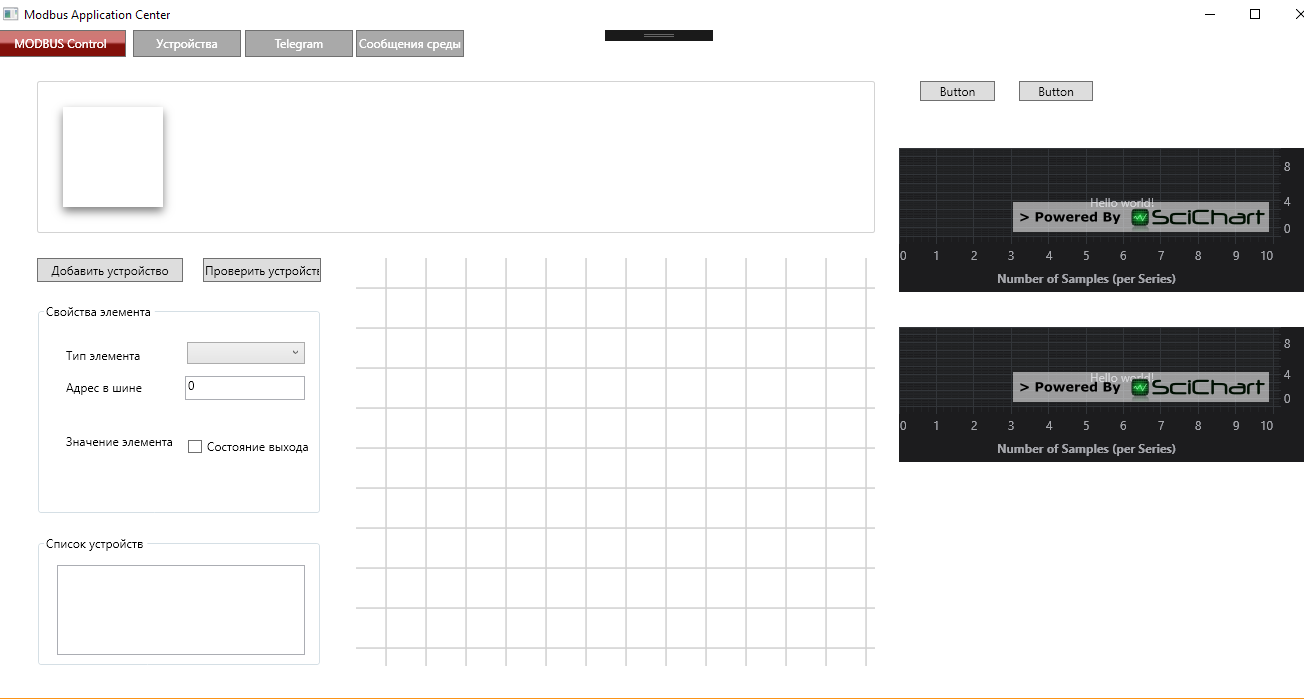
Приложение Modbus Application Center предназначено для общей работы с протоколом Modbus поверх стека TCP/IP.

Это означает, что данное приложение может быть использовано для управления любыми устройствами, которые используют в своей работе промышленный протокол обмена данными Modbus.

На первом этапе разработки элементов приложения были выделены следующие основные функциональные возможности, которые также должны быть отражены в интерфейсе:

* Полотно для графического отображения подключенных и управляемых устройств
* Возможность выбора для добавления в схему конкретных устройств из предварительно подготовленного списка
* Возможность добавлять / редактировать / удалять устройства, которые не находятся в списке предварительно сохраненных вручную
* Отображение списка всех устройств, за которыми ведется наблюдение
* Отображение основных свойств для выбранного на схеме устройства (адрес в шине, значение, тип элемента)
* Возможность проводить калибровку аналоговых устройств по нескольким точкам, задавать измеряемую величину
* Отображение в реальном времени показателей со всех подключенных устройств
* Возможность управления устройствами через мессенджер Telegram
* Возможность управления включением / отключением мониторинга за устройствами
* Возможность по отображению информационных сообщений или сообщений об ошибках приложения
* Возможность предпросмотра выбранного на схеме элемента



Для реализации отображения полотна с упрощенных схематическим обозначением устройств используется элемент canvas с нанесенной на него прямоугольной равномерной сеткой.

Для реализации отображения состояния устройств в режиме реального времени используются возможности графической библиотеки Scichart для построения 2D – графиков.

Для того, чтобы каждый из элементов интерфейса оставался строго на своей позиции при запуске приложения, и чтобы эта позиция не отличалась от той, которая задается элементам в дизайнере формы, использовано выравнивание по верхнему и левому краю формы для всех элементов.

HorizontalAlignment="Left" VerticalAlignment="Top"

Для вынесения реализации получения и задания значений регистров и входов/выходов устройства создан класс ModbusConnection. Этот класс также отвечает за выполнение подключения к устройству, работающему по протоколу Modbus. Достаточными для подключения являются IP – адрес устройства в сети, обычно стандартный порт «502».

public class ModbusConnection

{

private ModbusClient modbusClient;

public ModbusConnection()

{

}

//подключиться к серверу Modbus

public void ConnectToServer(string ip, int port)

{

if (modbusClient == null)

modbusClient = new ModbusClient();

//возможные исключения?

//исключение будет выбрасываться только при попытке подключения

//какие могут быть исключения помимо "timeout exception"

modbusClient.Connect(ip, port);

}

//таймер на считывание расположен в устройствах, соединение, которое устанавливается с протоколом, ничего не знает о работающих таймерах

public int ReadDiscreteOutput(int address)

{

bool[] output = modbusClient.ReadCoils(address, 1);

return Convert.ToInt32(output[0]);

}

}

Изначально предполагалось, что таймеры для доступа к значениям, которые показывают устройства, будут определены и запущены в этом классе, и будут храниться в отдельной структуре, например в списке System.Timers Timer<List>. Однако, у каждого из элементов, к которому можно обратиться по адресу, таймеру могут быть заданы параметры, такие как время обновления значения.

С одной стороны, удобнее управлять всеми таймерами из одного участка кода, т.к. в приложении существуют 2 функции, которые либо останавливают все запущенные таймеры, либо наоборот их запускают. Также при объявлении нового таймера, в аргументе передается функция, которая вызывается каждый раз по таймауту.

Реализация данной функции находится в классе ModbusConnection, однако в случае вызова этого метода из класса подключенного устройства с определенным адресом в схеме, можно вызвать этот метод – для этого нужно:

* В классе устройства, например, DigitalOutputDevice указать ссылку на соединение с основным устройством
* Вызвать соответствующий метод с передачей адреса, по которому располагается нужное устройство. Например, int DOvalue = connection.ReadDiscreteOutput(deviceAddress);

Таким образом, вся реализация, которая относится к аппаратнозависимым особенностям и работе с типами возвращаемых значений, возлагается на класс, осуществляющий непосредственное подключение и соединение с устройством по протоколу. Более обобщенно, такой класс полностью реализует работу с устройством по промышленному протоколу обмена данными.

На этапе создания таймера для нового добавляемого дискретного устройства схемы, а также на этапе считывания значения с устройства с определенными промежутками, класс DigitalOutputDevice содержит код:

public class DigitalOutputDevice

{

string deviceName;

int deviceAddress;

int deviceValue;

ModbusConnection connection;

SciChartSurface DOChart2D;

Timer updateTimer;

public DigitalOutputDevice(string name, int address, int value, ModbusConnection connection)

//SciChartSurface surface

{

deviceName = name;

deviceAddress = address;

deviceValue = value;

this.connection = connection;

//DOChart2D = surface;

updateTimer = new Timer();

updateTimer.Elapsed += new ElapsedEventHandler(UpdateDOChart);

updateTimer.Interval = 5000;

updateTimer.Enabled = true;

}

public void UpdateDOChart(object source, ElapsedEventArgs e)

{

//получить значение с помощью протокола Modbus

int DOvalue = connection.ReadDiscreteOutput(deviceAddress);

//DOChart2D.

}

}

При запуске приложения подразумевается, что пользователь работает с одним общим устройством, к которому подключаются остальные. Для этого создан класс Schema. При этом схема хранит структуры данных, которые позволяют обращаться к каждому из её элементов.

На этапе проверки работоспособности доступа к элементам сервера Modbus выполняется автоматическое подключение – заглушка к устройству по предварительно записанным IP – адресу и TCP – порту в классе Schema.

Схема, в любому случае, отвечает за подключение к устройству. Поэтому дальнейшим изменением будет создание отдельных методов, которые устанавливают либо разрывают соединение с удаленным устройством.

public class Schema

{

private static Schema schema;

public ModbusConnection connection;

public List<DigitalOutputDevice> DOlist;

public List<DigitalInputDevice> DIlist;

public List<AnalogOutputDevice> AOlist;

public List<AnalogInputDevice> AIlist;

public Schema()

{

//создание экземпляра подключения

connection = new ModbusConnection();

//заглушка на подключение к серверу Modbus

connection.ConnectToServer("127.0.0.10", 502);

//инициализация списков с устройствами

DOlist = new List<DigitalOutputDevice>();

DIlist = new List<DigitalInputDevice>();

AOlist = new List<AnalogOutputDevice>();

AIlist = new List<AnalogInputDevice>();

}

public void addDO(string name, int address, int value)

{

DOlist.Add(new DigitalOutputDevice(name, address, value, connection));

}

}

Так как общая схема, которая содержит описание всех внутренних устройств, может быть только одна, то правильным решением будет использовать паттерн проектирования приложения Singleton для того, чтобы ограничить количество одновременно доступных для создания экземпляров класса Schema. В таком случае будет затруднительно в дальнейшем производить сохранение уже полностью созданной схемы, которая может содержать в себе от нескольких подключенных до десятков устройств. Это обусловлено тем, что для сохранения сериализованных данных в XML – файл необходимо соблюдение как минимум 3 общих условий:

* Доступность класса (public class)
* Доступность сериализуемых полей (public int field)
* Доступность стандартного конструктора класса, принимающего 0 аргументов

Для введения шаблона Singleton для Schema необходимо объявить конструктор, который не принимает аргументов, как приватный.

Поэтому создание экземпляра класса Schema выполняется только в одной части приложения – на главной форме. Выполняется создание при вызове проверки checkSchemaExistence

//метод сначала проверяет, существует ли схема, если нет - то создает её

//схема может существовать только в единственном экземпляре

public void checkSchemaExistence()

{

if (schema\_link == null)

schema\_link = new Schema();

}

Проверка выполняется при создании любого из устройств. Это означает, что любое из устройств может оказаться первым, которое будет помещено на схему. Если устройство добавляется первым, то оно добавляется в структуру данных схемы. Т.к. схемы еще не существует, то сначала создается схема, а затем выполняется добавление в нее объекта.

Начальный код главного окна обрабатывает перенаправление к форме создания в системе нового устройства, а также проверку схемы на наличие:

public partial class MainWindow : Window

{

public Schema schema\_link;

public MainWindow()

{

InitializeComponent();

}

//метод сначала проверяет, существует ли схема, если нет - то создает её

//схема может существовать только в единственном экземпляре

public void checkSchemaExistence()

{

if (schema\_link == null)

schema\_link = new Schema();

}

private void add\_device\_Click(object sender, RoutedEventArgs e)

{

Window add = new AddDevice(this);

add.ShowDialog();

}

}

Класс формы добавления нового устройства AddDevice предоставляет функционал для подстановки в текстовое поле свободного имени устройства, содержит обработку полей адреса, имени, дополнительных параметров, таких как «нужно ли отображать новое устройство на схеме», «нужно ли создавать для нового устройства график мониторинга его значений».

public partial class AddDevice : Window

{

public Schema schema\_link;

private MainWindow mw;

public AddDevice(MainWindow main\_w)

{

InitializeComponent();

mw = main\_w;

schema\_link = main\_w.schema\_link;

//проинициализировать значение для поля device\_name\_box

//инициализация и проверка имени устройства очень важна, т.к. в приложении имя используется в качестве ID устройства

//по имени устройства осуществляется его поиск в списке, его удаление, изменение или вызов его методов

if (schema\_link == null)

{

//схема не не создана => ни одного устройства еще не было добавлено => первое возможное имя для устройства

//- устройство1

device\_name\_box.Text = "устройство1";

}

else

{

int numeration\_count = schema\_link.DOlist.Count;

numeration\_count += schema\_link.DIlist.Count;

numeration\_count += schema\_link.AOlist.Count;

numeration\_count += schema\_link.AIlist.Count;

device\_name\_box.Text = "устройство" + (numeration\_count + 1).ToString();

}

}

private void add\_full\_device\_Click(object sender, RoutedEventArgs e)

{

//позиция в combobox

int box\_position = device\_type\_box.SelectedIndex;

//обработать значение combobox

if (box\_position == -1)

{

MessageBox.Show("Выберите один из доступных типов добавляемого устройства");

}

else

{

mw.checkSchemaExistence();

this.schema\_link = mw.schema\_link;

//получение адреса устройства

int deviceAddress = -1;

int.TryParse(position\_box.Text, out deviceAddress);

//перевод номера устройства в адрес

deviceAddress -= 1;

if (box\_position == 0)

{

//добавление DO

schema\_link.addDO(device\_name\_box.Text, deviceAddress, 0);

}

Close();

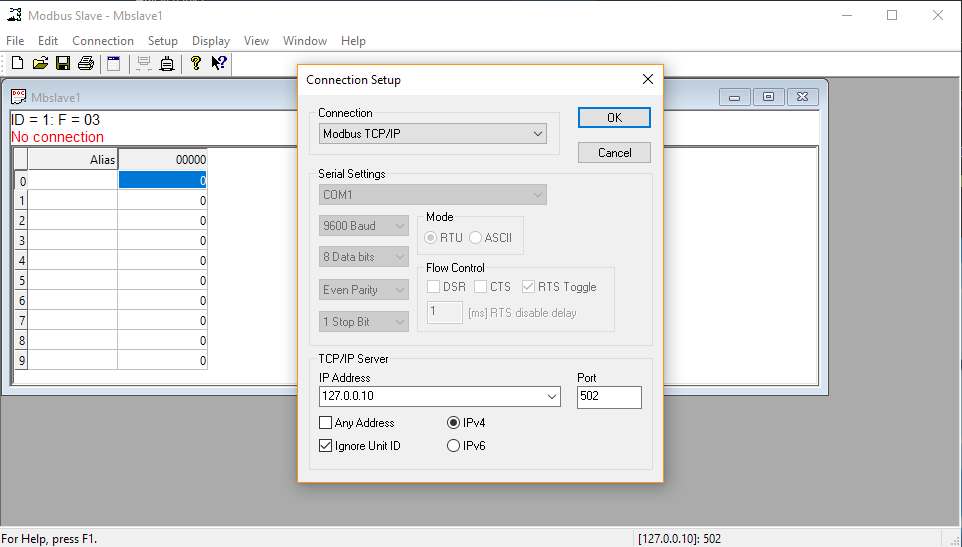
}

}

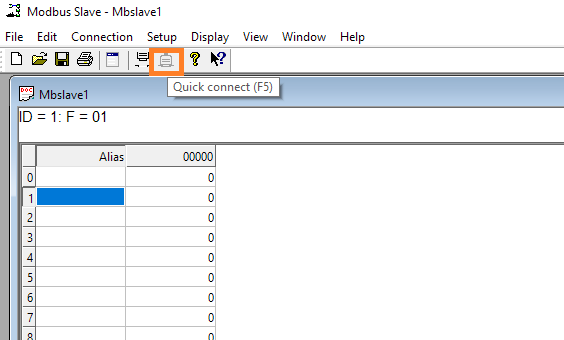
}

Для эмуляции действий промышленных устройств, работающих по протоколу Modbus, используется программное обеспечение modbustools. Эмуляция сервера выполняется с помощью приложения Modbus Slave.

Через настройки соединения, расположенных по Hotkey F3, в разделе найстройки TCP/IP Server указывается один из локальных адресов, порт остается без изменений.

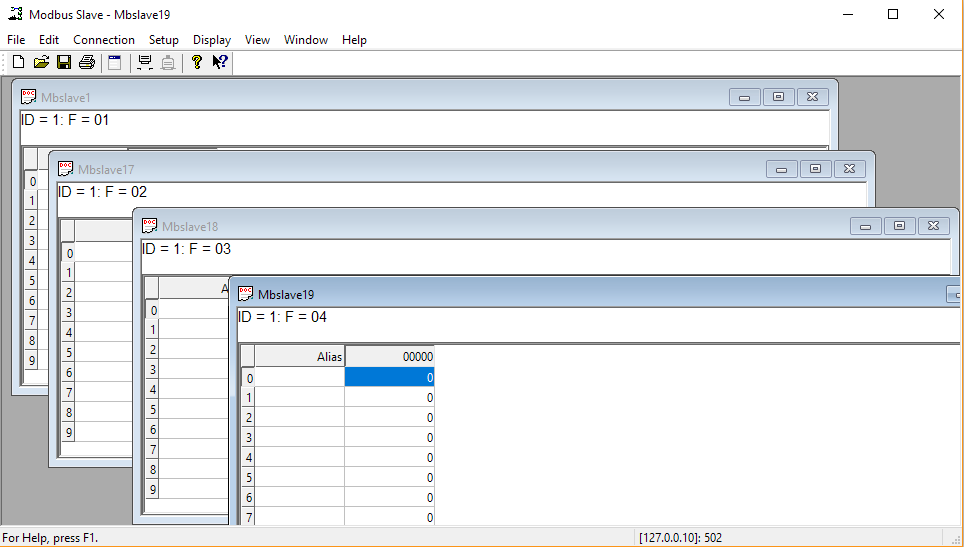


В главной области приложения в случае если в подокнах Mbslave1 …. MbslaveN отображается состояние Not Connected, необходимо выполнить подключение вручную через Quick Connect



В рабочей области могут быть одновременно размещены сразу несколько адресных пространств, например, F=01 функция для работы с Coils или Digital Outputs, F = 02 функция для обращения к Digital Inputs.

Столбец Alias может быть заполнен любым текстом, т.к. не имеет отношения к самому протоколу и служит для различия виртуальных устройств внутри приложения.



В примере выше в рабочую область были добавлены окна для обращения ко всем типам виртуальных устройств, поддерживаемых протоколом. У каждого из типов устройств свое адресное пространство, к которого начинается нумерация первого порта подключения.

В данном примере выполнено добавление по 10 выходов/выходов или регистров записи/хранения.

Реализация вывода значений устройств на динамических графиков реального времени позволяет отслеживать состояние устройств.

Создание элемента для вывода состояния устройства на языке разметки XAML выглядит следующим образом:

<Window x:Class="SciChart.Tutorial.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:s="http://schemas.abtsoftware.co.uk/scichart"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

<!-- Create the chart surface -->

<!-- where xmlns:s="http://schemas.abtsoftware.co.uk/scichart" -->

<s:SciChartSurface x:Name="sciChartSurface">

<!-- Create an X Axis -->

<s:SciChartSurface.XAxis>

<s:NumericAxis AxisTitle="Number of Samples (per Series)"/>

</s:SciChartSurface.XAxis>

<!-- Create a Y Axis -->

<s:SciChartSurface.YAxis>

<s:NumericAxis AxisTitle="Value"/>

</s:SciChartSurface.YAxis>

<!-- Specify interactivity modifiers -->

<s:SciChartSurface.ChartModifier>

<s:ModifierGroup>

<s:RubberBandXyZoomModifier />

<s:ZoomExtentsModifier />

</s:ModifierGroup>

</s:SciChartSurface.ChartModifier>

<!-- Add annotations hints to the user -->

<s:SciChartSurface.Annotations>

<s:TextAnnotation Text="Hello world!" X1="5.0" Y1="5.0"/>

</s:SciChartSurface.Annotations>

</s:SciChartSurface>

</Grid>

</Window>

Создание такого же элемента через подход Code First выглядит следующим образом:

// Create the chart surface

var sciChartSurface = new SciChartSurface();

// Create the X and Y Axis

var xAxis = new NumericAxis() { AxisTitle = "Number of Samples (per series)"};

var yAxis = new NumericAxis() { AxisTitle = "Value"};

sciChartSurface.XAxis = xAxis;

sciChartSurface.YAxis = yAxis;

// Specify Interactivity Modifiers

sciChartSurface.ChartModifier = new ModifierGroup(new RubberBandXyZoomModifier(), new ZoomExtentsModifier());

// Add annotation hints to the user

var textAnnotation = new TextAnnotation()

{

Text = "Hello World!",

X1=5.0,

Y1=5.0

};

sciChartSurface.Annotations.Add(textAnnotation);

В создаваемом приложении необходимо динамическое добавление нескольких экземпляров одного и того же элемента граф. интерфейса sciChartSurface с 2-мя осями координат.

При этом не предполагается, что пользователь будет изменять масштаб графика, текстовая аннотация также была добавлена для примера.