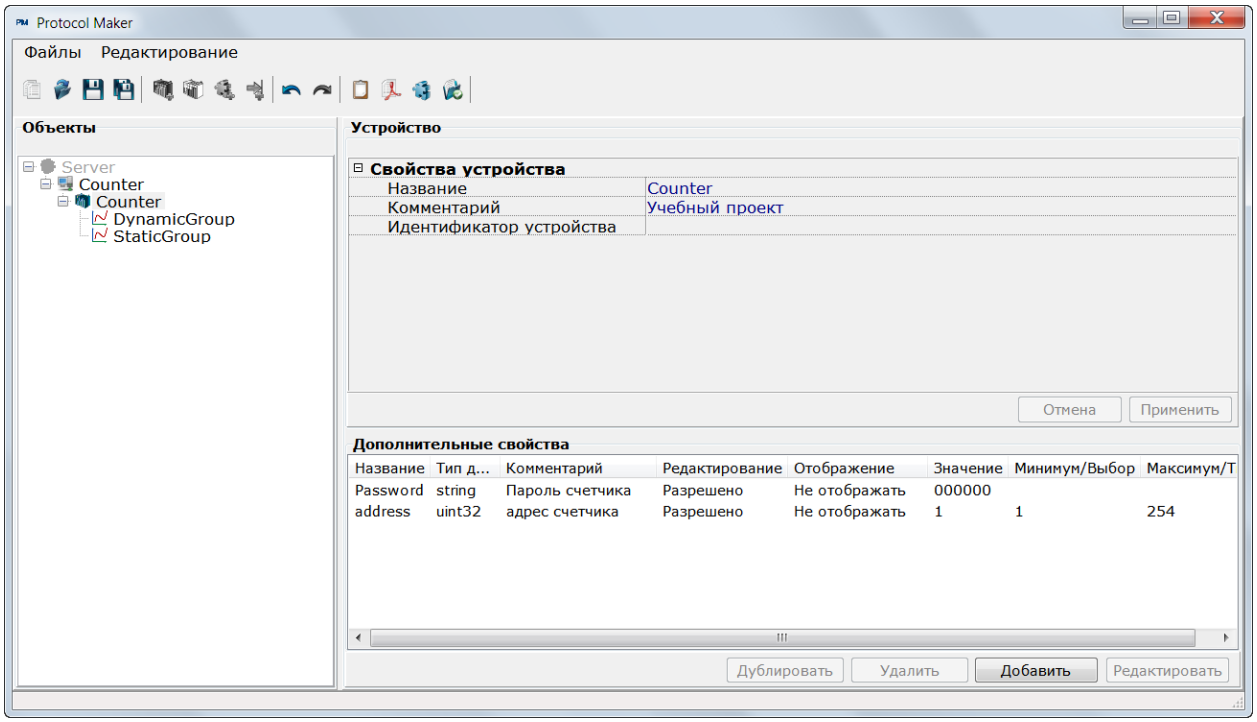
Название	**Значение**
Название	Counter
Комментарий	Счетчик. Учебный проект
Тип лицензии	tlDEVICES – лицензирование по количеству устройств tlTAGS - лицензирование по количеству тэгов
Коммуникационный узел	Отметьте следующие: ✓ tnCOM – опрос устройства производится через COM-порт компьютера ✓ tnTCPIP – опрос устройства производится поверх TCP IP. Данный режим используется при работе с преобразователями Ethernet/RS-485, такими как Moxa NPort.

Заполните поля ***Свойства устройства***.

Название	Значение
Название	Counter
Комментарий	Учебный проект

У счетчика электроэнергии есть такие параметры как его ***адрес*** и ***пароль***. При настройке OPC-сервера пользователь должен иметь возможность настраивать эти параметры. В этом нам помогут дополнительные свойства устройства. Создайте следующие свойства:

Название	Комментарий	Разрешение редактирования	Тип данных свойства	Начальное значение	Минимальное значение	Максимальное значение
Password	Пароль счетчика	True	tdpString	000000		
address	адрес счетчика	True	tdpUint32	1	1	254



Пароль счетчика состоит из 6 символов. Заводская установка ***000000***.

Адрес счетчика задается в диапазоне от ***1*** до ***254***.

Перейдем к созданию тэгов. Со счетчика будем считывать суммарную мощность и напряжение фазы А. В OPC-тэг `Failure` будем выводить информацию, есть ли связь со счетчиком. Создадим следующие тэги в группе ***StaticGroup***:

Название	Комментарий	Тип данных	Тип доступа	Тип редактирования
Psum	Суммарная активная мощность	tdDouble	taRead	По желанию
Ua	Напряжение фазы А	tdDouble	taRead	
Failure	Отказ устройства	tdBool	taRead	Запретите все действия – пользователь не должен иметь возможности редактировать данный тег

Для каждого тэга создадим ***дополнительное свойство***. Рекомендуется сначала создать один тэг, в нем создать дополнительные свойства, а потом продублировать этот тэг.

Название	Комментарий	Разрешение редактирования	Видимость	Тип данных свойства	Начальное значение
channel1	Тэги в программе будем различать по их номеру	False	viNone	tdpUint32	
TypeAnswer	Как расшифровывать ответ от счетчика	False	viNone	tdpUint32	

Значения дополнительных свойств:

ТЭГ	channel1	TypeAnswer
Psum	1	1
Ua	2	2
Failure	0	0

Пока работа в ***Protocol-maker*** закончена. Нажмите кнопки ***Сохранить*** и ***Сохранить в рабочей папке***.

5.2. Программирование обмена с устройством

Прочитаем адрес счетчика и пароль, введенные пользователем и сохраним их в рабочей структуре. В файле ***plugin-common.h*** в структуру `my_plugin_data` добавим

```
const char * Password; //Пароль счетчика
```

В файле ***plugin-common.h*** в структуру `Zpr` добавим

```
char Password[6]; //Пароль счетчика
```

и убедимся, что там есть

```
int    address; //Адрес счетчика
```

В файле ***plugin-init.cpp*** прочитаем адрес счетчика и пароль, введенные пользователем:

```
data->address = mps_property_as_int(propdevicenode, "address", 1);
```

```
data->Password = mps_property_as_string(propdevicenode, "Password", "000000");
```

Прочитаем свойства тэгов и запомним их в рабочей структуре. В файле ***plugin-common.h*** в структуру `my_tagdata и Zpr`, добавим

```
unsigned int TypeAnswer;
```

и убедимся, что там есть

```
unsigned int NumberChannel;
```

В файле ***plugin-init.cpp*** прочитаем свойства тэгов:

```
mytagdata->NumberChannel = mps_property_as_int(propnode, "channel1", 1000);
```

```
mytagdata->TypeAnswer = mps_property_as_int(propnode, "TypeAnswer", 0);
```

Цикл опроса счетчика будет состоять из трех частей:

1. Открыть канал связи;
2. Прочитать активную мощность;
3. Прочитать напряжение.

Найдем процедуру `ReadDevice` в ***plugin-common.cpp***. Удалим из нее все. При необходимости можно переделать ее под функцию и возвращать результат успешности опроса.

5.2.1. Открытие канала связи

Перед началом обмена нужно выполнить ***авторизацию*** - послать в счетчик ***пароль***, т.е. открыть канал связи. Если этого не сделать, счетчик не будет выдавать данные.

Создадим структуру `Zpr`

```
struct Zapros *Zpr = new struct Zapros;
```

Заполним структуру `Zpr`

```
Zpr->Address = data->address; //Адрес устройства
```

Открытие канала связи происходит с использованием пароля, который ввел пользователь.

Перенесем пароль в структуру `Zpr` из `data`.

```
//Проверка правильности введенного пароля. Пароль всегда должен быть 6 символов!
if (strlen(data->Password) != 6)
	{
		mps_DebugMessage(plugin, "plugin: ОШИБКА: Пароль счетчика не 6 символов");
		Zpr->Password[0] = 30;
		Zpr->Password[1] = 30;
		Zpr->Password[2] = 30;
		Zpr->Password[3] = 30;
		Zpr->Password[4] = 30;
		Zpr->Password[5] = 30;
	}
	else
	{
		Zpr->Password[0] = data->Password[0];
		Zpr->Password[1] = data->Password[1];
		Zpr->Password[2] = data->Password[2];
		Zpr->Password[3] = data->Password[3];
		Zpr->Password[4] = data->Password[4];
		Zpr->Password[5] = data->Password[5];

	}
```

Формирование пакетов, прием данных от счетчика и их расшифровку нагляднее делать в отдельной функции `Read`. Можно использовать уже существующую процедуру ***Read*** - удалив из нее все, переделать ее под функцию, и писать в ней свой код. В качестве параметра будем передавать, что считывать через `Zpr->QKind`.

```
//Открыть канал связи. Послать пароль
Zpr->QKind = 0;//Послать пароль
```

Для открытия канала мы должны счетчику послать пакет

Сетевой адрес	Код запроса 01h	Пароль (6 байт)	КС (CRC)

При адресе счетчика равном 1 и пароле 000000 он будет таким:

```
01 01 30 30 30 30 30 30 CF E8
```

Формирование пакета:

```
WORD  WORD_CRC;
byte byte_h, byte_l;

Addr = Zpr->Address;

BufferTo[0] = Addr;

BufferTo[1] = 0x01;// Код запроса =1h
BufferTo[2] = Zpr->Password[0];
BufferTo[3] = Zpr->Password[1];
```

```
        BufferTo[4] = Zpr->Password[2];
        BufferTo[5] = Zpr->Password[3];
        BufferTo[6] = Zpr->Password[4];
        BufferTo[7] = Zpr->Password[5];

        WORD_CRC = Crc16(BufferTo, 8); //Вычисление контрольной суммы ModBus приведено в прил.1
                byte_h = WORD_CRC >> 8;
                byte_l = WORD_CRC & 0x00FF;

                BufferTo[8] = byte_l;
                BufferTo[9] = byte_h;

                Rez = SendBlockToRs(Plugin, Zpr->Gaddress, BufferTo, 10,BufferFrom, 57,3);
//Функция SendBlockToRs приведена в приложении 6

if (Rez == 0) //При успешном открытии счетчик должен вернуть 4 байта 01 00 00 20
        {
//Если счетчик не ответил на открытие канала связи, то всем тэгам надо присвоить плохое
качество!!!
                        mps_DebugMessage(Plugin, "счетчик не ответил на открытие канала связи");
                        Zpr->Status = -1; //Возвращаем информацию о неудаче
                        return false;
                };
                if (Rez == 4) //проверка контрольной суммы
                {
                                WORD_CRC = Crc16(BufferFrom, Rez - 2);
                                byte_h = WORD_CRC >> 8;
                                byte_l = WORD_CRC & 0x00FF;
                                if ((byte_l == BufferFrom[Rez - 2]) && (byte_h == BufferFrom[Rez
- 1]))
                                {
                                        Zpr->Status = 1; //Контрольная сумма сошлась!

                                }
                                else
                                {
                                        Zpr->Status = -1; //Возвращаем информацию о неудаче
                                        mps_DebugMessage(Plugin, "ошибка контрольной суммы CRC");
                                        return false;
                                }

                        return true;
                };//Если придет 3 байта – это неправильный пакет. Игнорируем его
```

Кроме этих проверок можно проверить первый байт ответа – он всегда равен адресу счетчика.

Второй байт - значение состояния обмена (код ответа). В младших 4 битах кода ответа содержится информация о состоянии обмена. Если в них 0, то все в порядке, в противном случае содержится какая то ошибка. Самый распространенный случай – не открыт канал связи, в этом в младших битах содержится число 5. Остальные коды ошибок можно найти в документации к протоколу счетчика.

Выполним маскирование 4 младших бит и сравним с константой ошибки открытия канала:

```
    if ((BufferFrom[1] & 0x0F)==5)  //X5h Не открыт канал связи
        {
        mps_DebugMessage(Plugin, "Ответ от счетчика: неправильный пароль счетчика. Не открыт канал
связи");
        };
```

Если открытие канала связи неудачное, то тэгу ***Failure*** присвоим `true`, не следует тратить время на опрос других параметров.

```
	if (Zpr->Status == -1)
	{
		Main_Error = true;//запишем true, если открытие канала неуспешно, другие
параметры не будем опрашивать
	}
	else
	{
		Main_Error = false; //запишем false, если открытие канала успешно
	}
```

После открытия канала связи переберем тэги.

```
	for (int i = 0; i < readdev->count(); i++)
	{
		OpcMP_tag_t *opcmp_tag = (OpcMP_tag_t *)readdev->getItem(i);
		my_tagdata * mytagdata = (my_tagdata *)mps_getTagData(opcmp_tag);
		double Val;//
		if (mytagdata->NumberChannel == 0) //Ищем тэг Failure, говорящий OPC-
клиенту, есть ли связь со счетчиком.
		{
			if (Main_Error)
			{
				mps_setTagQwality(opcmp_tag, OPCMP_QUALITY_GOOD);
				mps_setTagAsBoolean(opcmp_tag, true);
			}
			else
			{
				mps_setTagQwality(opcmp_tag, OPCMP_QUALITY_GOOD);
				mps_setTagAsBoolean(opcmp_tag, false);
			}
		}
		else
		if (mytagdata->NumberChannel < 1000)
		{
			Zpr->QKind = mytagdata->NumberChannel;//В это значение мы занесли в
п. 5.1. По нему мы узнаем, какой параметр надо опросить
			Zpr->TypeAnswer = mytagdata->TypeAnswer;
			if (Main_Error == false)
				Read(plugin, Zpr, readdev);
			if (Zpr->Status == -1) //Опрос параметра неудачный
			{

				mps_setTagQwality(opcmp_tag, OPCMP_QUALITY_BAD); //Качество
тэга 					плохое
			}
			else
			{
				Val = UnpackResult(Zpr->Buffer, Zpr, Zpr->LenBuffer);
				mps_setTagAsDouble(opcmp_tag, Val); //Передаем значение
переменной Val OPC-клиенту. Переменная Val пересчитана по формуле из ответа счетчика

				mps_setTagQwality(opcmp_tag, OPCMP_QUALITY_GOOD); //Качество
тэга хорошее
			}
		}
	}
```

5.2.2. Считывание активной мощности

Запрос на чтение:

```
01 08 11 00 8C 4A
```

01 – адрес счетчика

08 – код запроса

11 – код параметра

00 - байт RWRI (см. описание протокола обмена счетчиков)

8C 4A – контрольная сумма

В ответ счетчик должен вернуть 6 байтов:

```
01 00 E3 04 49 2B
```

01 – адрес счетчика

00 E3 04 - ответ на запрос чтения вспомогательных параметров – это слово из трех байт в двоичном коде. Слово содержит значение и направление энергии. Пример получения активной мощности из данного слова приведен в приложении 2.

49 2B – контрольная сумма

5.2.3. Считывание напряжения фазы А

Запрос на чтение:

```
01 08 11 11 4C 46
```

01 – адрес счетчика

08 – код запроса

11 – код параметра

11 - байт RWRI (см. описание протокола обмена счетчиков)

4C 46 – контрольная сумма

В ответ счетчик должен вернуть 6 байтов:

```
01 00 E3 04 49 2B
```

01 – адрес счетчика

00 E3 04 - ответ на запрос чтения вспомогательных параметров. Формула пересчета $U(B) = \frac{Nu}{100} \cdot Kн$

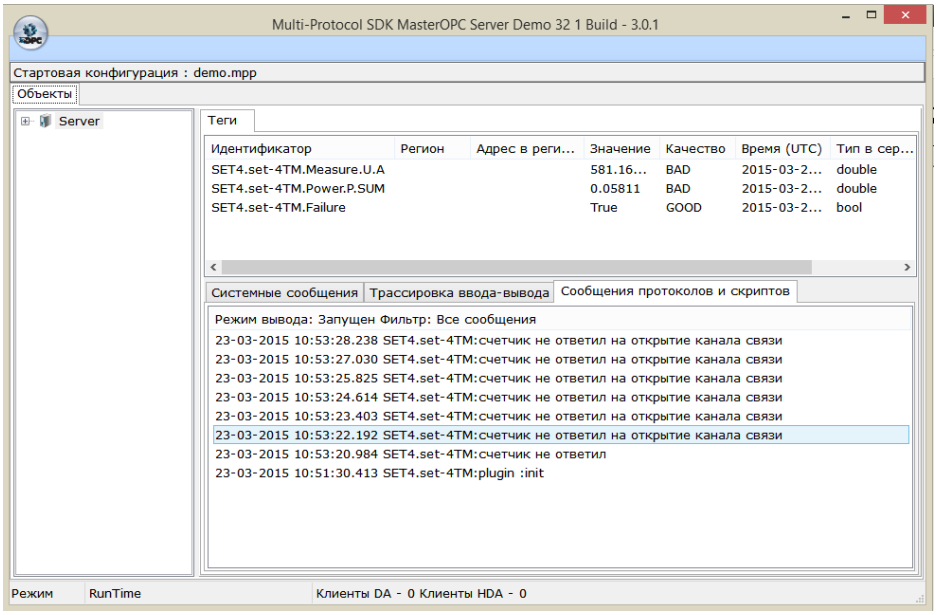
Nu - ответ на запрос. *Кн* - коэффициент трансформации трансформатора напряжения (1 – если трансформатора напряжения нет). Код пересчета приведен в приложении 3.

49 2B – контрольная сумма

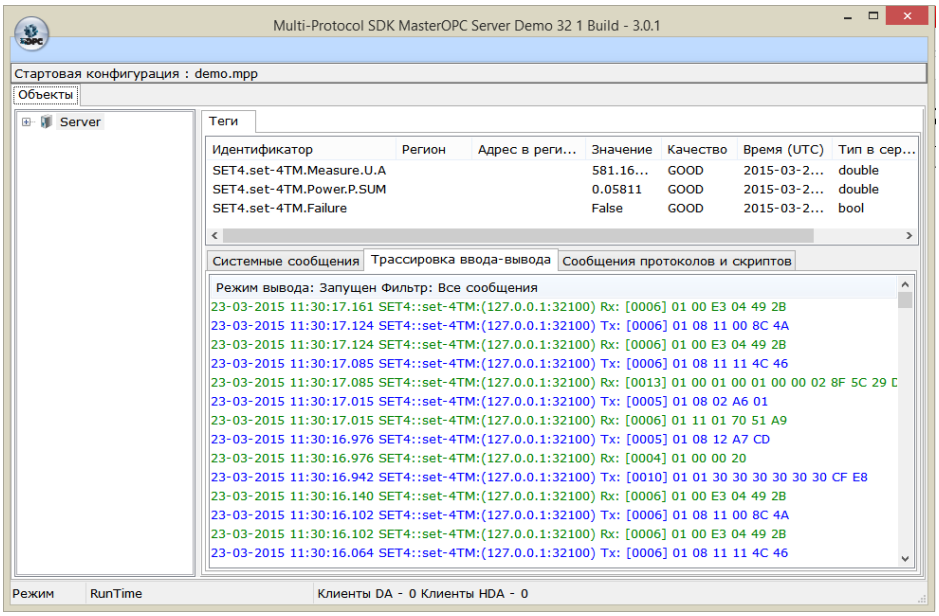
5.2.4. Рекомендации по отладке

Можно отлаживать проект средствами ***Microsoft Visual Studio 2013***. Нажатием на кнопку ***Локальный отладчик Windows*** компилируется проект и запускается ***mpssdk.exe***, можно устанавливать точки останова, смотреть значения переменных и т.д.

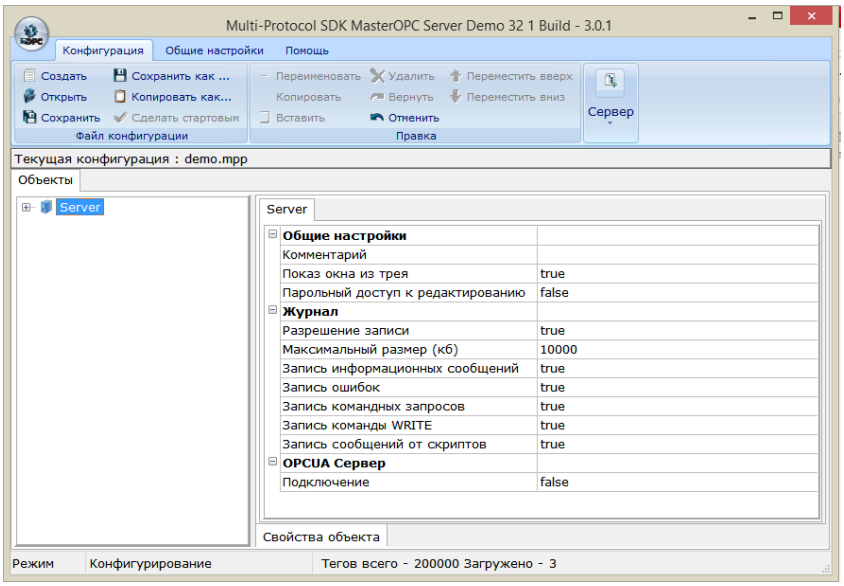
Кроме того, в коде плагина можно генерировать сообщения, отображающие состояние работы – для вывода таких сообщений используйте процедуру `mps_DebugMessage`. Сообщения будут отображаться на закладке ***Сообщения протоколов и скриптов*** в режиме исполнения OPC сервера.



На вкладке ***Трассировка ввода-вывода*** пишутся запросы, посылаемые сервером на счетчик и ответы от него.



В режиме конфигурирования в корне сервера можно установить запись в журнал сообщений (ведение лог-файлов). В ***«Журнал – Разрешение записи»*** установите ***true***, максимальный размер в ***10000***, остальные пункты выбора тоже нужно установить в ***true***.



Лог-файл будет писаться в папку ***C:\ProgramData\InSAT\Multi-Protocol SDK MasterOPC Server\SERVERLOGS***

6. Создание OPC-сервера HDA на примере счетчика электроэнергии

6.1. Типовой алгоритм записи в HDA тег

Сначала считываем из архива устройства время и значение, согласно алгоритму протокола. При отсутствии связи считывать архивные данные с устройства не нужно.

Формируем метку времени в формате ***TimeStamp*** (дата, время, количество миллисекунд) - упаковываем его функцией `PackSystemTime`. Пример функции приведен в приложении 4.

```
OpcMP_plugin_timestamp_t  ts = PackSystemTime (int wYear, int wMonth, int wDay, int
wHour, int wMinute, int wSecond, int wMilliseconds) //Пример функции упаковки времени
```

Записываем считанное значение `Val`, качество и полученную метку времени `ts` в тег – в его DA часть (не архивную). Если устройство возвращает достоверные данные, записываем качество `OPCMP_QUALITY_GOOD.`

```
mps_setTagAsDouble(opcmp_tag, Val, OPCMP_QUALITY_GOOD, &ts);
```

Теперь запишем значение в архив - в HDA раздел тега.

```
mps_WriteTagToHDA(Plugin, opcmp_tag); // записываем тег DA в HDA
```

Если прибор возвращает признак недостоверности архивной записи, тегу нужно установить качество `OPCMP_QUALITY_BAD`.

При отсутствии связи с устройством в DA часть тэга необходимо записать качество `OPCMP_QUALITY_BAD`, при этом значение тэга менять не нужно.

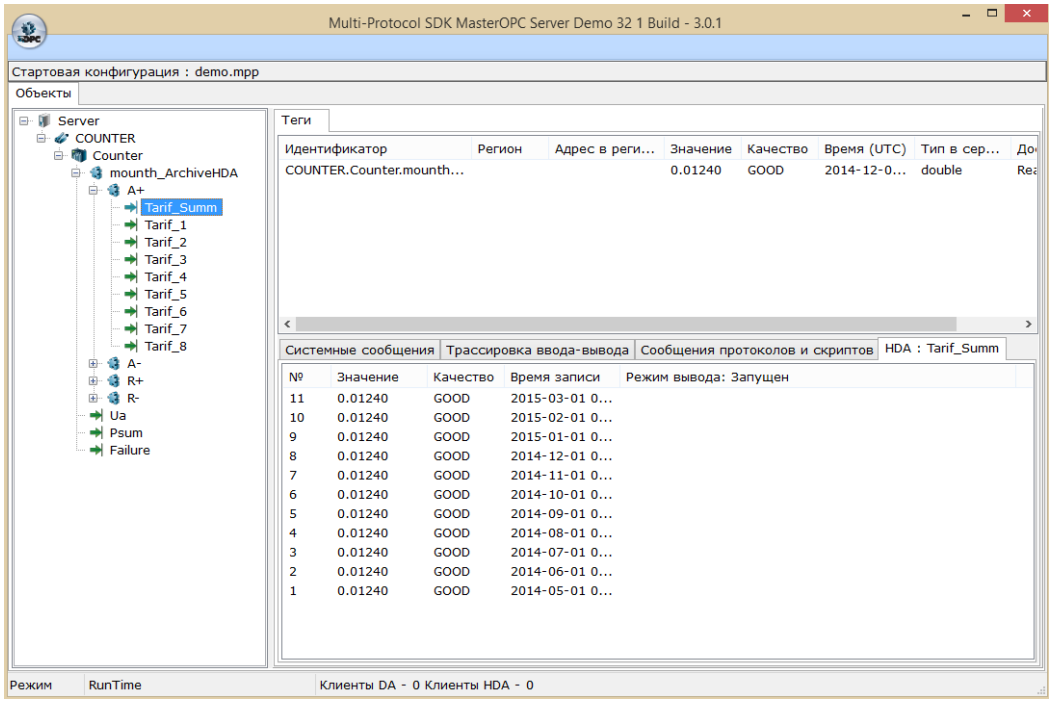
```
mps_setTagQwality(opcmp_tag, OPCMP_QUALITY_BAD);
```

Однако писать такое значение в HDA часть тега как правило не следует – так как OPC клиент считает данное значение и запишет в свой архив значение с плохим признаком качества. Но если OPC сервер впоследствии сможет считать архивное значение (например, восстановится связь с прибором), то передать его в OPC клиенту уже не получится – так как значение с этой меткой времени уже записано.

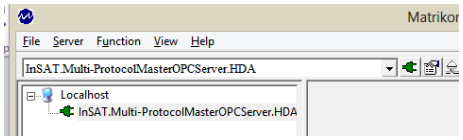
Пример чтения энергий за 12 месяцев со счетчика электроэнергии приведен в приложении 5.

6.2. Просмотр архивов

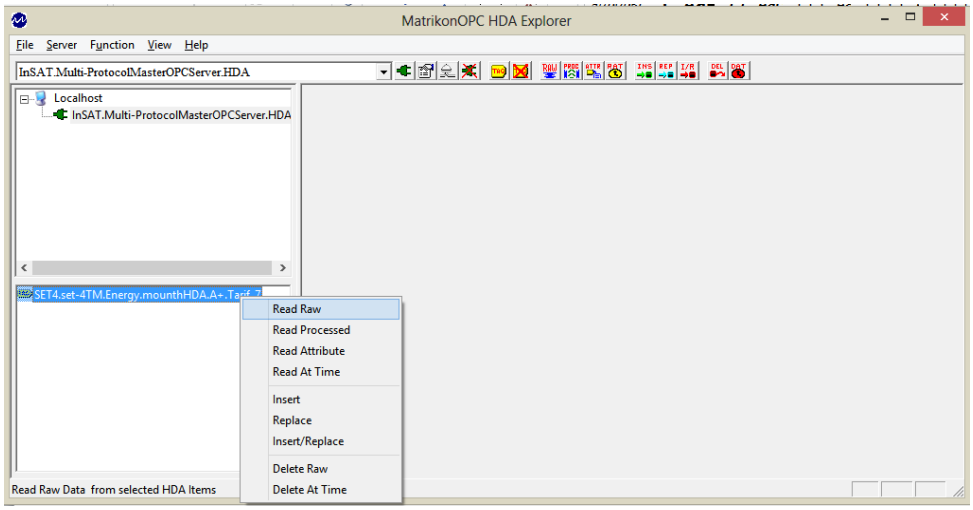
В ***Multi-Protocol SDK MasterOPC Server*** архив можно посмотреть, выделив мышкой тэг и открыв вкладку HDA.



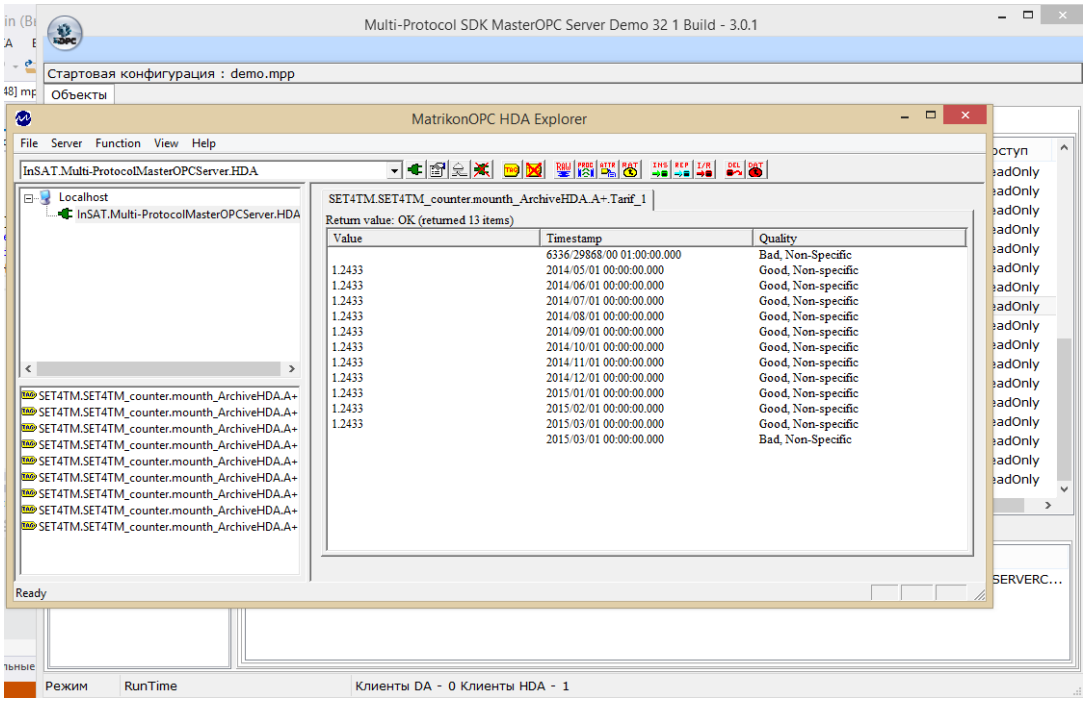
Так же архивы можно смотреть с помощью OPC HDA клиента, например ***Matrikon OPC HDA Explorer***. Для этого запустите его. Выберите HDA OPC-сервер:



Нажмите кнопку ***Connect***. Добавьте тэги. Выделите нужный тэг, нажмите правую кнопку мыши и выберите ***Read Raw***



В появившемся окне выберите ***Start Time*** и ***End Time*** и нажмите ***Read Raw***.

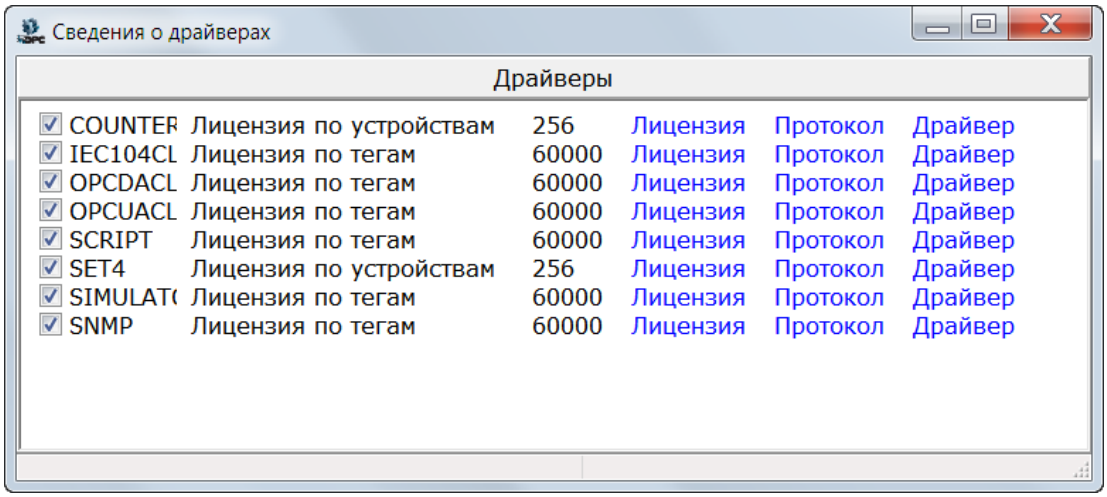

Multi-Protocol SDK MasterOPC Server Demo 32 1 Build - 3.0.1
Стартовая конфигурация : demo.mpp
Объекты
MatrikonOPC HDA Explorer
File Server Function View Help
InSAT.Multi-ProtocolMasterOPCServer.HDA
Localhost
InSAT.Multi-ProtocolMasterOPCServer.HDA
SET4TM.SET4TM_counter.mounth_ArchiveHDA.A+.Tarif_1
Return value: OK (returned 13 items)
Value Timestamp Quality
6336/29868/00 01:00:00.000 Bad, Non-Specific
1.2433 2014/05/01 00:00:00.000 Good, Non-specific
1.2433 2014/06/01 00:00:00.000 Good, Non-specific
1.2433 2014/07/01 00:00:00.000 Good, Non-specific
1.2433 2014/08/01 00:00:00.000 Good, Non-specific
1.2433 2014/09/01 00:00:00.000 Good, Non-specific
1.2433 2014/10/01 00:00:00.000 Good, Non-specific
1.2433 2014/11/01 00:00:00.000 Good, Non-specific
1.2433 2014/12/01 00:00:00.000 Good, Non-specific
1.2433 2015/01/01 00:00:00.000 Good, Non-specific
1.2433 2015/02/01 00:00:00.000 Good, Non-specific
1.2433 2015/03/01 00:00:00.000 Good, Non-specific
2015/03/01 00:00:00.000 Bad, Non-Specific
SET4TM.SET4TM_counter.mounth_ArchiveHDA.A+
Ready
Режим RunTime Клиенты DA - 0 Клиенты HDA - 1

7. Перенос плагина в Multi-Protocol MasterOPC Server

Multi-Protocol SDK MasterOPC сервер – это OPC сервер для разработчика. Данный OPC сервер не имеет ограничений на количество добавляемых тегов и не имеет ограничений на время работы в режиме исполнения, но имеет ограничение на количество передаваемых стороннему OPC клиенту тегов – не более 32. Это позволяет эффективно использовать его для разработки и отладки плагинов, но для использования с OPC клиентами (SCADA системами) необходимо использовать ***Multi-Protocol MasterOPC сервер*** – плагин ***User Protocol***.

Для переноса разработанного плагина скопируйте файл плагина ***mpsplugin-<имя протокола>.spl*** и соответствующий PCF-файл в папку ***C:\ProgramData\InSAT\Multi-Protocol MasterOPC Server\SERVERPLUGINS\INPOUT***.

После перезапуска Multi-Protocol MasterOPC сервера данный плагин появится в списке доступных драйверов. Теперь плагин можно добавлять в пользовательскую конфигурацию.



Разработанный плагин относится к плагину ***User Protocol***. Плагин лицензируется по количеству добавленных тегов. Если в конфигурации используется несколько разных плагинов написанных пользователем, то количество тегов каждого плагина суммируется.

Стоимость лицензии можно узнать на странице продукта.

Приложение 1. Вычисление контрольной суммы ModBus

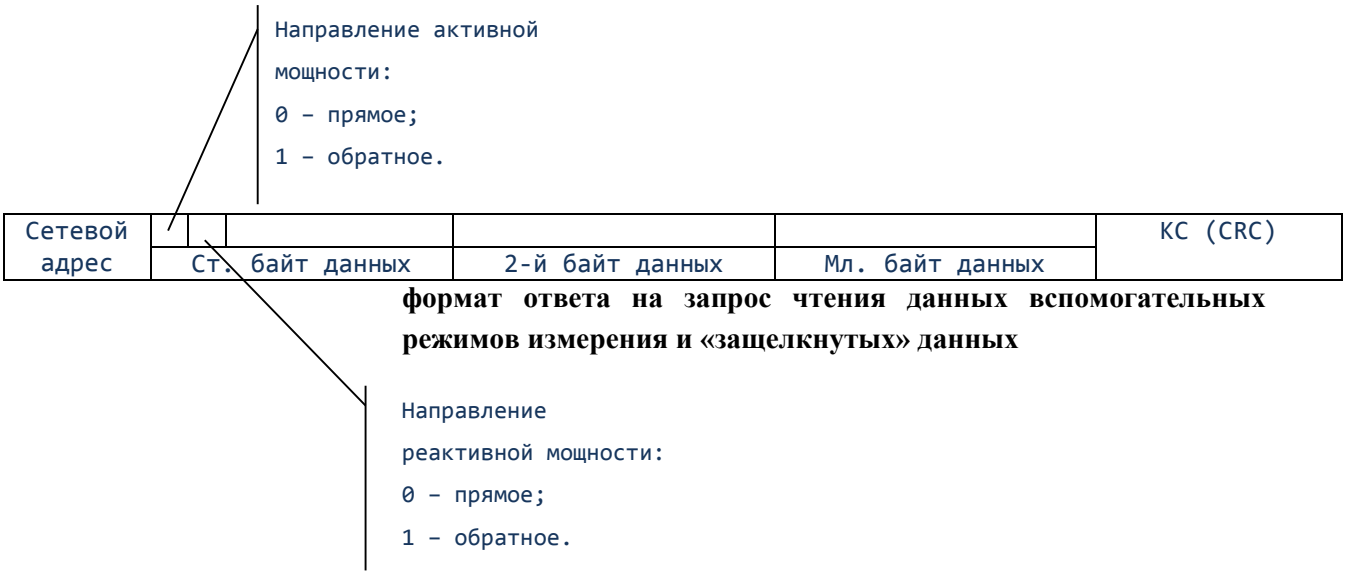
Расчет контрольной суммы Modbus находится в файле ***dcon.cpp***

```
//вычисление контрольной суммы CRC. 2 байта
WORD Crc16(byte * Data2, ULONG size)
{
	BYTE Data[256];
	int i;
	for (i = 0; i < size; i++)
	{
		Data[i] = Data2[i];

	};

	union  { BYTE b[2]; unsigned short w; } Sum;
	char shift_cnt;  BYTE *ptrByte;
	ULONG byte_cnt = size;
	ptrByte = Data;
	Sum.w = 0xffffU;
	for (; byte_cnt>0; byte_cnt--)
	{
		Sum.w = (unsigned short)((Sum.w / 256U) * 256U + ((Sum.w % 256U) ^
(*ptrByte++)));
		for (shift_cnt = 0; shift_cnt<8; shift_cnt++)
		{/*обработка байта*/   if ((Sum.w & 0x1) == 1)   Sum.w = (unsigned
short)((Sum.w >> 1) ^ 0xa001U);  else   Sum.w >>= 1;
		}
	}
	return Sum.w;
}
```

Приложение 2. Вычисление активной мощности

При запросе значения активной мощности счетчик возвращает слово из трех байт в двоичном коде. Два старших бита старшего байта указывают положение вектора полной мощности и должны маскироваться для получения модуля значения параметра.

Формат ответа на запрос чтения вспомогательных параметров («защелкнутых» данных):



формат ответа на запрос чтения данных вспомогательных режимов измерения и «защелкнутых» данных

Значения считанных данных должны интерпретироваться в соответствии с формулой:

$$P,Q,S(Вт,вар,ВА) = \frac{Np,q,s}{1000} \cdot Кн \cdot Кт \cdot Кс$$

Где: Np,q,s - трехбайтный код ответа на запрос соответствующих физических величин с отмаскированными битами направления;

Кн – коэффициент трансформации трансформатора напряжения;

Кт - коэффициент трансформации трансформатора тока;

Кс – коэффициент Кс (зависит от типа счетчика).

В нашем учебном примере примем Кн=Кт=Кс=1. Код вычисления активной мощности находится в файле ***plugin-common.cpp***.

```
if (Zpr->TypeAnswer == 1)//это значение мы забили в п. 5.1.
{
       b[1] = RecvBuffer[1];
       b[2] = RecvBuffer[2];
       b[3] = RecvBuffer[3];
    NaprP = b[1] & 0x80;
       if (NaprP > 0)//Направление активной мощности
       {
              NaprP = 1;
```

```
            }
            else
            {
                NaprP  = 0;
            }

            NaprQ = b[1] & 0x40;
            if (NaprQ > 0)//Направление реактивной мощности
            {
                  NaprQ = 1;
            }
            else
            {
                  NaprQ = 0;
            }
            b[1] = b[1] & 0x3F;
            DRes = b[1] * (0xFFFF + 1) + b[2] * (0xFF + 1) + b[3];//значение
            Znak = 1;

                  if (NaprP > 0)
                  {
                        Znak = -1;
                  }
                  else
                  {
               Znak  = 1;
                  }

            dUnpackAnswer = (Znak * (DRes / 1000))/ 1000;//Делим на 1000, т.к. у нас
размерность кВт

      }//if (Zpr->TypeAnswer == 1)
```

Приложение 3. Вычисление напряжения

Код вычисления напряжения находится в файле ***plugin-common.cpp***

```
if (Zpr->TypeAnswer == 2) //В это значение мы забили в п. 5.1.
	{
		b[1] = RecvBuffer[1];
		b[2] = RecvBuffer[2];
		b[3] = RecvBuffer[3];
		DRes = b[1] * (0xFFFF + 1) + b[2] * (0xFF + 1) + b[3]; //значение

		dUnpackAnswer = ((DRes / 1000) * Zpr->Ku)/ 100;
	}
```

Приложение 4. Пример функции упаковки времени

Код упаковки времени находится в файле ***dcon.cpp***.

```
OpcMP_plugin_timestamp_t PackSystemTime(int wYear, int wMonth, int wDay, int wHour, int
wMinute, int wSecond, int wMilliseconds)
{
	SYSTEMTIME systime;
	FILETIME LocalFileTime, FileTime;
	OpcMP_plugin_timestamp_t ft = { 0 };
	memset(&systime, 0, sizeof(systime));
	systime.wYear = wYear;
	systime.wMonth = wMonth;
	systime.wDay = wDay;
	systime.wHour = wHour;
	systime.wMinute = wMinute;
	systime.wSecond = wSecond;
	systime.wMilliseconds = wMilliseconds;
	::SystemTimeToFileTime(&systime, &FileTime);
	::LocalFileTimeToFileTime(&FileTime, &LocalFileTime);
	::FileTimeToSystemTime(&LocalFileTime, &systime);
	::SystemTimeToFileTime(&systime, (FILETIME *)&ft);
	return  ft;
}
```

Приложение 5. Пример чтения энергий за 12 месяцев со счетчика электроэнергии

Код чтения архива энергий находится в файле ***dcon.cpp***.

```
//Чтение энергий на месяц 1-12
bool ReadE_mounth(OpcMP_plugin_t *Plugin, struct Zapros *Zpr, USList *readdev, int ttyp,
int N_mounth)
{
	int   Rez, LenZapros;
	unsigned short int QKind, Addr;


	WORD  WORD_CRC;
	//byte byte_h, byte_l;
	static byte byte_h, byte_l;
	char string[6] = "";
	QKind = Zpr->QKind;
	Addr = Zpr->Address;
	mps_DebugMessage(Plugin, "Чтение энергий на месяц 1-12");
	//перебираем 12 месяцев
	for (int i = 1; i <= 12; i++)
	{
	    BufferTo[0] = Addr;
		BufferTo[1] = 0x05; // Код запроса
		BufferTo[2] = (0x03 << 4) + i;///  Код параметра № массива     № месяца
		BufferTo[3] = ttyp;// № тарифа
		LenZapros = 4;//длина запроса


		WORD_CRC = Crc16(BufferTo, LenZapros);
		byte_h = WORD_CRC >> 8;
		byte_l = WORD_CRC & 0x00FF;

		BufferTo[LenZapros] = byte_l;
		BufferTo[LenZapros + 1] = byte_h;

		Rez = SendBlockToRs(Plugin, Zpr->Gaddress,
			BufferTo, LenZapros + 2,
			BufferFrom, 57,
			3);
	if (Rez <= 2)
	{
		mps_DebugMessage(Plugin, "счетчик не ответил  ");
		Zpr->Status = -1;
		return false;
	};
	if (Rez > 2) //проверка контрольной суммы
	{

			WORD_CRC = Crc16(BufferFrom, Rez - 2);
			byte_h = WORD_CRC >> 8;
			byte_l = WORD_CRC & 0x00FF;
			if ((byte_l == BufferFrom[Rez - 2]) && (byte_h == BufferFrom[Rez -
1]))
			{
				Zpr->Status = 1;

			}
			else
			{
```

```
                        Zpr->Status = -1;
                        mps_DebugMessage(Plugin, "ошибка контрольной суммы CRC");
                        return false;
                }

        }
    };

    if (BufferFrom[0] != Addr)  //Первый байт ответа всегда адрес счетчика. Если нет,
то что-то не то!!!
    {
        Zpr->Status = -1;
        mps_DebugMessage(Plugin, "Первый байт ответа не адрес счетчика");
        Zpr->Status = -1;
        return false;
    };
    if (Rez > 0)
    {
        //Zpr->Status = 1;
        if (Rez <= 5)
        {

            if (BufferFrom[1] > 0)  //Если Значение байта состояния обмена не 0,
то есть какая-то ошибка!
            {
                mps_DebugMessage(Plugin, "Счетчик вернул ошибку Чтение
энергий на месяц 1-12");

            }
        }
        else
        {
            if (Rez < 170)
            {

                byte RecvBuffer[170];
                INT64 E[5];
                double E_peresch[5];   //Массивы энергий готовые на вывод.
                double E_m[5];
                for (int i2 = 0; i2 < Rez; i2++)
                    RecvBuffer[i2] = BufferFrom[i2];


                E[1] = (RecvBuffer[1] << 24) + (RecvBuffer[2] << 16) +
(RecvBuffer[3] << 8) + (RecvBuffer[4]);
                E[2] = (RecvBuffer[5] << 24) + (RecvBuffer[6] << 16) +
(RecvBuffer[7] << 8) + (RecvBuffer[8]);
                E[3] = (RecvBuffer[9] << 24) + (RecvBuffer[10] << 16) +
(RecvBuffer[11] << 8) + (RecvBuffer[12]);
                E[4] = (RecvBuffer[13] << 24) + (RecvBuffer[14] << 16) +
(RecvBuffer[15] << 8) + (RecvBuffer[16]);

                if (Zpr->A <= 0)
                {
                    Zpr->A = 1;
                    return false;
                }

                E_peresch[1] = E[1];
                E_peresch[2] = E[2];
                E_peresch[3] = E[3];
                E_peresch[4] = E[4];
```

```
                        for (int h = 0; h < 5; h++)
                        {
                    E_m[h] = (E_peresch[h] / (2 * Zpr->A)) * Zpr->Ku * Zpr->Ki;
                        }
                        SYSTEMTIME systime;
                        int  wwyear,wwmounth;
                        GetLocalTime(&systime);
                        if (systime.wMonth > i)
                        {
                                wwyear = systime.wYear;
                        }
                        else
                        {
                                wwyear = systime.wYear-1;
                        }
                        wwmounth = systime.wMonth;
                        OpcMP_plugin_timestamp_t  ts = PackSystemTime(wwyear, i, 1,
0, 0, 0, 0);//упаковываем нужное время
                        if (readdev != NULL)
                        for (int ip = 0; ip < readdev->count(); ip++)
                        {
                                OpcMP_tag_t *opcmp_tag = (OpcMP_tag_t *)readdev-
>getItem(ip);
                                my_tagdata * mytagdata = (my_tagdata
*)mps_getTagData(opcmp_tag);
                                ///for (int ip = 0; ip < readdev->count(); ip++)
                                for (int p = 1; p <= 4; p++)
                                if (mytagdata->NumberChannel == (14000+ttyp+100*p))
                                {
                                        if (N_mounth == 0)
                                        {
                                                if (i != wwmounth)
                                                {

                                        mps_setTagAsDouble(opcmp_tag, E_m[p],
OPCMP_QUALITY_GOOD, &ts);//// записываем значение,качество GOOD и  полученное время в тег
                                        mps_WriteTagToHDA(Plugin, opcmp_tag); //
записываем тег в HDA
                                                }
                                        }
                                        else
                                        {
                                                if (N_mounth == i)
                                                {
                                                        mps_setTagAsDouble(opcmp_tag,
E_m[p], OPCMP_QUALITY_GOOD, &ts);//// записываем значение,качество GOOD и  полученное
время в тег
                                                        mps_WriteTagToHDA(Plugin,
opcmp_tag); // записываем тег в HDA
                                                }
                                        }

                                }
                        }



                        Zpr->LenBuffer = Rez;
                        //return true;
                }
```

```
                    else
                    {
                            Zpr->Status = -1;
                            Zpr->LenBuffer = 0;
                            return false;
                    }

            }
        }
}
        return true;
};
```

Приложение 6. Работа с COM-портом

Чтобы разобраться с работой с COM-портом откроем проект ***mpsplugin-dcondemo***, файл ***plugin-common.cpp***. И найдем функцию

`int SendBlockToRs(OpcMP_plugin_t *Plugin, int port, char *SourceBuffer, int SourceLen, char *DestBuffer, int DestLen, int TimeOut1).`

Эта функция отправляет в порт данные `SourceBuffer` длиной `SourceLen` и принимает в ответ данные в `DestBuffer` длиной `DestLen.`

Функция написана для протокола DCON. Немного переделаем ее. В качестве входного и выходного буфера лучше использовать тип `byte,` а не `unsigned char.` Функция API `sendandreceive` использует `unsigned char,` поэтому будем переводить `byte` в `unsigned char` и наоборот.

```
int SendBlockToRs(OpcMP_plugin_t *Plugin, int port, byte *SourceBuffer, int
SourceLen, byte *DestBuffer, int DestLen, int TimeOut1)
{
   int result;
   unsigned char DestBuffer2[2048];
   unsigned char SourceBuffer2[2048];
   mps_sendandreceive_t sendandreceive;
   if (Plugin == NULL)
          return -1;
   sendandreceive = mps_getSendAndReceive(Plugin);
   if (sendandreceive == NULL)
          return -1;
   my_plugin_data * data = (my_plugin_data*)mps_getPluginData(Plugin);
   for (int i = 0; i <= SourceLen; i++)
   {
          SourceBuffer2[i] = SourceBuffer[i];

   }
   result = sendandreceive(Plugin, (unsigned char *)SourceBuffer2, SourceLen,
(unsigned char *)DestBuffer2, DestLen);
   for (int i = 0; i < result; i++)
   {
          DestBuffer[i] = DestBuffer2[i];

   }
   return result;
}
```

Приложение 7. Константы признаков качества

Константа	Назначение
OPCMP_QUALITY_CONFIG_ERROR	Ошибка конфигурации
OPCMP_QUALITY_NOT_CONNECTED	Нет соединения
OPCMP_QUALITY_DEVICE_FAILURE	Ошибка устройства
OPCMP_QUALITY_SENSOR_FAILURE	Ошибка датчика
OPCMP_QUALITY_LAST_KNOWN	Последнее известное значение
OPCMP_QUALITY_COMM_FAILURE	Нет связи
OPCMP_QUALITY_OUT_OF_SERVICE	Не обслуживается
OPCMP_QUALITY_BAD	плохое качество тэга
OPCMP_QUALITY_GOOD	хорошее качество тэга