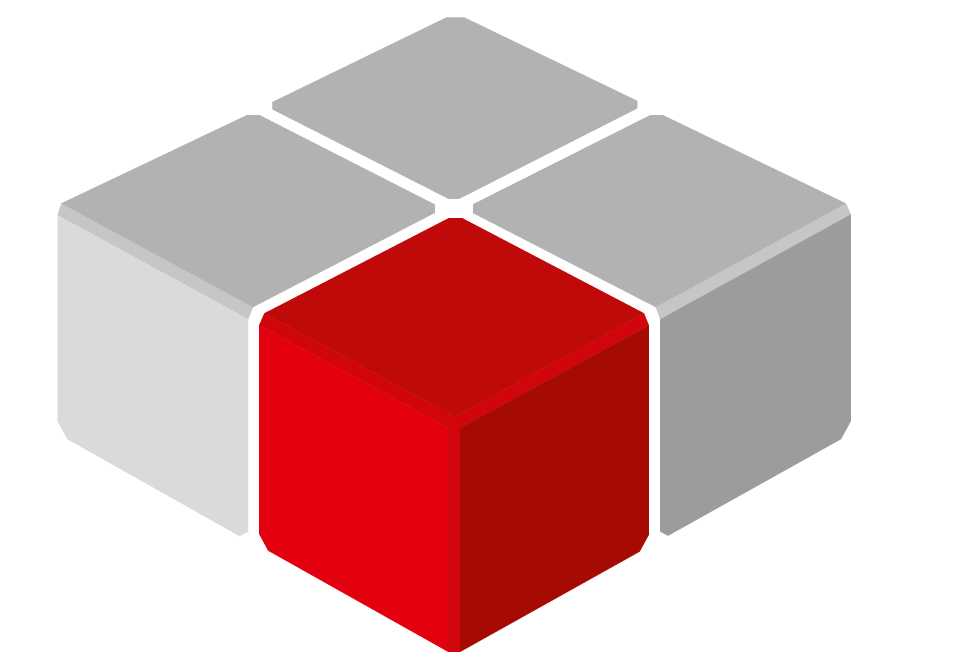


**CODESYS V3.5**

**Реализация обмена через сокеты**



Руководство пользователя

25.02.2020

версия 2.1

Оглавление

[Глоссарий 3](#_Toc33001150)

[1 Цель и структура документа 3](#_Toc33001151)

[2 Основные сведения о работе с сокетами 4](#_Toc33001152)

[2.1 Общая информация о сокетах 4](#_Toc33001153)

[2.2 Серверы и клиенты 5](#_Toc33001154)

[2.3 Протокол UDP 5](#_Toc33001155)

[2.4 Протокол TCP 6](#_Toc33001156)

[2.5 Вопросы информационной безопасности 7](#_Toc33001157)

[2.6 Средства отладки 7](#_Toc33001158)

[2.7 Средства для работы с сокетами в CODESYS 7](#_Toc33001159)

[3 Библиотека CAA Net Base Services 8](#_Toc33001160)

[3.1 Добавление библиотеки в проект CODESYS 8](#_Toc33001161)

[3.2 Структуры и перечисления 9](#_Toc33001162)

[3.2.1 Структура NBS.IP\_ADDR 9](#_Toc33001163)

[3.2.2 Перечисление NBS.ERROR 9](#_Toc33001164)

[3.3 ФБ работы с протоколом UDP 10](#_Toc33001165)

[3.3.1 ФБ NBS.UDP\_Peer 10](#_Toc33001166)

[3.3.2 ФБ NBS.UDP\_Receive 11](#_Toc33001167)

[3.3.3 ФБ NBS.UDP\_Send 12](#_Toc33001168)

[3.3.4 ФБ NBS.UDP\_ReceiveBuffer 13](#_Toc33001169)

[3.3.5 ФБ NBS.UDP\_SendBuffer 14](#_Toc33001170)

[3.4 ФБ работы с протоколом TCP 15](#_Toc33001171)

[3.4.1 ФБ NBS.TCP\_Server 15](#_Toc33001172)

[3.4.2 ФБ NBS.TCP\_Connection 16](#_Toc33001173)

[3.4.3 ФБ NBS.TCP\_Сlient 17](#_Toc33001174)

[3.4.4 ФБ NBS.TCP\_Read 18](#_Toc33001175)

[3.4.5 ФБ NBS.TCP\_Write 19](#_Toc33001176)

[3.4.6 ФБ NBS.TCP\_ReadBuffer 20](#_Toc33001177)

[3.4.7 ФБ NBS.TCP\_WriteBuffer 21](#_Toc33001178)

[3.5 Дополнительные функции 22](#_Toc33001179)

[3.5.1 Функция NBS.IPSTRING\_TO\_UDINT 22](#_Toc33001180)

[3.5.2 Функция NBS.IPSTRING\_TO\_UDINT 22](#_Toc33001181)

[3.5.3 Функция NBS.IS\_MULTICAST\_GROUP 23](#_Toc33001182)

[4 Примеры работы с библиотекой CAA Net Base Services 24](#_Toc33001183)

[4.1 Краткое описание примеров 24](#_Toc33001184)

[4.2 Реализация UDP-сервера и UDP-клиента 25](#_Toc33001185)

[4.2.1 Основная информация 25](#_Toc33001186)

[4.2.2 Реализация UDP-сервера 25](#_Toc33001187)

[4.2.3 Реализация UDP-клиента 29](#_Toc33001188)

[4.3 Реализация TCP-сервера и TCP-клиента 32](#_Toc33001189)

[4.3.1 Основная информация 32](#_Toc33001190)

[4.3.2 Реализация TCP-сервера 32](#_Toc33001191)

[4.3.3 Реализация TCP-клиента 36](#_Toc33001192)

[4.4 Работа с примером 39](#_Toc33001193)

[4.5 Рекомендации и замечания 40](#_Toc33001194)

[5 Библиотека OwenCommunication 41](#_Toc33001195)

[5.1 Общая информация 41](#_Toc33001196)

[5.2 ФБ UNM\_TcpRequest 42](#_Toc33001197)

[5.3 ФБ UNM\_UdpRequest 44](#_Toc33001198)

[Приложение А. Листинг примера UDP 46](#_Toc33001199)

[А.1. UDP-сервер 46](#_Toc33001200)

[А.2.1. Перечисление SERVER\_STATE 46](#_Toc33001201)

[А.2.2. Функция MIRROR 46](#_Toc33001202)

[А.2.3. Программа PLC\_PRG 47](#_Toc33001203)

[А.2. UDP-клиент 50](#_Toc33001204)

[А.2.1. Перечисление CLIENT\_STATE 50](#_Toc33001205)

[А.2.2. Программа PLC\_PRG 50](#_Toc33001206)

[Приложение Б. Листинг примера TCP 53](#_Toc33001207)

[Б.1. TCP-сервер 53](#_Toc33001208)

[Б.1.1. Перечисление SERVER\_STATE 53](#_Toc33001209)

[Б.1.2. Структура CONNECTION 53](#_Toc33001210)

[Б.1.3. Функция MIRROR 54](#_Toc33001211)

[Б.1.4. Программа PLC\_PRG 55](#_Toc33001212)

[Б.2. TCP-клиент 58](#_Toc33001213)

[Б.2.1. Перечисление CLIENT\_STATE 58](#_Toc33001214)

[Б.2.2. Программа PLC\_PRG 58](#_Toc33001215)

# Глоссарий

**ПЛК** – программируемый логический контроллер.

**ФБ** – функциональный блок.

# Цель и структура документа

Одним из современных трендов промышленной автоматизации является повсеместное внедрение интерфейса [Ethernet](https://ru.wikipedia.org/wiki/Ethernet) и использование для обмена данными между устройствами протоколов, основанных на стеке [TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP) – Modbus TCP, [KNX](https://ru.wikipedia.org/wiki/KNX), [MQTT](https://ru.wikipedia.org/wiki/MQTT), [SNMP](https://ru.wikipedia.org/wiki/SNMP) и др.

Другой тенденцией является расширение коммуникационных задач ПЛК: помимо опроса устройств и передачи данных на верхний уровень (в OPC-серверы или SCADA-системы) возникает потребность в передаче файлов (например, по [FTP](https://ru.wikipedia.org/wiki/FTP)), синхронизации данных с серверами точного времени ([NTP](https://ru.wikipedia.org/wiki/NTP)), рассылке сообщений по электронной почте ([SMTP](https://ru.wikipedia.org/wiki/SMTP)/[POP3](https://ru.wikipedia.org/wiki/POP3)) и т. д.

Некоторые ПЛК имеют готовые компоненты, предназначенные для решения конкретных задач. Такие компоненты просты и удобны в использовании, но зачастую требуют покупки отдельной лицензии. Кроме того, набор доступных компонентов далеко не всегда соответствует потребностям пользователя.

Данное руководство описывает настройку передачи данных с помощью сетевых протоколов **UDP** и **TCP** для контроллеров ОВЕН, программируемых в **CODESYS V3.5**

Среда **CODESYS V3.5** предоставляет возможность работы с [сетевыми сокетами,](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)) что позволяет программисту реализовать свой собственный протокол обмена поверх стандартных протоколов **UDP** или **TCP**. Для этого требуется:

* понимание основ сетевого взаимодействия систем;
* хорошие навыки программирования на языке ST;
* спецификация реализуемого протокола.

В [п. 2](#_2._Основные_сведения_1) приведена основная информация о работе с сокетами.

В [п. 3](#_3._Библиотека_CAA) приведено описание библиотеки **CAA Net Base Services.**

В [п. 4](#_4._Примеры_работы) рассмотрены примеры использования библиотеки.

В [п. 5](#_Библиотека_OwenCommunication) приведена информация о библиотеке **OwenCommunication**.

Документ рекомендуется читать строго последовательно.

# Основные сведения о работе с сокетами

## Общая информация о сокетах

[Сокет](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)) – это программный интерфейс, который обеспечивает обмен данными между процессами. Данный документ посвящен сетевым сокетам. Сетевые сокеты позволяют организовать обмен данными между процессами, которые выполняются на разных устройствах. Примером такого процесса может являться пользовательская программа, выполняемая ПЛК. Формат передачи данных между двумя устройствами зависит от используемого [протокола обмена](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

С точки зрения пользователя сокет представляет собой пару «IP-адрес – порт». [IP-адрес](https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81) позволяет идентифицировать сетевой адаптер конкретного устройства, а [порт](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D1%81%D0%B5%D1%82%D0%B8)) – конкретное приложение этого устройства. Примером таких приложений, например, могут быть Modbus TCP Slave и web-сервер, обслуживающий web-визуализацию. Фактически порт представляет собой целое число в диапазоне **1…65535**. В большинстве случаев порт стандартизирован на уровне используемого протокола обмена. В [данной статье](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BF%D0%BE%D1%80%D1%82%D0%BE%D0%B2_TCP_%D0%B8_UDP) приведен список портов, используемых различными протоколами, реализованными поверх **UDP** и **TCP**. Приложение может использовать несколько портов, но каждый порт в отдельно взятый момент времени может использоваться только одним приложением.

Список портов, используемых средой CODESYS и сервисами контроллера, приведен в руководстве **CODESYS V3.5. FAQ**.

Итак, тезисно подведем итоги данного подпункта:

* сокет характеризуется **IP-адресом** и **портом**. Зная их (а также протокол), можно организовать обмен данными с конкретным приложением конкретного устройства;
* среда исполнения **CODESYS** в процессе своей работы использует определенные порты ПЛК. Не следует пытаться занимать их другими процессами.

## Серверы и клиенты

Большинство сетевых протоколов основано на архитектуре «[клиент – сервер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80)». Фактически клиент и сервер являются приложениями, выполняемые на различных (в определенных случаях – на одном и том же) устройствах. Сокеты также разделяются на серверные и клиентские.

Сервер ожидает запросов от клиента, и в случае их получения выполняет заданные операции – после чего, в случае необходимости, отправляет клиенту ответ. Сервер не может являться инициатором обмена. Простейшим примером сервера и клиента являются web-сервер и web-браузер. Следует отметить, что один сервер может обслуживать множество клиентов.

Архитектура «клиент – сервер» достаточно близка к архитектуре «ведущий – ведомый» (Master – Slave), используемой при обмене данными по последовательной линии связи (RS-232/RS-485). Принципиальным отличием является то, что при сетевом обмене нет явного ограничения на число активных устройств (в случае использования последовательных интерфейсов в каждый момент времени активным является только одно устройство, регулирующее уровень сигнала на линии связи).

В настоящем руководстве рассматривается реализация сервера и клиента для протоколов UDP и TCP.

## Протокол UDP

[UDP](https://ru.wikipedia.org/wiki/UDP) (User Datagram Protocol) – простой протокол транспортного уровня [модели OSI](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI), не подразумевающий установки выделенного соединения между сервером и клиентом. Связь достигается путём передачи информации в одном направлении от источника к получателю без проверки готовности получателя. К основным характеристикам протокола относятся:

* *ненадёжность* — когда сообщение посылается, неизвестно, достигнет ли оно точки назначения или потеряется по пути. Нет таких понятий, как подтверждение, повторная передача, таймаут;
* *неупорядоченность* — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан;
* *легковесность* — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. UDP – это небольшой транспортный уровень, разработанный на [IP](https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81);
* *использование датаграмм* — пакеты посылаются по отдельности и проверяются на целостность только в том случае, если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано;
* *отсутствие контроля перегрузок* — UDP сам по себе не избегает перегрузок. Для приложений с большой пропускной способностью возможно вызвать коллапс перегрузок, если только они не реализуют меры контроля на прикладном уровне.

Как упоминалось выше, пакеты имеют определенные границы. Если размер пакета превышает эти границы, то он разбивается на несколько отдельных пакетов (фрагментируется). Не всё сетевое оборудование поддерживает работу с фрагментированными UDP-пакетами.

Для предотвращения фрагментации размер данных в пакете не должен превышать **1432 байт**, а для уверенности в том, что пакет сможет быть принят любым устройством – **508 байт**.

Протокол UDP поддерживает следующие схемы маршрутизации:

* [Unicast](https://ru.wikipedia.org/wiki/Unicast) – передача данных конкретному устройству;
* [Multicast](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%B2%D0%B5%D1%89%D0%B0%D0%BD%D0%B8%D0%B5) – передача данных группе устройств. Для этого устройство должно быть подписано на Multicast-группу, которая характеризуется IP-адресом. Для мультивещания зарезервирована подсеть **224.0.0.0 – 239.255.255.255**, при этом выделенные для частного использования адреса начинаются с **239.0.0.0**;
* [Broadcast](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D0%B5%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%B0%D0%BD%D0%B0%D0%BB) – передача данных всем устройствам данного сегмента сети. Для передачи должен использоваться последний IP-адрес сегмента. Например, в случае отправки UDP-пакета на адрес **10.2.11.255**, он будет доставлен устройствам с адресами **10.2.11.1 – 10.2.11.254**.

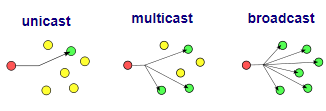


Рисунок 2.1 – Схемы маршрутизации UDP

## Протокол TCP

[TCP](https://ru.wikipedia.org/wiki/Transmission_Control_Protocol) (Transmission Control Protocol) – один из основных протоколов интернета, предназначенный для управления передачей данных. Протокол TCP выполняет функции протокола транспортного уровня [модели OSI](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI). Сети и подсети, в которых совместно используются протоколы TCP и IP, называются сетями TCP/IP. К основным характеристикам протокола относятся:

* *надежность* — TCP управляет подтверждением, повторной передачей и таймаутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется по пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае многочисленных таймаутов) разорванных соединений;
* *упорядоченность* — если два сообщения отправлены последовательно, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению;
* *тяжеловесность* — TCP необходимо три пакета для установки сокет-соединения перед тем, как отправить данные. TCP следит за надежностью и перегрузками;
* *потоковость* — данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов.

Принципиальным отличием TCP от UDP является необходимость установки соединения перед началом обмена данными.

## Вопросы информационной безопасности

В рамках данного документа не рассматриваются вопросы информационной безопасности и защищенной передачи данных. В качества источника информации по этому вопросу можно использовать документ [CODESYS Security Whitepaper](https://www.codesys.com/fileadmin/data/customers/security/CODESYS-Security-Whitepaper.pdf) и раздел [Security](https://www.codesys.com/support-training/codesys-support/codesys-security-information.html) сайта CODESYS.

## Средства отладки

В процессе отладки ПО, реализующего сетевой обмен, удобно использовать анализатор траффика [Wireshark](https://ru.wikipedia.org/wiki/Wireshark) и TCP/UDP-терминал [Hercules](https://www.hw-group.com/products/hercules/index_en.html) (для эмуляции сервера и клиента).

## Средства для работы с сокетами в CODESYS

В сети можно найти множество материалов по программированию сокетов на различных языках программирования. В качестве примера отметим [эту статью](http://rsdn.org/article/unix/sockets.xml). Работа с сокетами в Codesys происходит по тем же общим принципам.

В среде **CoDeSys 2.3** для работы с сокетами используется библиотека [SysLibSockets.](ftp://ftp.moeller.net/AUTOMATION/DOWNLOAD/MANUALS/ENGLISH/SOFTWARE/XSOFT_PROFESSIONAL/XSoftSysLibs/directory%20Pdf%20files/syslibsockets.pdf) Она содержит типичные функции, которые можно найти в подобных библиотеках для любого языка программирования (например, С) – connect(), bind(), accept() и т. д.

Хорошим источником информации по ее применению являются статьи [Войцеха Гомолка](https://www.researchgate.net/profile/Wojciech_Gomolka):

* [CoDeSys and Ethernet communication:The concept of Sockets and basic Function Blocks for communication over Ethernet. Part 1: UPD Client/Server](https://www.researchgate.net/publication/262198350_CoDeSys_and_Ethernet_communicationThe_concept_of_Sockets_and_basic_Function_Blocks_for_communication_over_Ethernet_Part_1_UPD_ClientServer)
* [The concept of Sockets and basic Function Blocks for communication over Ethernet. Part 2: TCP Server and TCP Client](https://www.researchgate.net/publication/264081315_The_concept_of_Sockets_and_basic_Function_Blocks_for_communication_over_Ethernet_Part_2_TCP_Server_and_TCP_Client)

В среде **CODESYS V3.5** аналогом этой библиотеки является библиотека **SysSocket**. Пример работы с ней описан Михаилом Шевцовым ([ПК Пролог](http://www.prolog-plc.ru/)) в видеоуроке [Программирование сокетов в CODESYS V3](https://www.youtube.com/watch?v=ThVLXygHnnU) и Ниной Кузьминой ([НПФ Доломант](http://www.dolomant.ru/)) в статье **Реализация TCP- и UDP-сокетов в среде разработки CODESYS V3** ([СТА № 3/2018](https://www.cta.ru/rubrics/239877.htm)).

Применение данной библиотеки может оказаться затруднительным для пользователей, не имеющих опыта работы с сокетами, и потребует определенных затрат времени даже для тех, у кого подобный опыт есть. Это стало одной из причин разработки и включения в состав CODESYS V3 библиотеки **CAA Net Base Services**. Эта библиотека реализована на более высоком уровне абстракции и представляет собой обвязку вокруг стандартных функций работы с сокетами, предоставляя пользователю удобный и емкий программный интерфейс. Для создания сетевой части серверного или клиентского приложения в данном случае достаточно будет использовать всего несколько функциональных блоков. Описание и примеры использования этой библиотеки приведены в настоящем руководстве.

Компания ОВЕН разработала библиотеку **OwenCommunication**, которая еще в большей степени упрощает разработку нестандартного в том случае, если контроллер выступает в роли TCP- или UDP-клиента. Библиотека доступна на сайте [ОВЕН](https://www.owen.ua/) в разделе **CODESYS V3/Библиотеки и компоненты**.

Информация о библиотеке приведена в [п. 5](#_Библиотека_OwenCommunication).

# Библиотека CAA Net Base Services

## Добавление библиотеки в проект CODESYS

Библиотека **CAA Net Base Services** используется для обмена данными по протоколам UDP и TCP.

Для добавления библиотеки в проект **CODESYS** в **Менеджере библиотек** следует нажать кнопку **Добавить** и выбрать библиотеку **CAA Net Base Services**.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Версия библиотеки не должна превышать версию таргет-файла контроллера. В противном случае корректная работа контроллера не гарантируется. |

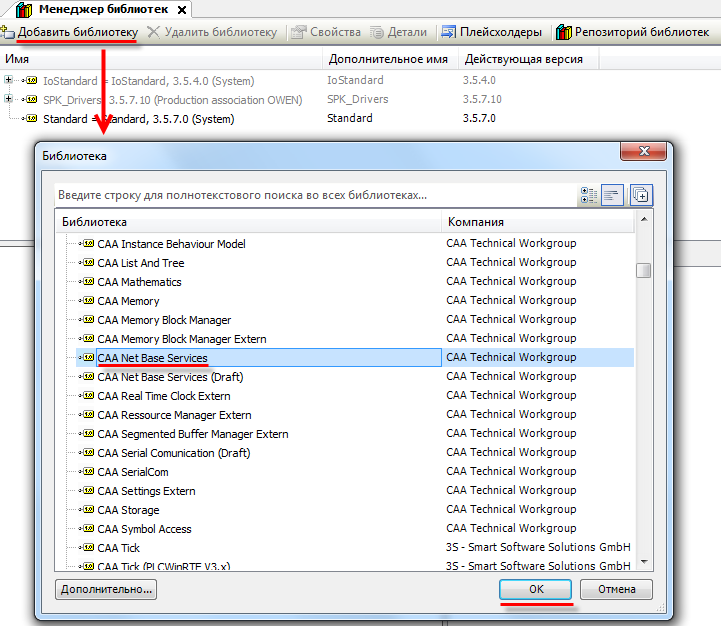


Рисунок 3.1 – Добавление библиотеки CAA Net Base Services в проект CODESYS

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  При объявлении экземпляров ФБ библиотеки следует перед их названием указывать префикс NBS**.** (пример: **NBS.**TCP\_Server). |

## Структуры и перечисления

### Структура NBS.IP\_ADDR

Структура **NBS.IP\_ADDR** описывает IP-адрес устройства.

**Таблица 3.1 – Переменные структуры NBS.IP\_ADDR**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| sAddr | STRING(80) | IP-адрес устройства в виде строки (например, ’10.2.11.10’) |

### Перечисление NBS.ERROR

Перечисление **NBS.ERROR** описывает ошибки, которые могут возникнуть при использовании ФБ библиотеки.

**Таблица 3.2 – Переменные перечисления NBS.ERROR**

|  |  |  |
| --- | --- | --- |
| **Название** | **Значение** | **Описание** |
| NO\_ERROR | 0 | Нет ошибок |
| TIME\_OUT | 6001 | Истек лимит времени для данной операции |
| INVALID\_ADDR | 6002 | На вход ФБ подан некорректный IP-адрес |
| INVALID\_HANDLE | 6003 | На вход ФБ подан некорректный дескриптор (handle) |
| INVALID\_DATAPOINTER | 6004 | На вход ФБ подан некорректный указатель |
| INVALID\_DATASIZE | 6005 | На вход ФБ подан некорректный размер данных |
| UDP\_RECEIVE\_ERROR | 6006 | Ошибка получения данных по UDP |
| UDP\_SEND\_ERROR | 6007 | Ошибка передачи данных по UDP |
| UDP\_SEND\_NOT\_COMPLETE | 6008 | Передача по UDP не была завершена – возможно, были отправлены не все данные |
| UDP\_OPEN\_ERROR | 6009 | Ошибка создания UDP-сокета |
| UDP\_CLOSE\_ERROR | 6010 | Ошибка закрытия UDP-сокета |
| TCP\_SEND\_ERROR | 6011 | Ошибка передачи данных по TCP |
| TCP\_RECEIVE\_ERROR | 6012 | Ошибка получения данных по TCP |
| TCP\_OPEN\_ERROR | 6013 | Ошибка создания TCP-сокета |
| TCP\_CONNECT\_ERROR | 6014 | Ошибка сервера при обработке соединения клиента |
| TCP\_CLOSE\_ERROR | 6015 | Ошибка закрытия TCP-сокета |
| TCP\_SERVER\_ERROR | 6016 | Ошибка TCP-сервера |
| WRONG\_PARAMETER | 6017 | ФБ вызван с некорректными аргументами |
| TCP\_NO\_CONNECTION | 6019 | Превышен лимит подключений к серверу |

## ФБ работы с протоколом UDP

### ФБ NBS.UDP\_Peer

Функциональный блок **NBS.UDP\_Peer** создает UDP-сокет и возвращает его дескриптор (**handle**), который используется для операций получения (ФБ [NBS.UDP\_Receive](#_3.3.2._ФБ_NBS.UDP_Receive), [NBS.UDP\_ReceiveBuffer](#_3.3.4._ФБ_NBS.UDP_ReceiveBuffer)) и передачи данных (ФБ [NBS.UDP\_Send](#_3.3.3._ФБ_NBS.UDP_Send), [NBS.UDP\_SendBuffer](#_3.3.5._ФБ_NBS.UDP_SendBuffer)).

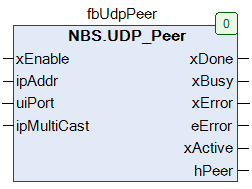


Рисунок 3.2 – Внешний вид ФБ NBS.UDP\_Peer на языке CFC

**Таблица 3.3 – Описание входов и выходов ФБ NBS. UDP\_Peer**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **ipAddr** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес устройства, на котором создается сокет |
| **uiPort** | UINT | Номер порта сокета |
| **uiMultiCast** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес мультикаст-группы.  В текущих версиях CODESYS мультикаст не поддерживается |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **xActive** | BOOL | Флаг «сокет успешно открыт». Пока он имеет значение **TRUE** – сокет открыт, и его можно использовать |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор открытого сокета устройства |

### ФБ NBS.UDP\_Receive

Функциональный блок **NBS.UDP\_Receive** используется для получения данных. Прослушиваемый порт задается при создании UDP-сокета с помощью ФБ [NBS.UDP\_Peer](#_3.3.1._ФБ_NBS.UDP_Peer).

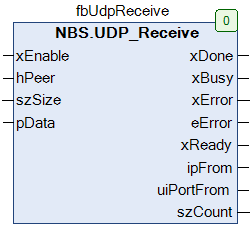


Рисунок 3.3 – Внешний вид ФБ NBS.UDP\_Receive на языке CFC

**Таблица 3.4 – Описание входов и выходов ФБ NBS. UDP\_Receive**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства, полученный от ФБ [NBS.UDP\_PEER](#_3.4.1._ФБ_FILE.DirCreate) |
| **szSize** | NBS.CAA.SIZE | Максимально допустимый размер получаемых данных в байтах. Может быть указан помощью оператора **SIZEOF** |
| **pData** | NBS.CAA.PVOID | Начальный адрес для размещения принятых данных. Может быть указан с помощью оператора **ADR** |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **xReady** | BOOL | Флаг «данные получены». Принимает значение **TRUE** на один цикл при получении нового пакета данных |
| **ipFrom** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес отправителя |
| **uiPortFrom** | UINT | Номер порта отправителя |
| **szCount** | NBS.CAA.SIZE | Размер принятых данных в байтах |

### ФБ NBS.UDP\_Send

Функциональный блок **NBS.UDP\_Send** используется для отправки данных на заданный IP-адрес/порт.

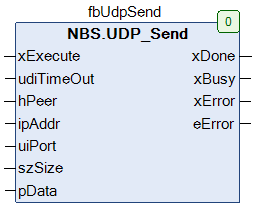


Рисунок 3.4 – Внешний вид ФБ NBS.UDP\_Send на языке CFC

**Таблица 3.5 – Описание входов и выходов ФБ NBS. UDP\_Send**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **udiTimeOut** | UDINT | Допустимое время операции (в мкс). Значение **0** означает, что время выполнения ФБ не ограничивается |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства, полученный от ФБ [NBS.UDP\_PEER](#_3.4.1._ФБ_FILE.DirCreate) |
| **ipAddr** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес получателя |
| **uiPort** | UINT | Номер порта получателя |
| **szSize** | NBS.CAA.SIZE | Размер отправляемых данных в байтах. Может быть указан помощью оператора **SIZEOF** |
| **pData** | NBS.CAA.PVOID | Начальный адрес отправляемых данных. Может быть указан с помощью оператора **ADR** |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |

### ФБ NBS.UDP\_ReceiveBuffer

Функциональный блок **NBS.UDP\_ReceiveBuffer** используется для получения данных. Прослушиваемый порт задается при создании UDP-сокета с помощью ФБ [NBS.UDP\_Peer](#_3.3.1._ФБ_NBS.UDP_Peer). В отличие от ФБ [NBS.UDP\_Receive](#_3.3.2._ФБ_NBS.UDP_Receive) данный блок не копирует данные по указателю, а возвращает дескриптор буфера, в котором они были размещены. Для работы с буфером используется библиотека **CAA SegBufMan**. Этот способ является менее ресурсозатратным, но более сложным в использовании.

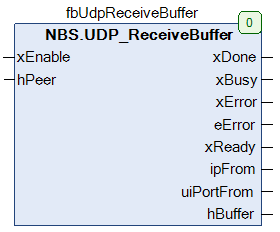


Рисунок 3.5 – Внешний вид ФБ NBS.UDP\_ReceiveBuffer на языке CFC

**Таблица 3.6 – Описание входов и выходов ФБ NBS. UDP\_ReceiveBuffer**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства, полученный от ФБ [NBS.UDP\_PEER](#_3.4.1._ФБ_FILE.DirCreate) |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **xReady** | BOOL | Флаг «данные получены». Принимает значение **TRUE** на один цикл при получении нового пакета данных |
| **ipFrom** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес отправителя |
| **uiPortFrom** | UINT | Номер порта отправителя |
| **hBuffer** | NBS.CAA.HANDLE | Дескриптор буфера принятых данных |

### ФБ NBS.UDP\_SendBuffer

Функциональный блок **NBS.UDP\_SendBuffer** используется для передачи данных. В отличие от ФБ [NBS.UDP\_Send](#_3.3.3._ФБ_NBS.UDP_Send) данный блок не копирует данные по указателю, а принимает на вход дескриптор буфера, в котором они размещены. Для работы с буфером используется библиотека **CAA SegBufMan**. Этот способ является менее ресурсозатратным, но более сложным в использовании.

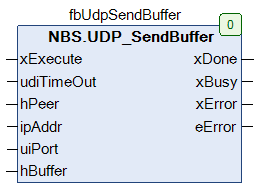


Рисунок 3.6 – Внешний вид ФБ NBS.UDP\_SendBuffer на языке CFC

**Таблица 3.7 – Описание входов и выходов ФБ NBS. UDP\_SendBuffer**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **udiTimeOut** | UDINT | Допустимое время операции (в мкс). Значение **0** означает, что время выполнения ФБ не ограничивается |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства, полученный от ФБ [NBS.UDP\_PEER](#_3.4.1._ФБ_FILE.DirCreate) |
| **ipAddr** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес получателя |
| **uiPort** | UINT | Номер порта получателя |
| **hBuffer** | NBS.CAA.HANDLE | Дескриптор буфера отправляемых данных |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |

## ФБ работы с протоколом TCP

### ФБ NBS.TCP\_Server

Функциональный блок **NBS.TCP\_Server** создает серверный TCP-сокет и возвращает его дескриптор (**handle**), который используется для обработки соединений с помощью ФБ [NBC.TCP\_Connection](#_3.4.2._ФБ_NBS.TCP_Connection).

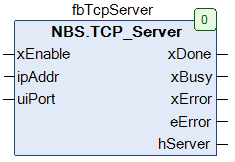


Рисунок 3.7 – Внешний вид ФБ NBS.TCP\_Server на языке CFC

**Таблица 3.8 – Описание входов и выходов ФБ NBS. TCP\_Server**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **ipAddr** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес устройства, на котором создается сокет |
| **uiPort** | UINT | Номер порта сервера |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **hServer** | NBS.CAA.HANDLE | Дескриптор сервера |

### ФБ NBS.TCP\_Connection

Функциональный блок **NBS.TCP\_Connection** используется для обработки одного клиента, подключенного к TCP-серверу. ФБ принимает на вход дескриптор блока [NBS.TCP\_Server](#_3.4.1._ФБ_NBS.TCP_Server) и возвращает дескриптор подключения, который используется для операций получения (ФБ [NBS.TCP\_Read](#_3.4.4._ФБ_NBS.TCP_Read), [NBS.TCP\_ReadBuffer](#_3.4.6._ФБ_NBS.TCP_ReadBuffer)) и передачи данных (ФБ [NBS.TCP\_Write](#_3.4.5._ФБ_NBS.TCP_Write), [NBS.TCP\_WriteBuffer](#_3.4.7._ФБ_NBS.TCP_WriteBuffer)).

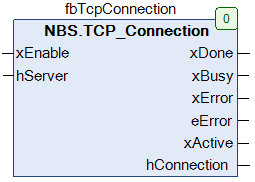


Рисунок 3.8 – Внешний вид ФБ NBS.TCP\_Connection на языке CFC

**Таблица 3.9 – Описание входов и выходов ФБ NBS. TCP\_Connection**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **hServer** | NBS.CAA.HANDLE | Дескриптор сервера |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **xActive** | BOOL | Флаг активности соединения. Если он имеет значение **TRUE** – то к серверу подключен клиент |
| **hConnection** | NBS.CAA.HANDLE | Дескриптор подключения |

### ФБ NBS.TCP\_Сlient

Функциональный блок **NBS.TCP\_Connection** создает клиентский TCP-сокет и возвращает дескриптор подключения, который используется для операций получения (ФБ [NBS.TCP\_Read](#_3.4.4._ФБ_NBS.TCP_Read), [NBS.TCP\_ReadBuffer](#_3.4.6._ФБ_NBS.TCP_ReadBuffer)) и передачи данных (ФБ [NBS.TCP\_Write](#_3.4.5._ФБ_NBS.TCP_Write), [NBS.TCP\_WriteBuffer](#_3.4.7._ФБ_NBS.TCP_WriteBuffer)).

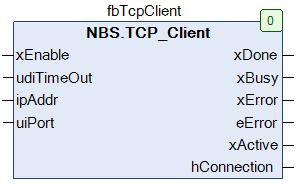


Рисунок 3.9 – Внешний вид ФБ NBS.TCP\_Client на языке CFC

**Таблица 3.10 – Описание входов и выходов ФБ NBS. TCP\_Client**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **udiTimeOut** | UDINT | Допустимое время операции (в мкс). Значение **0** означает, что время выполнения ФБ не ограничивается |
| **ipAddr** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес сервера |
| **uiPort** | UINT | Номер порта сервера |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **xActive** | BOOL | Флаг «сокет успешно открыт». Пока он имеет значение **TRUE** – сокет открыт, и его можно использовать |
| **hConnection** | NBS.CAA.HANDLE | Дескриптор подключения |

### ФБ NBS.TCP\_Read

Функциональный блок **NBS.TCP\_Read** используется для получения данных в заданном подключении. На вход блока подается дескриптор подключения с выхода ФБ [NBS.TCP\_Connection](#_3.4.2._ФБ_NBS.TCP_Connection) (если получатель данных – сервер) или [NBS.TCP\_Client](#_3.4.3._ФБ_NBS.TCP_Сlient) (если получатель данных – клиент).

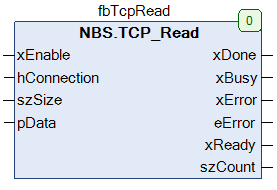


Рисунок 3.10 – Внешний вид ФБ NBS.TCP\_Read на языке CFC

**Таблица 3.11 – Описание входов и выходов ФБ NBS. TCP\_Read**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства |
| **szSize** | NBS.CAA.SIZE | Максимально допустимый размер получаемых данных в байтах. Может быть указан помощью оператора **SIZEOF** |
| **pData** | NBS.CAA.PVOID | Начальный адрес для размещения принятых данных. Может быть указан с помощью оператора **ADR** |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **xReady** | BOOL | Флаг «данные получены». Принимает значение **TRUE** на один цикл при получении нового пакета данных |
| **ipFrom** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес отправителя |
| **uiPortFrom** | UINT | Номер порта отправителя |
| **szCount** | NBS.CAA.SIZE | Размер принятых данных в байтах |

### ФБ NBS.TCP\_Write

Функциональный блок **NBS.TCP\_Write** используется для передачи данных в заданном подключении. На вход блока подается дескриптор подключения с выхода ФБ [NBS.TCP\_Connection](#_3.4.2._ФБ_NBS.TCP_Connection) (если получатель данных – сервер) или [NBS.TCP\_Client](#_3.4.3._ФБ_NBS.TCP_Сlient) (если получатель данных – клиент).

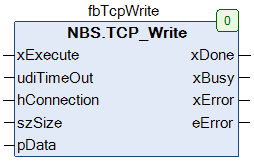


Рисунок 3.11 – Внешний вид ФБ NBS.TCP\_Write на языке CFC

**Таблица 3.12 – Описание входов и выходов ФБ NBS. TCP\_Write**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **udiTimeOut** | UDINT | Допустимое время операции (в мкс). Значение **0** означает, что время выполнения ФБ не ограничивается |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства |
| **ipAddr** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес получателя |
| **uiPort** | UINT | Номер порта получателя |
| **szSize** | NBS.CAA.SIZE | Размер отправляемых данных в байтах. Может быть указан помощью оператора **SIZEOF** |
| **pData** | NBS.CAA.PVOID | Начальный адрес отправляемых данных. Может быть указан с помощью оператора **ADR** |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |

### ФБ NBS.TCP\_ReadBuffer

Функциональный блок **NBS.TCP\_ReadBuffer** используется для получения данных в заданном подключении. На вход блока подается дескриптор подключения с выхода ФБ [NBS.TCP\_Connection](#_3.4.2._ФБ_NBS.TCP_Connection) (если получатель данных – сервер) или [NBS.TCP\_Client](#_3.4.3._ФБ_NBS.TCP_Сlient) (если получатель данных – клиент).

В отличие от ФБ [NBS.TCP\_Read](#_3.4.4._ФБ_NBS.TCP_Read) данный блок не копирует данные по указателю, а возвращает дескриптор буфера, в котором они были размещены. Для работы с буфером используется библиотека **CAA SegBufMan**. Этот способ является менее ресурсозатратным, но более сложным в использовании.

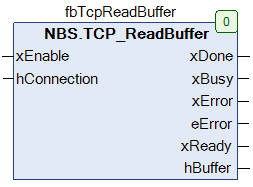


Рисунок 3.12 – Внешний вид ФБ NBS.TCP\_ReadBuffer на языке CFC

**Таблица 3.14 – Описание входов и выходов ФБ NBS. TCP\_ReadBuffer**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xEnable** | BOOL | Вход управления блоком. Пока он имеет значение **TRUE** – блок находится в работе |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки. |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |
| **xReady** | BOOL | Флаг «данные получены». Принимает значение **TRUE** на один цикл при получении нового пакета данных |
| **ipFrom** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес отправителя |
| **uiPortFrom** | UINT | Порт отправителя |
| **hBuffer** | NBS.CAA.HANDLE | Дескриптор буфера принятых данных |

### ФБ NBS.TCP\_WriteBuffer

Функциональный блок **NBS.TCP\_WriteBuffer** используется для отправки данных в заданном подключении. На вход блока подается дескриптор подключения с выхода ФБ [NBS.TCP\_Connection](#_3.4.2._ФБ_NBS.TCP_Connection) (если получатель данных – сервер) или [NBS.TCP\_Client](#_3.4.3._ФБ_NBS.TCP_Сlient) (если получатель данных – клиент).

Функциональный блок **NBS.TCP\_WriteBuffer** используется для получения данных. В отличие от ФБ [NBS.TCP\_Write](#_3.4.5._ФБ_NBS.TCP_Write) данный блок не копирует данные по указателю, а принимает на вход дескриптор буфера, в котором они размещены. Для работы с буфером используется библиотека **CAA SegBufMan**. Этот способ является менее ресурсозатратным, но более сложным в использовании.

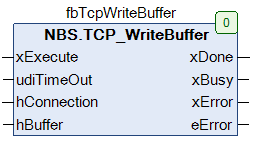


Рисунок 3.13 – Внешний вид ФБ NBS.TCP\_WriteBuffer на языке CFC

**Таблица 3.14 – Описание входов и выходов ФБ NBS. TCP\_WriteBuffer**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **udiTimeOut** | UDINT | Допустимое время операции (в мкс). Значение 0 означает, что время выполнения ФБ не ограничивается |
| **hPeer** | NBS.CAA.HANDLE | Дескриптор сокета устройства |
| **ipAddr** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | IP-адрес получателя |
| **uiPort** | UINT | Номер порта получателя |
| **hBuffer** | NBS.CAA.HANDLE | Дескриптор буфера отправляемых данных |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг завершения работы блока. Принимает значение **TRUE** на один цикл |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновения ошибки. |
| **eError** | [NBS.ERROR](#_3.2.3._Перечисление_FILE.ERROR) | Статус работы ФБ (или имя ошибки) |

## Дополнительные функции

### Функция NBS.IPSTRING\_TO\_UDINT

Функция **NBS.IPSTRING\_TO\_UDINT** конвертирует строковое представление IP-адреса в бинарное (’10.2.11.10’ --> 16#0A02B00A).

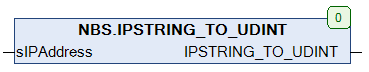


Рисунок 3.14 – Внешний вид функции NBS.IPSTRING\_TO\_UDINT на языке CFC

**Таблица 3.15 – Описание входов и выходов функции NBS. IPSTRING\_TO\_UDINT**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **stIPAdress** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | Строковое представление IP-адреса |
| ***Выходные переменные*** | | |
| **IPSTRING\_TO\_UDINT** | UDINT | Бинарное представление IP-адреса |

### Функция NBS.IPSTRING\_TO\_UDINT

Функция **NBS.UDINT\_TO\_IPSTRING** конвертирует бинарное представление IP-адреса в строковое (16#0A02B00A --> ’10.2.11.10’).

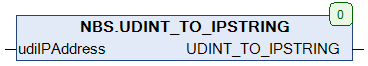


Рисунок 3.15 – Внешний вид функции NBS. UDINT\_TO\_IPSTRING на языке CFC

**Таблица 3.16 – Описание входов и выходов функции NBS. UDINT\_TO\_IPSTRING**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **udiIPAdress** | UDINT | Бинарное представление IP-адреса |
| ***Выходные переменные*** | | |
| **UDINT\_TO\_IPSTRING** | [NBS.IP\_ADDR](#_3.2.1._Структура_NBS.IP_ADDR) | Строковое представление IP-адреса |

### Функция NBS.IS\_MULTICAST\_GROUP

Функция **NBS.IS\_MULTICAST\_GROUP** возвращает **TRUE**, если указанный IP-адрес является адресом [Multicast-группы](#_2.3_Протокол_UDP).

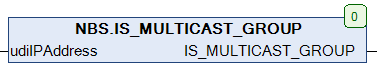


Рисунок 3.16 – Внешний вид функции NBS.IS\_MULTICAST\_GROUP на языке CFC

**Таблица 3.17 – Описание входов и выходов функции NBS. IS\_MULTICAST\_GROUP**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **udiIPAdress** | UDINT | Бинарное представление IP-адреса |
| ***Выходные переменные*** | | |
| **IS\_MULTICAST\_GROUP** | BOOL | Флаг «Multicast-адрес» |

# Примеры работы с библиотекой CAA Net Base Services

## Краткое описание примеров

В данной главе описываются принципы работы с библиотекой **CAA Net Base Services** на примере решения простейшей задачи:

1. Клиент отправляет на сервер строку данных.
2. Сервер получает эти данные и отправляет клиенту инвертированную строку.

Соответственно, в случае отправления на сервер строки ‘hello’ клиент получит в ответ строку ‘olleh’.

В [п. 4.2](#_4.2._Реализация_UDP-сервера) приводится пример решения этой задачи с использованием протокола **UDP**, а в [п. 4.3](#_4.3._Реализация_TCP-сервера) – с использованием протокола **TCP**.

Примеры созданы в среде **CODESYS V3.5 SP11 Patch 5** и подразумевают запуск на виртуальном контроллере **CODESYS Control Win V3**, который входит в состав **CODESYS** и представляет собой программную эмуляцию реального контроллера, запускаемую на ПК с ОС семейства Windows. Для полноценной работы с примерами потребуются два виртуальных контроллера, запущенных на ПК, находящихся в одной локальной сети. Пользователь также может запустить примеры на других устройствах, изменив таргет-файл в проекте CODESYS (**ПКМ** на узел **Device** – **Обновить устройство**).

Каждый пример содержит два приложения (для сервера и клиента). Для загрузки в контроллер конкретного приложения следует нажать **ПКМ** на узел **Application** и выбрать команду **Установить активное приложение**.

Запуск виртуального контроллера выполняется с помощью иконки на **панели задач** Windows. В случае необходимости запустить несколько экземпляров виртуального контроллера на одном ПК следует использовать соответствующий ярлык в **меню** **Пуск** (**Все программы – 3S CODESYS – Codesys Control WinV3 – Codesys Control Win V3**).

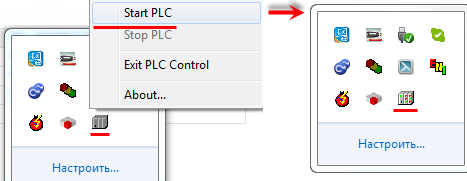


Рисунок 4.1 – Запуск виртуального контроллера

## Реализация UDP-сервера и UDP-клиента

### Основная информация

В данном примере рассматривается реализация UDP-сервера и UDP-клиента.

Решаемая задача описана в [п. 4.1](#_4.1._Краткое_описание).

Листинг примера приведен в приложениях [А1](#_А1._UDP-сервер) (сервер) и [А2](#_А2._UDP-клиент) (клиент).

Пример доступен для скачивания: [Example\_UDP.projectarchive](https://owen.ua/uploads/101/example_udp_3511_v1.projectarchive)

### Реализация UDP-сервера

Условие решаемой задачи – необходимо реализовать UDP-сервер, который будет получать от клиента строку данных, и возвращать ему инвертированную строку.

Сначала следует создать функцию инверсии строки. Такая функция уже есть в свободно распространяемой библиотеке **OSCAT**. Библиотека доступна для скачивания на сайте [oscat.de](http://www.oscat.de/index.php?option=com_jdownloads), а также на сайте компании **ОВЕН** в разделе [CODESYS V3/Библиотеки](http://www.owen.ru/catalog/codesys_v3/80224381). Библиотека OSCAT имеет открытые исходные коды, поэтому во многих случаях рекомендуется копировать ее функции и ФБ в пользовательский проект (вместо добавления через **Менеджер библиотек**).

Функция инверсии строки называется **MIRROR**. Функцию следует скопировать в свой проект и удалить константы **STRING\_LENGTH**, определяющие максимальную длину строк (чтобы не копировать из библиотеки дополнительные POU):

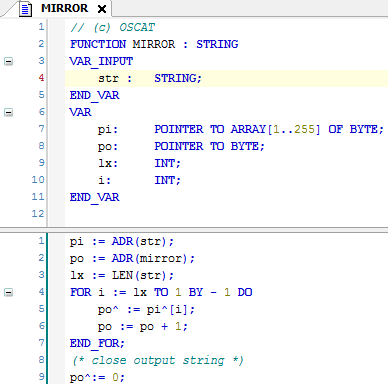


Рисунок 4.2.1 – Код функции MIRROR

Алгоритм работы сервера можно представить следующим образом:

1. Создание сокета.
2. Ожидание запроса от клиента и извлечение данных из полученного запроса.
3. Отправка ответа клиенту.
4. Возвращение на шаг 2.

Данный алгоритм легко представить в виде последовательности шагов, выполняемых с помощью оператора **CASE**. В качестве меток оператора **CASE** можно использовать обычные числа (0, 1, 2 и т. д.) – но это затруднит чтение программы. Поэтому следует объявить перечисление **SERVER\_STATE** (**Application – Добавление объекта – DUT – Перечисление**), в котором свяжем номера шагов с символьными именами.

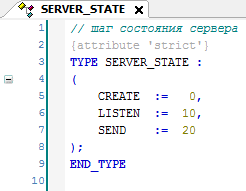


Рисунок 4.2.2 – Объявление перечисления SERVER\_STATE

Затем следует объявить в программе **PLC\_PRG** следующие переменные:

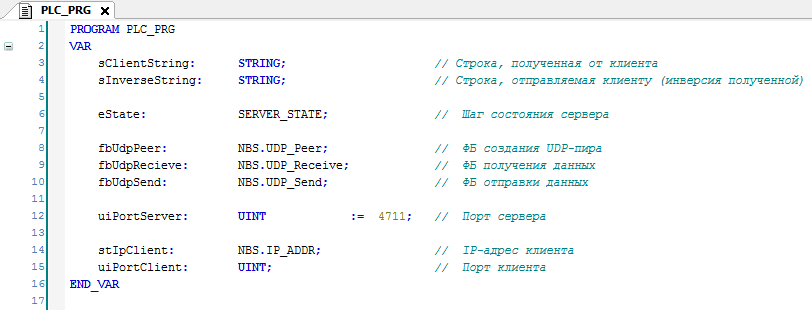


Рисунок 4.2.3 – Объявление переменных программы PLC\_PRG

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Переменная **uiPortServer** определяет номер порта сервера. |

Код программы выглядит следующим образом:

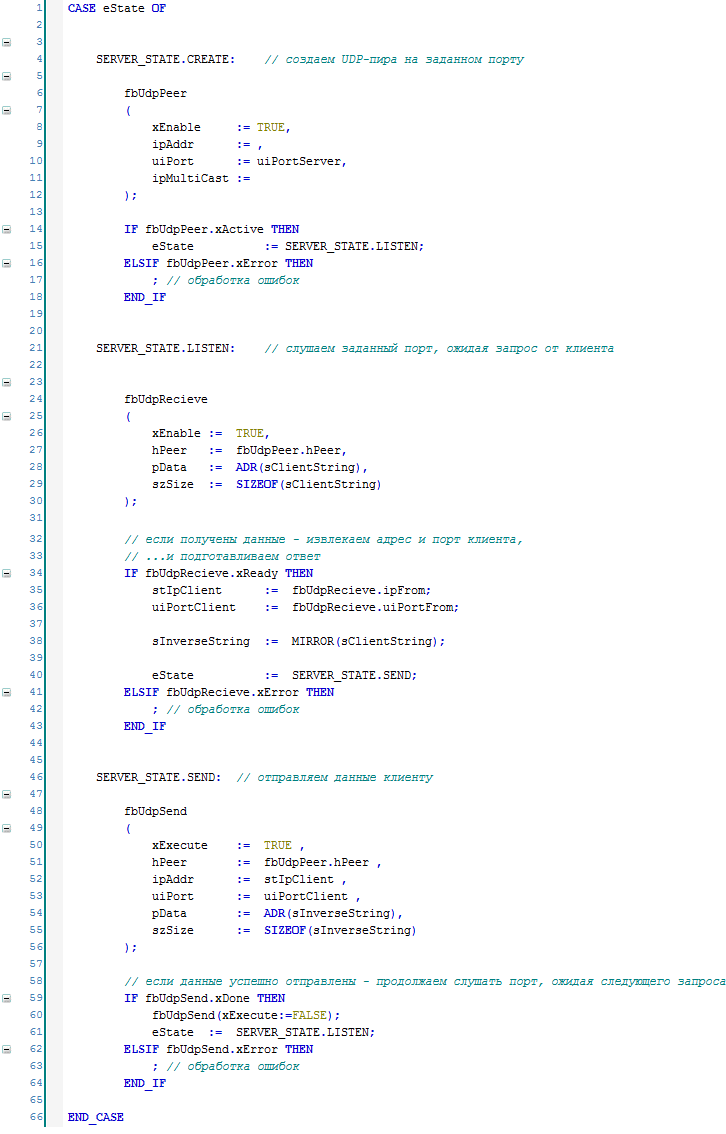


Рисунок 4.2.4 – Код программы PLC\_PRG

На шаге **CREATE** с помощью экземпляра ФБ [NBS.UDP\_Peer](#_3.4.1._ФБ_FILE.DirCreate) происходит создание серверного UDP-сокета на порту, номер которого определяется значением переменной **uiPortServer**. В данном примере используется порт **4711** – он был выбран произвольным образом. Результатом успешного создания сокета является получение его дескриптора (**hPeer**), который будет использоваться для приема и передачи данных на следующих шагах. Если сокет успешно создан (**xActive=TRUE**), то происходит переход на шаг **LISTEN**.

На шаге **LISTEN** с помощью экземпляра ФБ [NBS.UDP\_Receive](#_3.3.2._ФБ_NBS.UDP_Receive) происходит прослушивание порта и ожидание запроса от клиента. Если получен запрос (**xReady=TRUE**), то выполняются следующие операции:

* копирование IP-адреса и номера порта клиента, отправившего запрос, в переменные **stIpCLient** и **uiPortClient**;
* инверсия (см. функцию **MIRROR**) полученной от клиента строки (**sClientString**) с записью результата в переменную **sInverseString**;
* переход на шаг **SEND**.

На шаге **SEND** с помощью экземпляра ФБ [NBS.UDP\_Send](#_3.3.3._ФБ_NBS.UDP_Send) происходит отправление ответа клиенту на заданный IP-адрес (**stIpClient**) и порт (**uiPortClient**). Ответ представляет собой строку **sInverseString**. После завершения операции (**xDone=TRUE**) происходит переход на шаг **LISTEN** для ожидания следующего запроса.

### Реализация UDP-клиента

Задача UDP-клиента. – отправить запрос на сервер и получить ответ.

Как и в случае с сервером, алгоритм работы клиента представляется в виде последовательности шагов, выполняемых с помощью оператора **CASE**. Для использования символьных имен в качестве меток оператора **CASE** следует объявить перечисление **CLIENT\_STATE** (**Application – Добавление объекта – DUT – Перечисление**), в котором номера шагов связываются с символьными именами.

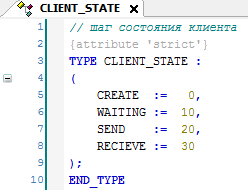


Рисунок 4.2.5 – Объявление перечисления СLIENT\_STATE

Затем следует объявить в программе **PLC\_PRG** следующие переменные:

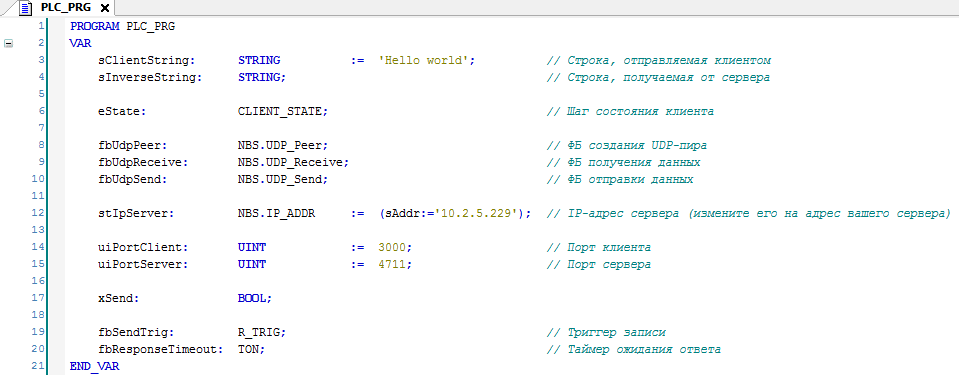


Рисунок 4.2.6 – Объявление переменных программы PLC\_PRG

Переменные программы, определяющие настройки сервера и клиента:

* **stIpServer** – содержит IP-адрес сервера, с которым работает клиент;
* **uiPortServer – содержит номер порта сервера, с которым работает клиент;**
* **uiPortClient – содержит номер порта клиента.**

Код программы будет выглядеть следующим образом:

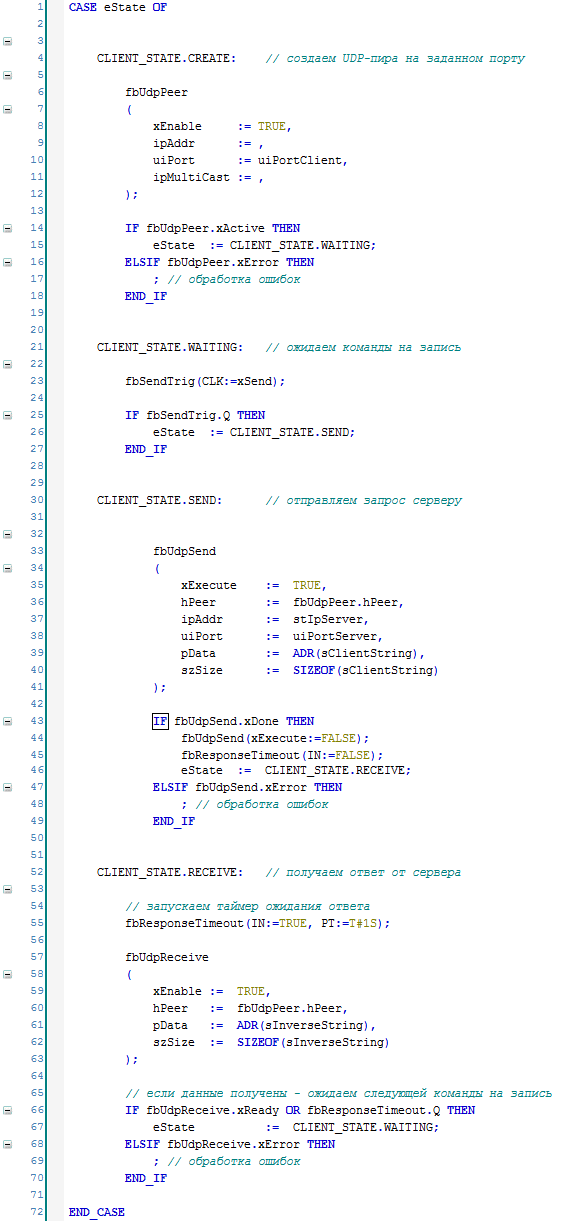


Рисунок 4.2.7 – Код программы PLC\_PRG

На шаге **CREATE** с помощью экземпляра ФБ [NBS.UDP\_Peer](#_3.4.1._ФБ_FILE.DirCreate) происходит создание клиентского UDP-сокета на порту, номер которого определяется значением переменной **uiPortСlient**. В данном примере используется порт **3000** – он был выбран произвольным образом. Результатом успешного создания сокета является получение его дескриптора (**hPeer**), который будет использоваться для приема и передачи данных на следующих шагах. Если сокет успешно создан (**xActive=TRUE**), то происходит переход на шаг **WAITING**.

На шаге **WAITING** происходит ожидание команды отправления запроса на сервер. Команда обрабатывается через триггер, чтобы предотвратить циклическую отправку запросов на сервер (запрос будет отправляться однократно по переднему фронту команды). После получения команды (**xSend=TRUE**) следует переход на шаг **SEND.**

На шаге **SEND** с помощью экземпляра ФБ [NBS.UDP\_Send](#_3.3.3._ФБ_NBS.UDP_Send) происходит отправление запроса на заданный IP-адрес (**stIpServer**) и порт (**uiPortServer**). В рамках рассматриваемого примера UDP-сервер имеет IP-адрес **10.2.5.229** и порт **4711**. Номер порта соответствует порту, указанному при создании сокета на сервере (см. рисунок 4.2.3). Запрос представляет собой строку **sClientString**. Если запрос успешно отправлен (**xDone=TRUE**), то происходит сброс таймера ожидания ответа и переход на шаг **RECEIVE**.

На шаге **RECEIVE** с помощью экземпляра ФБ [NBS.UDP\_Receive](#_3.3.2._ФБ_NBS.UDP_Receive) происходит получение ответа от сервера и запись его в строку **sInverseString**. Если ответ получен (**xReady=TRUE**) или время ожидания истекло (**fbResponseTimeout.Q=TRUE**), то выполняется переход на шаг **WAITING** для ожидания команды отправки следующего запроса.

Пример простой визуализации проекта:



Рисунок 4.2.8 – Внешний вид визуализации клиента

К элементу **Данные для отправки на сервер** привязана переменная **sClientString** и настроена возможность ее изменения (вкладка **InputConfiguration** - **OnMouseClick** – действие **Записать переменную**). К элементу **Ответ сервера** привязана переменная **sInverseString**. К переключателю (тип действия **Клавиша изображения**) привязана переменная **xSend** – она принимает значение **TRUE** при нажатии на элемент и **FALSE** – при его отпускании.

## Реализация TCP-сервера и TCP-клиента

### Основная информация

В данном примере рассматривается реализация TCP-сервера и TCP-клиента.

Решаемая задача описана в [п. 4.1](#_4.1._Краткое_описание).

Листинг примера приведен в приложениях [Б1](#_Б1._TCP-сервер) (сервер) и [Б2](#_Б2._UDP-клиент) (клиент).

Пример доступен для скачивания: [Example\_TCP.projectarchive](https://owen.ua/uploads/101/example_tcp_3511_v1.projectarchive)

### Реализация TCP-сервера

Условие решаемой задачи – необходимо реализовать TCP-сервер, который будет получать от клиента строку данных, и возвращать ему инвертированную строку.

Сначала следует создать функцию инверсии строки. Такая функция уже есть в свободно распространяемой библиотеке **OSCAT**. Библиотека доступна для скачивания на сайте [oscat.de](http://www.oscat.de/index.php?option=com_jdownloads), а также на сайте компании **ОВЕН** в разделе [CODESYS V3/Библиотеки](http://www.owen.ru/catalog/codesys_v3/80224381). Библиотека OSCAT имеет открытые исходные коды, поэтому во многих случаях рекомендуется копировать ее функции и ФБ в пользовательский проект (вместо добавления через **Менеджер библиотек**).

Функция инверсии строки называется **MIRROR**. Функцию следует скопировать в свой проект и удалить константы **STRING\_LENGTH**, определяющие максимальную длину строк (чтобы не копировать из библиотеки дополнительные POU):

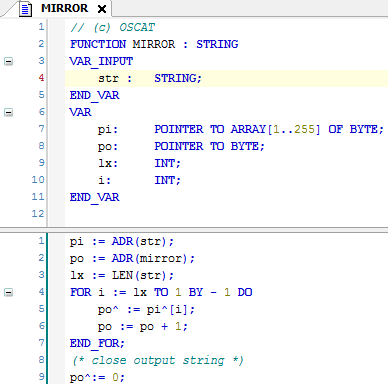


Рисунок 4.3.1 – Код функции MIRROR

Алгоритм работы сервера можно представить следующим образом:

1. Создание сокета.
2. Создание обработчиков для клиентов.
3. Ожидание запросов от клиентов и извлечение данных из полученных запросов.
4. Отправка ответов клиентам.
5. Возвращение на шаг 3.

Данный алгоритм легко представить в виде последовательности шагов, выполняемых с помощью оператора **CASE**. В качестве меток оператора **CASE** можно использовать обычные числа (0, 1, 2 и т. д.) – но это затруднит чтение программы. Поэтому следует объявить перечисление **SERVER\_STATE** (**Application – Добавление объекта – DUT – Перечисление**), в котором номера шагов связываются с символьными именами.

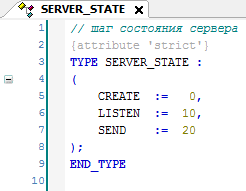


Рисунок 4.3.2 – Объявление перечисления SERVER\_STATE

TCP-сервер может одновременно обслуживать нескольких клиентов. Следует создать структуру **CONNECTION**, которая содержит переменные и ФБ, необходимые для этого.

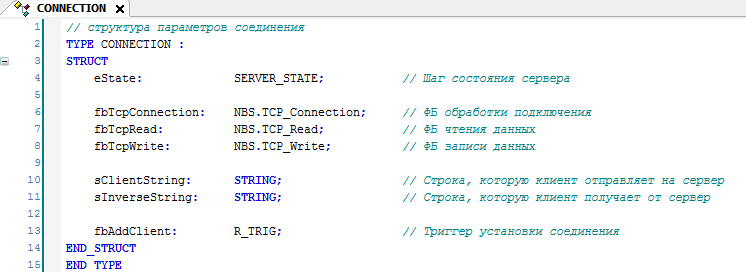


Рисунок 4.3.3 – Объявление структуры CONNECTION

Затем следует объявить в программе **PLC\_PRG** следующие переменные:

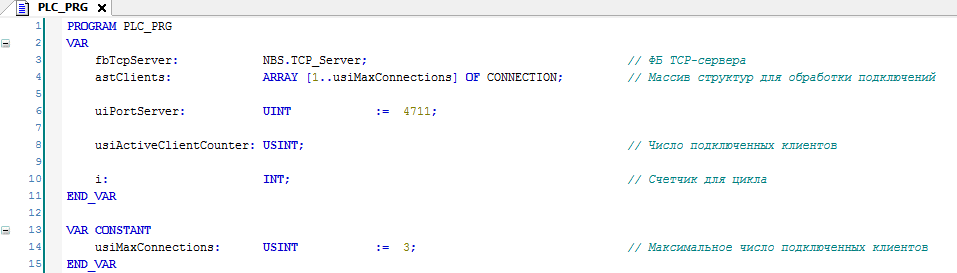


Рисунок 4.3.4 – Объявление переменных программы PLC\_PRG

Переменные программы, определяющие настройки сервера:

* + - **uiPortServer** – содержит номер порта сервера;
    - **usiMaxConnections** – определяет максимальное число клиентов, которые могут быть подключены к серверу.

Код программы выглядит следующим образом:

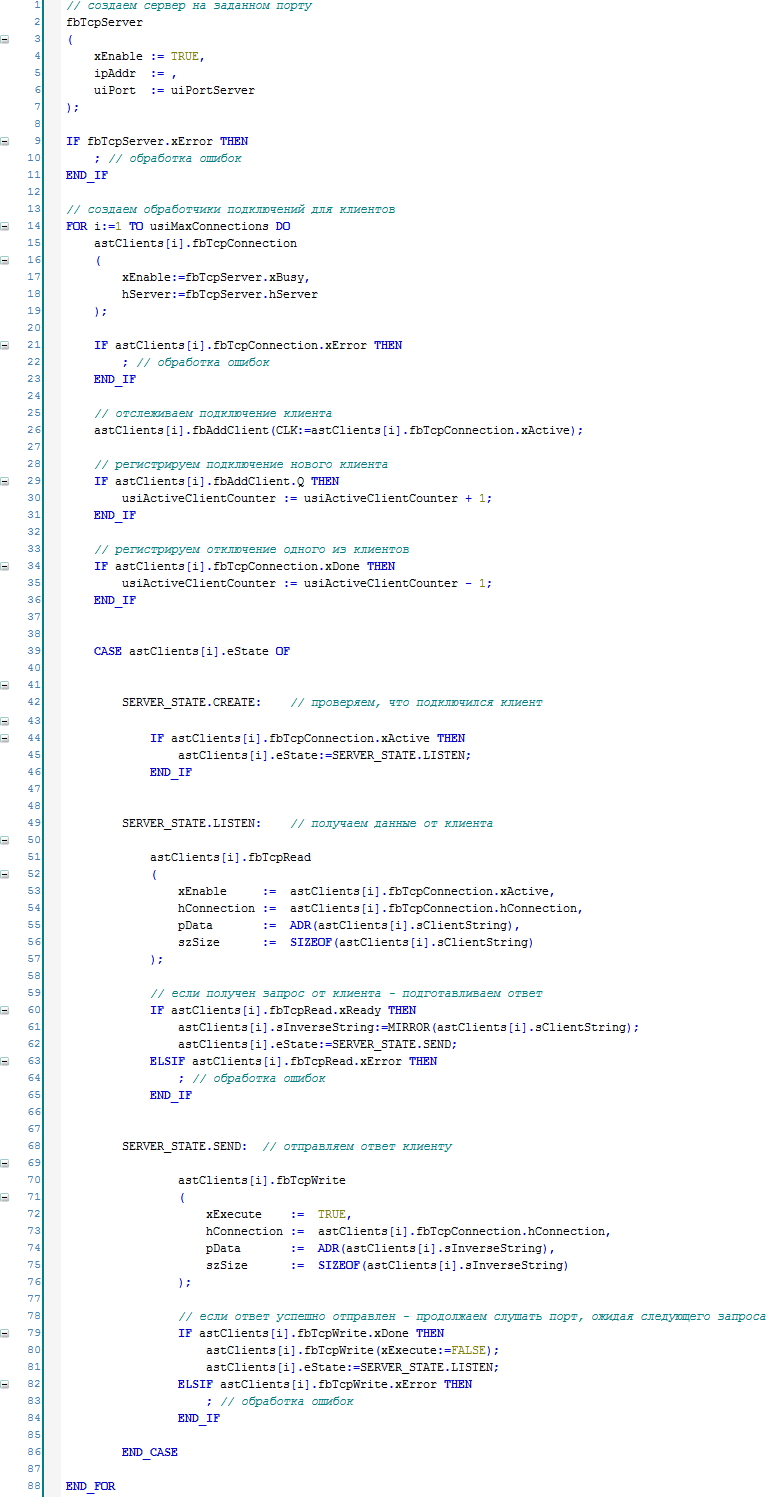


Рисунок 4.3.5 – Код программы PLC\_PRG

В первых строках программы с помощью экземпляра ФБ [NBS.TCP\_Server](#_3.5.1._ФБ_FILE.Open) происходит создание серверного TCP-сокета на порту, номер которого определяется значением переменной **uiPortServer**. В данном примере используется порт **4711** – он был выбран произвольным образом. Результатом успешного создания сокета является получение его дескриптора (**hServer**), который будет использоваться обработчиками клиентов.

В цикле **FOR** происходит последовательная обработка клиентов, подключенных к серверу (максимальное число клиентов определяется значением переменной **usiMaxConnections**). В процессе обработки выполняются следующие операции:

* вызов экземпляров ФБ [NBS.TCP\_Connection](#_3.4.2._ФБ_NBS.TCP_Connection) для обработки клиентов. В случае подключения клиента создается дескриптор (**hConnection**), который будут использовать ФБ получения ([NBS.TCP\_Read](#_3.4.4._ФБ_NBS.TCP_Read)) и передачи данных ([NBS.TCP\_Write](#_3.4.5._ФБ_NBS.TCP_Write));
* подсчет числа клиентов, подключенных к серверу (в случае подключения клиента выход **xActive** принимает значение **TRUE**, в случае отключения – на выходе **xDone** генерируется единичный импульс);
* обмен данными, разбитый на отдельные шаги через оператор **CASE**.

На шаге **CREATE** проверяется, подключен ли клиент к серверу. Если подключен (**xActive=TRUE**), то следует переход на шаг **LISTEN**.

На шаге **LISTEN** с помощью экземпляра ФБ [NBS.TCP\_Read](#_3.4.4._ФБ_NBS.TCP_Read) происходит прослушивание порта и ожидание запроса от клиента. Если получен запрос (**xReady=TRUE**), то выполняются следующие операции:

* инверсия (см. функцию **MIRROR**) полученной от клиента строки (**sClientString**) с записью результата в переменную **sInverseString**;
* переход на шаг **SEND**.

На шаге **SEND** с помощью экземпляра ФБ [NBS.TCP\_Write](#_3.4.5._ФБ_NBS.TCP_Write) происходит отправление ответа клиенту. Ответ представляет собой строку **sInverseString**. После завершения операции (**xDone=TRUE**) происходит переход на шаг **LISTEN** для ожидания следующего запроса.

### Реализация TCP-клиента

Задача TCP-клиента – отправить запрос на сервер и получить ответ.

Как и в случае с сервером, алгоритм работы клиента можно представить в виде последовательности шагов, выполняемых с помощью оператора **CASE**. Для использования символьных имен в качестве меток оператора **CASE** следует объявить перечисление **CLIENT\_STATE** (**Application – Добавление объекта – DUT – Перечисление**), в котором номера шагов связываются с символьными именами.

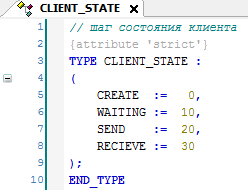


Рисунок 4.3.6 – Объявление перечисления СLIENT\_STATE

Затем следует объявить в программе **PLC\_PRG** следующие переменные:

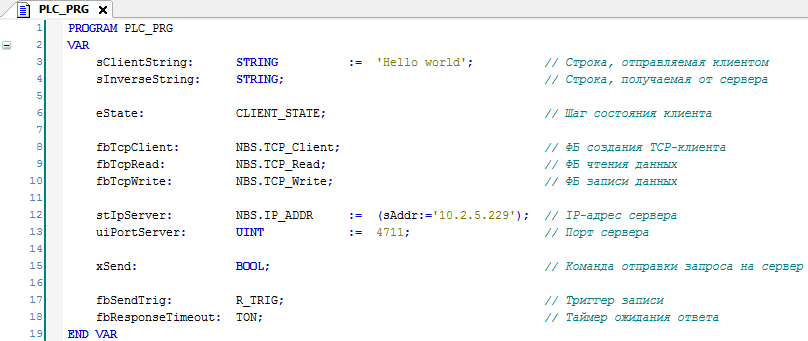


Рисунок 4.3.7 – Объявление переменных программы PLC\_PRG

Переменные программы, определяющие настройки сервера:

* + - **stIpServer** – содержит IP-адрес сервера, с которым работает клиент;
* **uiPortServer** – содержит номер порта сервера, с которым работает клиент.

Код программы будет выглядеть следующим образом:

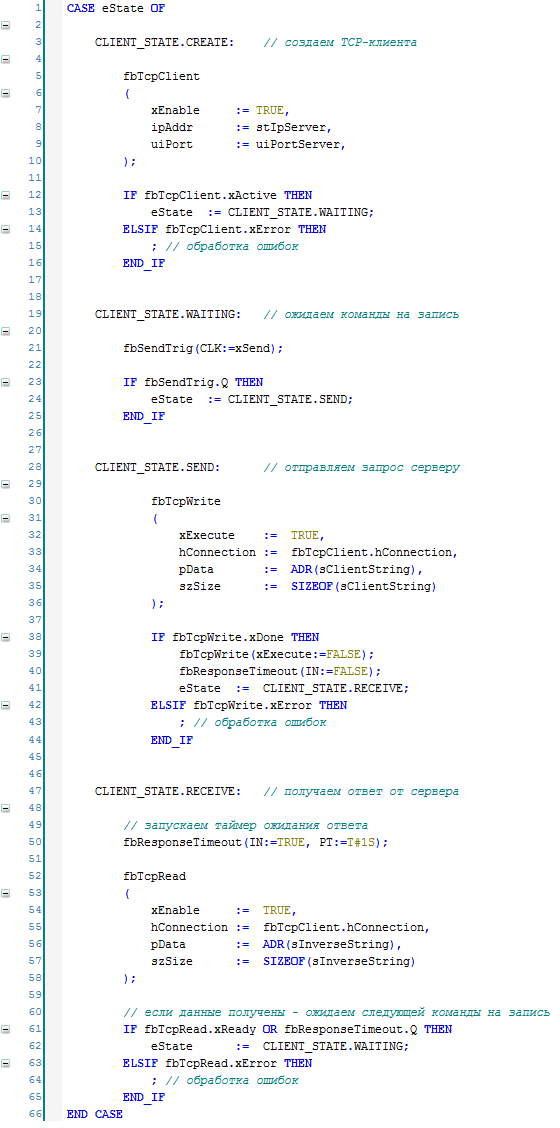


Рисунок 4.3.8 – Код программы PLC\_PRG

На шаге **CREATE** с помощью экземпляра ФБ [NBS.TCP\_Client](#_3.4.3._ФБ_NBS.TCP_Сlient) происходит создание клиентского TCP-сокета для работы с сервером, который имеет IP-адрес **stIpServer** и номер порта **uiPortServer**. В данном примере используется порт **3000** – он был выбран произвольным образом. Результатом успешного создания сокета является получение дескриптора соединения (**hConnection**), который будет использоваться для получения и передачи данных на следующих шагах. Если сокет успешно создан (**xActive=TRUE**), то происходит переход на шаг **WAITING**.

На шаге **WAITING** происходит ожидание команды отправления запроса на сервер. Команда обрабатывается через триггер, чтобы предотвратить циклическую отправку запросов на сервер (запрос будет отправляться однократно по переднему фронту команды). После получения команды (**xSend=TRUE**) следует переход на шаг **SEND.**

На шаге **SEND** с помощью экземпляра ФБ [NBS.TCP\_Write](#_3.4.5._ФБ_NBS.TCP_Write) происходит отправление запроса серверу. Запрос представляет собой строку **sClientString**. Если запрос успешно отправлен (**xDone=TRUE**), то происходит переход на шаг **RECEIVE**.

На шаге **RECEIVE** с помощью экземпляра ФБ [NBS.TCP\_Read](#_3.4.4._ФБ_NBS.TCP_Read) происходит получение ответа от сервера и запись его в строку **sInverseString**. Если ответ получен (**xReady=TRUE**) или время ожидания истекло (**fbResponseTimeout.Q=TRUE**), то выполняется переход на шаг **WAITING** для ожидания команды отправки следующего запроса.

Пример простой визуализации проекта:



Рисуно 4.3.9 – Внешний вид визуализации клиента

К элементу **Данные для отправки на сервер** привязана переменная **sClientString** и настроена возможность ее изменения (вкладка **InputConfiguration** – **OnMouseClick** – действие **Записать переменную**). К элементу **Ответ сервера** привязана переменная **sInverseString**. К переключателю (тип действия **Клавиша изображения**) привязана переменная **xSend** – она принимает значение **TRUE** при нажатии на элемент и **FALSE** – при его отпускании.

## Работа с примером

1. Каждый пример содержит два приложения – **Server** и **Client.**

В приложениях следует отредактировать:

* IP-адрес сервера (переменная **stIpServer** в приложении **Client**);
* номер порта сервера (переменная **uiPortServer** в обоих приложениях);
* номер порта клиента (переменная **uiPortClient** в приложении **Client** – *только для примера* ***UDP***).

Сначала следует запустить два виртуальных контроллера на ПК, подключенных к одной локальной сети (можно использовать один ПК с несколькими сетевыми картами). Для загрузки в контроллер конкретного приложения необходимо нажать **ПКМ** на узел **Application** и выбрать команду **Установить активное приложение**. Предварительно рекомендуется выполнить команду **Сброс заводской** из меню **Онлайн**.

В случае необходимости можно изменить таргет-файлы приложений, чтобы запустить их на нужных устройствах (**ПКМ на узел Device – Обновить устройство**).

1. В приложении **Client** следует перейти на страницу визуализации.

Затем ввести строку данных, которая будет отправлена на сервер, и нажать кнопку.

В поле Ответ сервера должна появиться инвертированная строка.

Если строка не появляется – следует проверить корректность сетевых настроек устройств, на которых запускаются проекты, и уточнить особенности настроек сети (например, на маршрутизаторах могут быть заблокированы какие-то порты).



Рисунок 4.4.1 – Работа с примером

## Рекомендации и замечания

Ниже перечислены основные тезисы и рекомендации по разработке программ, работающих с сокетами, использованные в данном документе.

* ФБ и программы, реализующие обмен, разбиваются на шаги, которые выполняются через оператор **CASE.**
* Чтобы сделать прозрачным переходы между шагами, можно использовать **перечисления**.
* Переход к следующему шагу должен происходить только после окончания предыдущего. Контроль окончания шага, в частности, может осуществляться с помощью выходов соответствующих ФБ (**xDone, xActive, xReady** и т. д.).

Следует также отметить ряд моментов, оставшихся за пределами примеров документа:

* В рамках примера рассматривается обмен данными между сервером и клиентом с помощью обычных текстовых строк. Для реализации конкретного протокола потребуется его спецификация, описывающая форматы и последовательности запросов и ответов.
* В большинстве случаев требуется тщательная обработка ошибок. Контролируйте выходы **xError** и **eError** соответствующих ФБ. См. описание кодов ошибок в [п. 3.2.2](#_3.2.2._Перечисление_FILE.ERROR).

# Библиотека OwenCommunication

## Общая информация

Библиотека **OwenCommunication** содержит функциональные блоки, которые могут использоваться для реализации нестандартных протоколов в том случае, если контроллер выступает в роли TCP- или UDP-клиента. Также библиотека включает в себя блоки для работы по протоколу Modbus, блок реализации нестандартных протоколов для последовательной линии связи и вспомогательные функции и ФБ для конвертации данных, обзор которых не является задачей данного руководства.

Библиотека доступна на сайте [ОВЕН](https://www.owen.ua/) в разделе **CODESYS V3/Библиотеки и компоненты**.

Для обмена по TCP и UDP используются следующие ФБ библиотеки:

* **TCP\_Client** – аналогичен блоку [TCP\_Client](#_ФБ_NBS.TCP_Сlient) из библиотеки [CAA Net Base Services](#_Библиотека_CAA_Net). Единственное отличие – IP-адрес явно задается в виде переменной типа **STRING**, а не структуры [IP\_ADDR](#_Структура_NBS.IP_ADDR));
* **UNM\_TcpRequest** – блок отправки произвольного запроса по протоколу TCP и получения ответа. Включает в себя функционал TCP-клиента из [п. 4.3.3](#_Реализация_TCP-клиента);
* **UNM\_UdpRequest** – блок отправки произвольного запроса по протоколу UDP и получения ответа. Включает в себя функционал UDP-клиента из [п. 4.2.3](#_Реализация_UDP-клиента).

Блоки библиотеки **OwenCommunication** построены на базе ФБ библиотеки **CAA Net Base Services** и предоставляют готовый функционал для реализации TCP- и UDP-клиентов. Описание блоков приведено ниже.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Работа библиотеки поддерживается только на контроллерах ОВЕН и виртуальном контроллере **CODESYS Control Win V3**. |

## ФБ UNM\_TcpRequest

Функциональный блок **UNM\_TcpRequest** используется для реализации нестандартного протокола поверх протокола **TCP**. По переднему фронту на входе **xExecute** происходит отправка содержимого буфера запроса, расположенного по указателю **pRequest**, размером **szRequest** байт через соединение, определяемое дескриптором **hConnection**, полученным от ФБ **TCP\_Client**. Ответ от slave-устройства ожидается в течение времени **tTimeout.** При получении ответа происходит его проверка на основании значений входов **szExpectedSize** и **wStopChar**:

* если **szExpectedSize** <> **0**, то ответ считается корректным, если его размер в байтах = **szExpectedSize**;
* если **szExpectedSize** = **0** и **wStopChar** <> **16#0000**, то последние один (при **wStopChar** = **16#00xx**) или два (при **wStopCha**r = **16#xxxx**) байта ответа (где x – произвольное значение) проверяются на равенство младшему или обоим байтам **wStopChar**. Это может использоваться при реализации строковых протоколов, в которых заранее известен стоп-символ;
* если **szExpectedSize** = **0** и **wStopChar** = **16#0000**, то любой полученный ответ считается корректным.

В случае получения корректного ответа выход **xDone** принимает значение **TRUE**, выход **eError** = **NO\_ERROR**, а на выходе **uiResponseSize** отображается размер ответа в байтах. Полученные данные помещается в буфер, расположенный по указателю **pResponse** и имеющий размер **szResponse** байт.

В случае отсутствия ответа ФБ повторяет запрос. Число переповторов определяется входом **usiRetry** (значение **0** соответствует отсутствию переповторов). Если ни на один из запросов не был получен ответ, то выход **xError** принимает значение **TRUE**, а выход **eError** = **TIME\_OUT**.

Для отправки нового запроса следует создать передний фронт на входе **xExecute**.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  В случае отправки запросов, для которых не подразумевается получение ответа, рекомендуется для входа **tTimeout** установить значение **T#1ms**. |

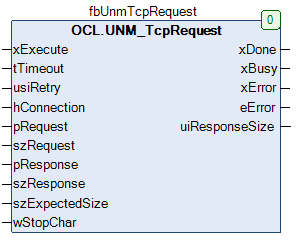


Рисунок 5.1 – Внешний вид ФБ UNM\_TcpRequest на языке CFC

**Таблица 5.1 – Описание входов и выходов ФБ UNM\_TcpRequest**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Описание** |
| **Входы** | | |
| xExecute | BOOL | По переднему фронту происходит однократная (с возможностью переповторов в случае отсутствия ответа) отправка запроса |
| tTimeout | TIME | Таймаут ожидания ответа от slave-устройства (**T#0ms** – время ожидания не ограничено) |
| usiRetry | USINT | Число переповторов в случае отсутствия ответа |
| hConnection | CAA.HANDLE | Дескриптор TCP-соединения, полученный от ФБ **TCP\_Client** |
| pRequest | CAA.PVOID | Указатель на буфер запроса |
| szRequest | CAA.SIZE | Размер буфера запроса в байтах |
| pResponse | CAA.PVOID | Указатель на буфер ответа |
| szResponse | CAA.SIZE | Размер буфера ответа в байтах |
| szExpectedSize | CAA.SIZE | Ожидаемый размер ответа в байтах (**0** – размер неизвестен) |
| wStopChar | WORD | Стоп-символы протокола. Для протокола с двумя стоп-символами оба байта переменной должны быть отличны от нуля. Для протокола с одним стоп-символом старший байт должен быть равен нулю, а младший отличный от нуля. Если в протоколе отсутствуют стоп-символы, то следует установить значение 0 |
| **Выходы** | | |
| xDone | BOOL | **TRUE** – получен корректный ответ от slave-устройства |
| xBusy | BOOL | **TRUE** – ФБ находится в работе |
| xError | BOOL | Принимает значение **TRUE** в случае возникновения ошибки |
| eError | ERROR | Статус работы ФБ (или код ошибки) |
| uiResponseSize | UINT | Размер полученного ответа в байтах |

## ФБ UNM\_UdpRequest

Функциональный блок **UNM\_UdpRequest** используется для реализации нестандартного протокола поверх протокола **UDP**. По переднему фронту на входе **xExecute** происходит отправка содержимого буфера запроса, расположенного по указателю **pRequest**, размером **szRequest** байт на IP-адрес **sServerIpAddr** и порт **uiServerPort**. На стороне контроллера для отправки используется порт **uiLocalPort** и IP-адрес **0.0.0.0**. (т. е. отправка запроса осуществляется по всем доступным интерфейсам).

Ответ от slave-устройства ожидается в течение времени **tTimeout.** При получении ответа происходит его проверка на основании значений входов **szExpectedSize** и **wStopChar**:

* если **szExpectedSize** <> **0**, то ответ считается корректным, если его размер в байтах = **szExpectedSize**;
* если **szExpectedSize** = **0** и **wStopChar** <> **16#0000**, то последние один (при **wStopChar** = **16#00xx**) или два (при **wStopCha**r = **16#xxxx**) байта ответа (где x – произвольное значение) проверяются на равенство младшему или обоим байтам **wStopChar**. Это может использоваться при реализации строковых протоколов, в которых заранее известен стоп-символ;
* если **szExpectedSize** = 0 и **wStopChar** = **16#0000**, то любой полученный ответ считается корректным.

В случае получения корректного ответа выход **xDone** принимает значение **TRUE**, выход **eError** = **NO\_ERROR**, а на выходе **uiResponseSize** отображается размер ответа в байтах. Полученные данные помещается в буфер, расположенный по указателю **pResponse** и имеющий размер **szResponse** байт.

В случае отсутствия ответа ФБ повторяет запрос. Число переповторов определяется входом **usiRetry** (значение **0** соответствует отсутствию переповторов). Если ни на один из запросов не был получен ответ, то выход **xError** принимает значение **TRUE**, а выход **eError** = **TIME\_OUT**.

Для отправки нового запроса следует создать передний фронт на входе **xExecute**.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  В случае отправки запросов, для которых не подразумевается получение ответа, рекомендуется для входа **tTimeout** установить значение **T#1ms**. |

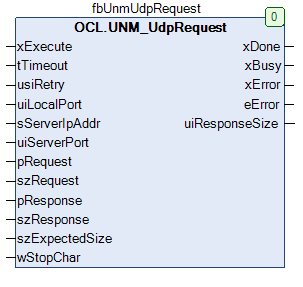


Рисунок 5.2 – Внешний вид ФБ UNM\_UdpRequest на языке CFC

**Таблица 5.7.3 – Описание входов и выходов ФБ UNM\_UdpRequest**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Описание** |
| **Входы** | | |
| xExecute | BOOL | По переднему фронту происходит однократная (с возможностью переповторов в случае отсутствия ответа) отправка запроса |
| tTimeout | TIME | Таймаут ожидания ответа от slave-устройства (**T#0ms** – время ожидания не ограничено) |
| usiRetry | USINT | Число переповторов в случае отсутствия ответа |
| uiLocalPort | UINT | Порт контроллера, через который отправляется запрос |
| sServerIpAddr | STRING | IP-адрес slave-устройства в формате [IPv4](https://ru.wikipedia.org/wiki/IPv4) (‘xxx.xxx.xxx.xxx’) |
| uiServerPort | UINT | Порт slave-устройства |
| pRequest | CAA.PVOID | Указатель на буфер запроса |
| szRequest | CAA.SIZE | Размер буфера запроса в байтах |
| pResponse | CAA.PVOID | Указатель на буфер ответа |
| szResponse | CAA.SIZE | Размер буфера ответа в байтах |
| szExpectedSize | CAA.SIZE | Ожидаемый размер ответа в байтах (**0** – размер неизвестен) |
| wStopChar | WORD | Стоп-символы протокола. Для протокола с двумя стоп-символами оба байта переменной должны быть отличны от нуля. Для протокола с одним стоп-символом старший байт должен быть равен нулю, а младший отличный от нуля. Если в протоколе отсутствуют стоп-символы, то следует установить значение 0 |
| **Выходы** | | |
| xDone | BOOL | **TRUE** – получен корректный ответ от slave-устройства |
| xBusy | BOOL | **TRUE** – ФБ находится в работе |
| xError | BOOL | Принимает значение **TRUE** в случае возникновения ошибки |
| eError | ERROR | Статус работы ФБ (или код ошибки) |
| uiResponseSize | UINT | Размер полученного ответа в байтах |

# Листинг примера UDP

## UDP-сервер

### Перечисление SERVER\_STATE

// шаг состояния сервера

{attribute 'strict'}

TYPE SERVER\_STATE :

(

CREATE := 0,

LISTEN := 10,

SEND := 20

);

END\_TYPE

### Функция MIRROR

// (c) OSCAT

FUNCTION MIRROR : STRING

VAR\_INPUT

str : STRING;

END\_VAR

VAR

pi: POINTER TO ARRAY [1..255] OF BYTE;

po: POINTER TO BYTE;

lx: INT;

i: INT;

END\_VAR

(\*

version 1.1 29. mar. 2008

programmer hugo

tested by tobias

this function reverses an input string.

\*)

pi := ADR(str);

po := ADR(mirror);

lx := LEN(str);

FOR i := lx TO 1 BY - 1 DO

Po^ := pi^[i];

po := po + 1;

END\_FOR;

(\* close output string \*)

po^:= 0;

(\* revision histroy

hm 4. feb. 2008 rev 1.0

original release

hm 29. mar. 2008 rev 1.1

changed STRING to STRING(STRING\_LENGTH)

\*)

### Программа PLC\_PRG

PROGRAM PLC\_PRG

VAR

sClientString: STRING; // Строка, полученная от клиента

sInverseString: STRING; // Строка, отправляемая клиенту (инверсия полученной)

eState: SERVER\_STATE; / /Шаг состояния сервера

fbUdpPeer: NBS.UDP\_Peer; // ФБ создания UDP-пира

fbUdpReceive: NBS.UDP\_Receive; // ФБ получения данных

fbUdpSend: NBS.UDP\_Send; // ФБ отправки данных

uiPortServer: UINT := 4711; // Порт сервера

stIpClient: NBS.IP\_ADDR; // IP-адрес клиента

uiPortClient: UINT; // Порт клиента

END\_VAR

CASE eState OF

SERVER\_STATE.CREATE: // создаем UDP-пира на заданном порту

fbUdpPeer

(

xEnable := TRUE,

ipAddr := ,

uiPort := uiPortServer,

ipMultiCast :=

);

IF fbUdpPeer.xActive THEN

eState := SERVER\_STATE.LISTEN;

ELSIF fbUdpPeer.xError THEN

; // обработка ошибок

END\_IF

SERVER\_STATE.LISTEN: // слушаем заданный порт, ожидая запрос от клиента

fbUdpReceive

(

xEnable := TRUE,

hPeer := fbUdpPeer.hPeer,

pData := ADR(sClientString),

szSize := SIZEOF(sClientString)

);

// если получены данные - извлекаем адрес и порт клиента,

// ...и подготавливаем ответ

IF fbUdpReceive.xReady THEN

stIpClient := fbUdpReceive.ipFrom;

uiPortClient := fbUdpReceive.uiPortFrom;

sInverseString := MIRROR(sClientString);

eState := SERVER\_STATE.SEND;

ELSIF fbUdpReceive.xError THEN

; // обработка ошибок

END\_IF

SERVER\_STATE.SEND: // отправляем данные клиенту

fbUdpSend

(

xExecute := TRUE ,

hPeer := fbUdpPeer.hPeer ,

ipAddr := stIpClient ,

uiPort := uiPortClient ,

pData := ADR(sInverseString),

szSize := SIZEOF(sInverseString)

);

// если данные успешно отправлены –

// …продолжаем слушать порт, ожидая следующего запроса

IF fbUdpSend.xDone THEN

fbUdpSend(xExecute:=FALSE);

eState := SERVER\_STATE.LISTEN;

ELSIF fbUdpSend.xError THEN

; // обработка ошибок

END\_IF

END\_CASE

## UDP-клиент

### Перечисление CLIENT\_STATE

// шаг состояния клиента

{attribute 'strict'}

TYPE CLIENT\_STATE :

(

CREATE := 0,

WAITING := 10,

SEND := 20,

RECEIVE := 30

);

END\_TYPE

### Программа PLC\_PRG

PROGRAM PLC\_PRG

VAR

sClientString: STRING := 'Hello world'; // Строка, отправляемая клиентом

sInverseString: STRING; // Строка, получаемая от сервера

eState: CLIENT\_STATE; // Шаг состояния клиента

fbUdpPeer: NBS.UDP\_Peer; // ФБ создания UDP-пира

fbUdpReceive: NBS.UDP\_Receive; // ФБ получения данных

fbUdpSend: NBS.UDP\_Send; // ФБ отправки данных

// IP-адрес сервера (измените его на адрес вашего сервера)

stIpServer: NBS.IP\_ADDR := (sAddr := '10.2.5.229');

uiPortClient: UINT := 3000; // Порт клиента

uiPortServer: UINT := 4711; // Порт сервера

xSend: BOOL;

fbSendTrig: R\_TRIG; // Триггер записи

fbResponseTimeout: TON; // Таймер ожидания ответа

END\_VAR

CASE eState OF

CLIENT\_STATE.CREATE: // создаем UDP-пира на заданном порту

fbUdpPeer

(

xEnable := TRUE,

ipAddr := ,

uiPort := uiPortClient,

ipMultiCast := ,

);

IF fbUdpPeer.xActive THEN

eState := CLIENT\_STATE.WAITING;

ELSIF fbUdpPeer.xError THEN

; // обработка ошибок

END\_IF

CLIENT\_STATE.WAITING: // ожидаем команды на запись

fbSendTrig(CLK:=xSend);

IF fbSendTrig.Q THEN

eState := CLIENT\_STATE.SEND;

END\_IF

CLIENT\_STATE.SEND: // отправляем запрос серверу

fbUdpSend

(

xExecute := TRUE,

hPeer := fbUdpPeer.hPeer,

ipAddr := stIpServer,

uiPort := uiPortServer,

pData := ADR(sClientString),

szSize := SIZEOF(sClientString)

);

IF fbUdpSend.xDone THEN

fbUdpSend(xExecute:=FALSE);

fbResponseTimeout(IN:= FALSE);

eState := CLIENT\_STATE.RECEIVE;

ELSIF fbUdpSend.xError THEN

; // обработка ошибок

END\_IF

CLIENT\_STATE.RECEIVE: // получаем ответ от сервера

// запускаем таймер ожидания ответа

fbResponseTimeout(IN:= TRUE, PT:= T#1s);

fbUdpReceive

(

xEnable := TRUE,

hPeer := fbUdpPeer.hPeer,

pData := ADR(sInverseString),

szSize := SIZEOF(sInverseString)

);

// если данные получены - ожидаем следующей команды на запись

IF fbUdpReceive.xReady OR fbResponseTimeout.Q THEN

eState := CLIENT\_STATE.WAITING;

ELSIF fbUdpReceive.xError THEN

; // обработка ошибок

END\_IF

END\_CASE

# Листинг примера TCP

## TCP-сервер

### Перечисление SERVER\_STATE

// шаг состояния сервера

{attribute 'strict'}

TYPE SERVER\_STATE :

(

CREATE := 0,

LISTEN := 10,

SEND := 20

);

END\_TYPE

### Структура CONNECTION

// структура параметров соединения

TYPE CONNECTION :

STRUCT

eState: SERVER\_STATE; // Шаг состояния сервера

fbTcpConnection: NBS.TCP\_Connection; // ФБ обработки подключения

fbTcpRead: NBS.TCP\_Read; // ФБ чтения данных

fbTcpWrite: NBS.TCP\_Write; // ФБ записи данных

sClientString: STRING; // Строка, которую клиент отправляет на сервер

sInverseString: STRING; // Строка, которую клиент получает от сервера

fbAddClient: R\_TRIG; // Триггер установки соединения

END\_STRUCT

END\_TYPE

### Функция MIRROR

// (c) OSCAT

FUNCTION MIRROR : STRING

VAR\_INPUT

str : STRING;

END\_VAR

VAR

pi: POINTER TO ARRAY [1..255] OF BYTE;

po: POINTER TO BYTE;

lx: INT;

i: INT;

END\_VAR

(\*

version 1.1 29. mar. 2008

programmer hugo

tested by tobias

this function reverses an input string.

\*)

pi := ADR(str);

po := ADR(mirror);

lx := LEN(str);

FOR i := lx TO 1 BY - 1 DO

Po^ := pi^[i];

po := po + 1;

END\_FOR;

(\* close output string \*)

po^:= 0;

(\* revision histroy

hm 4. feb. 2008 rev 1.0

original release

hm 29. mar. 2008 rev 1.1

changed STRING to STRING(STRING\_LENGTH)

\*)

### Программа PLC\_PRG

PROGRAM PLC\_PRG

VAR

fbTcpServer: NBS.TCP\_Server; // ФБ TCP-сервера

// Массив структур для обработки подключений

astClients: ARRAY [1..usiMaxConnections] OF CONNECTION;

uiPortServer: UINT := 4711;

usiActiveClientCounter: USINT; // Число подключенных клиентов

i: INT; // Счетчик для цикла

END\_VAR

VAR CONSTANT

// Максимальное число подключенных клиентов

usiMaxConnections: USINT := 3;

END\_VAR

// создаем сервер на заданном порту

fbTcpServer

(

xEnable := TRUE,

ipAddr := ,

uiPort := uiPortServer

);

IF fbTcpServer.xError THEN

; // обработка ошибок

END\_IF

// создаем обработчики подключений для клиентов

FOR i:=1 TO usiMaxConnections DO

astClients[i].fbTcpConnection

(

xEnable:=fbTcpServer.xBusy,

hServer:=fbTcpServer.hServer

);

IF astClients[i].fbTcpConnection.xError THEN

; // обработка ошибок

END\_IF

// отслеживаем подключение клиента

astClients[i].fbAddClient(CLK:=astClients[i].fbTcpConnection.xActive);

// регистрируем подключение нового клиента

IF astClients[i].fbAddClient.Q THEN

usiActiveClientCounter := usiActiveClientCounter + 1;

END\_IF

// регистрируем отключение одного из клиентов

IF astClients[i].fbTcpConnection.xDone THEN

usiActiveClientCounter := usiActiveClientCounter - 1;

END\_IF

CASE astClients[i].eState OF

SERVER\_STATE.CREATE: // проверяем, что подключился клиент

IF astClients[i].fbTcpConnection.xActive THEN

astClients[i].eState:=SERVER\_STATE.LISTEN;

END\_IF

SERVER\_STATE.LISTEN: // получаем данные от клиента

astClients[i].fbTcpRead

(

xEnable := astClients[i].fbTcpConnection.xActive,

hConnection := astClients[i].fbTcpConnection.hConnection,

pData := ADR(astClients[i].sClientString),

szSize := SIZEOF(astClients[i].sClientString)

);

// если получен запрос от клиента - подготавливаем ответ

IF astClients[i].fbTcpRead.xReady THEN

astClients[i].sInverseString:=MIRROR(astClients[i].sClientString);

astClients[i].eState:=SERVER\_STATE.SEND;

ELSIF astClients[i].fbTcpRead.xError THEN

; // обработка ошибок

END\_IF

SERVER\_STATE.SEND: // отправляем ответ клиенту

astClients[i].fbTcpWrite

(

xExecute := TRUE,

hConnection := astClients[i].fbTcpConnection.hConnection,

pData := ADR(astClients[i].sInverseString),

szSize := SIZEOF(astClients[i].sInverseString)

);

// если ответ успешно отправлен - продолжаем слушать порт, ожидая следующего запроса

IF astClients[i].fbTcpWrite.xDone THEN

astClients[i].fbTcpWrite(xExecute:=FALSE);

astClients[i].eState:=SERVER\_STATE.LISTEN;

ELSIF astClients[i].fbTcpWrite.xError THEN

; // обработка ошибок

END\_IF

END\_CASE

END\_FOR

## TCP-клиент

### Перечисление CLIENT\_STATE

// шаг состояния клиента

{attribute 'strict'}

TYPE CLIENT\_STATE :

(

CREATE := 0,

WAITING := 10,

SEND := 20,

RECEIVE := 30

);

END\_TYPE

### Программа PLC\_PRG

PROGRAM PLC\_PRG

VAR

sClientString: STRING := 'Hello world'; // Строка, отправляемая клиентом

sInverseString: STRING; // Строка, получаемая от сервера

eState: CLIENT\_STATE; // Шаг состояния клиента

fbTcpClient: NBS.TCP\_Client; // ФБ создания TCP-клиента

fbTcpRead: NBS.TCP\_Read; // ФБ чтения данных

fbTcpWrite: NBS.TCP\_Write; // ФБ записи данных

// IP-адрес сервера (измените его на адрес вашего сервера)

stIpServer: NBS.IP\_ADDR := (sAddr := '10.2.5.229');

uiPortClient: UINT := 3000; // Порт клиента

uiPortServer: UINT := 4711; // Порт сервера

xSend: BOOL;

fbSendTrig: R\_TRIG; // Триггер записи

fbResponseTimeout: TON; // Таймер ожидания ответа

END\_VAR

CASE eState OF

CLIENT\_STATE.CREATE: // создаем TCP-клиента

fbTcpClient

(

xEnable := TRUE,

ipAddr := stIpServer,

uiPort := uiPortServer,

);

IF fbTcpClient.xActive THEN

eState := CLIENT\_STATE.WAITING;

ELSIF fbTcpClient.xError THEN

; // обработка ошибок

END\_IF

CLIENT\_STATE.WAITING: // ожидаем команды на запись

fbSendTrig(CLK:=xSend);

IF fbSendTrig.Q THEN

eState := CLIENT\_STATE.SEND;

END\_IF

CLIENT\_STATE.SEND: // отправляем запрос серверу

fbTcpWrite

(

xExecute := TRUE,

hConnection := fbTcpClient.hConnection,

pData := ADR(sClientString),

szSize := SIZEOF(sClientString)

);

IF fbTcpWrite.xDone THEN

fbTcpWrite(xExecute:=FALSE);

fbResponseTimeout(IN:= FALSE);

eState := CLIENT\_STATE.RECEIVE;

ELSIF fbTcpWrite.xError THEN

; // обработка ошибок

END\_IF

CLIENT\_STATE.RECEIVE: // получаем ответ от сервера

// запускаем таймер ожидания ответа

fbResponseTimeout(IN:= TRUE, PT:= T#1s);

fbTcpRead

(

xEnable := TRUE,

hConnection := fbTcpClient.hConnection,

pData := ADR(sInverseString),

szSize := SIZEOF(sInverseString)

);

// если данные получены - ожидаем следующей команды на запись

IF fbTcpRead.xReady OR fbResponseTimeout.Q THEN

eState := CLIENT\_STATE.WAITING;

ELSIF fbTcpRead.xError THEN

; // обработка ошибок

END\_IF

END\_CASE