ESP32 Ladder Logic Configuration Tool

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 ladder_diagram_app Namespace Reference

Namespaces

- namespace Models
- namespace Services
- namespace UserControls
- namespace Views

Classes

- class App

  Interaction logic for App.xaml.
- class MainWindow

  Main application window for managing ladder diagrams, device communication, and variables.

5.2 ladder_diagram_app.Models Namespace Reference

Namespaces

- namespace CanvasElements
- namespace DeviceElement
- namespace Variables

5.3 ladder_diagram_app.Models.CanvasElements Namespace Reference

Namespaces

- namespace Instances

Classes

- class WiresManager

  Manages a collection of wires in a ladder diagram, providing methods to add, remove, insert, and clear wires.

5.4 ladder_diagram_app.Models.CanvasElements.Instances Namespace Reference

Classes

- class Branch

  Represents a branch node in a ladder diagram, containing two lists of child nodes and visual lines for rendering.
- class LadderElement

  Represents a ladder diagram element (e.g., contact, coil, timer) with an associated image and variable selection ComboBoxes.
- class Node

  Abstract base class for nodes in a ladder diagram, providing common properties and methods for positioning and highlighting.
- class Wire

  Represents a wire in a ladder diagram, connecting nodes with a visual line.

5.5 ladder_diagram_app.Models.DeviceElement Namespace Reference

Classes

- class Device

  Represents a device in a ladder diagram application, encapsulating its properties and configuration.
- class DevicePinManager

  Manages pin mapping options for a device, providing collections of available digital, analog, and one-wire pin names.

5.6 ladder_diagram_app.Models.Variables Namespace Reference

Namespaces

- namespace Instances

Classes

- class VariablesManager

  Manages variables in a ladder diagram application, associating them with a device and maintaining lists for UI components.

5.7 ladder_diagram_app.Models.Variables.Instances Namespace Reference

Classes

- class ADCSensorVariable

  Represents an ADC sensor variable in a ladder diagram, encapsulating sensor-specific properties and validation.
- class BooleanVariable

  Represents a boolean variable in a ladder diagram, encapsulating a true/false value.
- class CounterVariable

  Represents a counter variable in a ladder diagram, encapsulating preset and current values, count direction, and output states.
- class DigitalAnalogInputOutputVariable

  Represents a digital or analog input/output variable in a ladder diagram, associated with a specific pin.
- class NumericVariable

  Represents a numeric variable in a ladder diagram, encapsulating a double-precision value.
- class OneWireInputVariable

  Represents a one-wire input variable in a ladder diagram, associated with a specific pin.
- class TimerVariable

  Represents a timer variable in a ladder diagram, encapsulating preset time, elapsed time, input, and output states.
- class TimeVariable

  Represents a time variable in a ladder diagram, encapsulating a double-precision time value.
- class Variable

  Abstract base class for variables in a ladder diagram, providing common properties and change notification.

5.8 ladder_diagram_app.Services Namespace Reference

Namespaces

- namespace CanvasServices
- namespace CommunicationServices
- namespace ImportExportServices
- namespace MonitorServices

5.9 ladder_diagram_app.Services.CanvasServices Namespace Reference

Classes

- class CanvasElementFinder

  Provides methods to find canvas elements (wires, ladder elements, and branches) based on cursor position in a ladder diagram application.
- class CanvasInteractionManager

  Manages user interactions with the canvas in a ladder diagram application, including dragging, dropping, selecting, and deleting elements.
- class CanvasManager

  Manages the rendering and layout of canvas elements in a ladder diagram application.

5.10 ladder_diagram_app.Services.CommunicationServices Namespace Reference

Namespaces

- namespace BLE
- namespace MQTT

Classes

- class CommunicationServiceFactory

  Factory class for creating instances of IDeviceCommunicationService based on the specified connection type.
- class DeviceCommunicationManager

  Manages device communication, handling connection, disconnection, and configuration exchange for MQTT and BLE protocols.
- interface IDeviceCommunicationService

  Defines the contract for device communication services, supporting connection management and data exchange.

5.11 ladder_diagram_app.Services.CommunicationServices.BLE Namespace Reference

Namespaces

- namespace BluetoothSelection

Classes

- class BleCommunicationService

  Provides Bluetooth Low Energy (BLE) communication services for connecting to and interacting with a BLE device.

5.12 ladder_diagram_app.Services.CommunicationServices.BLE.↩ BluetoothSelection Namespace Reference

Classes

- class BleDeviceWatcher

  Monitors and manages Bluetooth Low Energy (BLE) devices using a device watcher.

5.13 ladder_diagram_app.Services.CommunicationServices.MQTT Namespace Reference

Classes

- class MqttCommunicationService

  Provides MQTT communication services for connecting to and interacting with a device using MQTT protocol.

5.14 ladder_diagram_app.Services.ImportExportServices Namespace Reference

Classes

- class ImportExportService
  Handles importing and exporting of ladder diagram configurations to and from JSON format.

5.15 ladder_diagram_app.Services.MonitorServices Namespace Reference

Classes

- class MonitorDataService
  Processes and displays monitor data received from a device in the main window.
- class OneWireDataService
  Manages one-wire sensor data processing and UI updates for the main window.

5.16 ladder_diagram_app.UserControls Namespace Reference

Classes

- class TimePicker
  A user control for selecting and displaying time in a HH:mm:ss format.

5.17 ladder_diagram_app.Views Namespace Reference

Classes

- class AddParentsWindow
  Window for adding and managing parent devices, centered relative to the owner window.
- class BleDeviceSelectionWindow
  Represents a window for selecting a Bluetooth Low Energy (BLE) device from a list.
- class NotificationWindow
  A customizable notification window that supports various button configurations and input fields.

Enumerations

- enum NotificationButtons {
  None , YesNo , Ok , OneInput ,
  TwoInputs , ThreeInputs }
  Defines the types of button configurations for the notification window.

5.17.1 Enumeration Type Documentation

5.17.1.1 NotificationButtons

enum ladder_diagram_app.Views.NotificationButtons

Defines the types of button configurations for the notification window.

Enumerator

None	
YesNo	
Ok	
OneInput	
TwoInputs	
ThreeInputs	

Definition at line 14 of file NotificationWindow.xaml.cs.

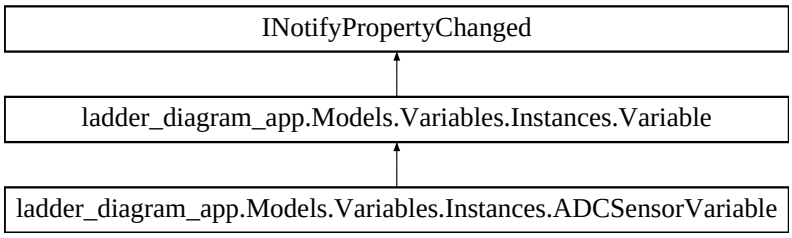
Chapter 6

Class Documentation

6.1 ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable Class Reference

Represents an ADC sensor variable in a ladder diagram, encapsulating sensor-specific properties and validation.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable:



Public Member Functions

- ADCSensorVariable ()

  Initializes a new instance of the ADCSensorVariable class with default values.
- override Dictionary< string, object > ToExportDictionary ()

  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()

  Converts the variable's properties to a dictionary for export purposes.

Properties

- string? SensorType [get, set]

  Gets or sets the type of the sensor, notifying subscribers on change.
- string? PD_SCK [get, set]

  Gets or sets the PD_SCK pin identifier, notifying subscribers on change.
- string? DOUT [get, set]

  Gets or sets the DOUT pin identifier, notifying subscribers on change.
- double MapLow [get, set]

  Gets or sets the low mapping value for the sensor, notifying subscribers on change.
- double MapHigh [get, set]

  Gets or sets the high mapping value for the sensor, notifying subscribers on change.
- double Gain [get, set]

  Gets or sets the gain value for the sensor, notifying subscribers on change.
- string? SamplingRate [get, set]

  Gets or sets the sampling rate of the sensor, notifying subscribers on change.
- string Value [get, set]

  Gets or sets the current value of the sensor, notifying subscribers on change.
- bool IsValid [get]

  Gets a value indicating whether the variable is valid based on required properties.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]

  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]

  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]

  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- string? _sensorType

  Stores the type of the sensor.
- string? _pdSck

  Stores the PD_SCK pin identifier.
- string? _dout

  Stores the DOUT pin identifier.
- double _mapLow

  Stores the low mapping value for the sensor.
- double _mapHigh

  Stores the high mapping value for the sensor.
- double _gain

  Stores the gain value for the sensor.
- string? _samplingRate

  Stores the sampling rate of the sensor.
- string _value

  Stores the current value of the sensor.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.1.1 Detailed Description

Represents an ADC sensor variable in a ladder diagram, encapsulating sensor-specific properties and validation.

Definition at line 6 of file ADCSensorVariable.cs.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 ADCSensorVariable()

ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.ADCSensorVariable () [inline]

Initializes a new instance of the ADCSensorVariable class with default values.

Definition at line 123 of file ADCSensorVariable.cs.

6.1.3 Member Function Documentation

6.1.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.ToExport↩ Dictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 145 of file ADCSensorVariable.cs.

6.1.4 Member Data Documentation

6.1.4.1 _dout

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._dout [private]

Stores the DOUT pin identifier.

Definition at line 21 of file ADCSensorVariable.cs.

6.1.4.2 _gain

double ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._gain [private]

Stores the gain value for the sensor.

Definition at line 36 of file ADCSensorVariable.cs.

6.1.4.3 _mapHigh

double ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._mapHigh [private]

Stores the high mapping value for the sensor.

Definition at line 31 of file ADCSensorVariable.cs.

6.1.4.4 _mapLow

double ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._mapLow [private]

Stores the low mapping value for the sensor.

Definition at line 26 of file ADCSensorVariable.cs.

6.1.4.5 _pdSck

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._pdSck [private]

Stores the PD_SCK pin identifier.

Definition at line 16 of file ADCSensorVariable.cs.

6.1.4.6 _samplingRate

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._samplingRate [private]

Stores the sampling rate of the sensor.

Definition at line 41 of file ADCSensorVariable.cs.

6.1.4.7 _sensorType

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._sensorType [private]

Stores the type of the sensor.

Definition at line 11 of file ADCSensorVariable.cs.

6.1.4.8 _value

string ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable._value [private]

Stores the current value of the sensor.

Definition at line 46 of file ADCSensorVariable.cs.

6.1.5 Property Documentation

6.1.5.1 DOUT

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.DOUT [get], [set]

Gets or sets the DOUT pin identifier, notifying subscribers on change.

Definition at line 69 of file ADCSensorVariable.cs.

6.1.5.2 Gain

double ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.Gain [get], [set]

Gets or sets the gain value for the sensor, notifying subscribers on change.

Definition at line 96 of file ADCSensorVariable.cs.

6.1.5.3 IsValid

bool ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.IsValid [get]

Gets a value indicating whether the variable is valid based on required properties.

Definition at line 135 of file ADCSensorVariable.cs.

6.1.5.4 MapHigh

double ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.MapHigh [get], [set]

Gets or sets the high mapping value for the sensor, notifying subscribers on change.

Definition at line 87 of file ADCSensorVariable.cs.

6.1.5.5 MapLow

double ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.MapLow [get], [set]

Gets or sets the low mapping value for the sensor, notifying subscribers on change.

Definition at line 78 of file ADCSensorVariable.cs.

6.1.5.6 PD_SCK

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.PD_SCK [get], [set]

Gets or sets the PD_SCK pin identifier, notifying subscribers on change.

Definition at line 60 of file ADCSensorVariable.cs.

6.1.5.7 SamplingRate

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.SamplingRate [get], [set]

Gets or sets the sampling rate of the sensor, notifying subscribers on change.

Definition at line 105 of file ADCSensorVariable.cs.

6.1.5.8 SensorType

string? ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.SensorType [get], [set]

Gets or sets the type of the sensor, notifying subscribers on change.

Definition at line 51 of file ADCSensorVariable.cs.

6.1.5.9 Value

string ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable.Value [get], [set]

Gets or sets the current value of the sensor, notifying subscribers on change.

Definition at line 114 of file ADCSensorVariable.cs.

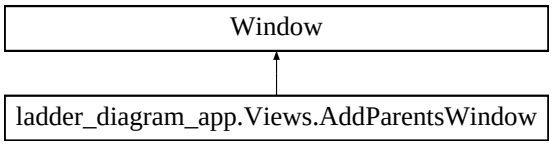
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/ADCSensorVariable.cs

6.2 ladder_diagram_app.Views.AddParentsWindow Class Reference

Window for adding and managing parent devices, centered relative to the owner window.

Inheritance diagram for ladder_diagram_app.Views.AddParentsWindow:



Classes

- struct MONITORINFO

  Contains information about a monitor's size and work area.
- struct RECT

  Represents a rectangle with left, top, right, and bottom coordinates.

Public Member Functions

- AddParentsWindow (List< string > parentDevices, Window owner)

  Initializes a new instance of the AddParentsWindow class.

Properties

- ObservableCollection< string > ParentDevices [get, set]

  Gets the observable collection of parent devices.

Private Member Functions

- static IntPtr MonitorFromWindow (IntPtr hwnd, uint dwFlags)
- static bool GetMonitorInfo (IntPtr hMonitor, ref MONITORINFO lpmi)
- void AddParentDevice_Click (object sender, RoutedEventArgs e)

  Adds a new parent device when the Add button is clicked.
- void DeleteParentDevice_Click (object sender, RoutedEventArgs e)

  Removes a parent device when the Delete button is clicked.
- void Save_Click (object sender, RoutedEventArgs e)

  Saves the parent devices list and closes the window.
- void Cancel_Click (object sender, RoutedEventArgs e)

  Closes the window without saving changes.
- void UpdatePosition ()

  Updates the window position to center it relative to the owner window or monitor.
- void Owner_PositionOrSizeChanged (object sender, EventArgs e)

  Handles changes in the owner window's position or size to reposition the dialog.
- void Owner_StateChanged (object sender, EventArgs e)

  Handles changes in the owner window's state (e.g., maximized/minimized) to reposition the dialog.

Private Attributes

- readonly List< string > _parentDevices

Static Private Attributes

- const uint MONITOR_DEFAULTTONEAREST = 2

6.2.1 Detailed Description

Window for adding and managing parent devices, centered relative to the owner window.

Definition at line 15 of file AddParentsWindow.xaml.cs.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 AddParentsWindow()

ladder_diagram_app.Views.AddParentsWindow.AddParentsWindow (
List< string > parentDevices,
Window owner) [inline]

Initializes a new instance of the AddParentsWindow class.

Parameters

parentDevices	The list of parent devices to manage.
owner	The owner window for positioning and event handling.

Definition at line 61 of file AddParentsWindow.xaml.cs.

6.2.3 Member Function Documentation

6.2.3.1 AddParentDevice_Click()

void ladder_diagram_app.Views.AddParentsWindow.AddParentDevice_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Adds a new parent device when the Add button is clicked.

Parameters

sender	The button that triggered the event.
e	The routed event arguments.

Definition at line 127 of file AddParentsWindow.xaml.cs.

6.2.3.2 Cancel_Click()

void ladder_diagram_app.Views.AddParentsWindow.Cancel_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Closes the window without saving changes.

Parameters

sender	The button that triggered the event.
e	The routed event arguments.

Definition at line 177 of file AddParentsWindow.xaml.cs.

6.2.3.3 DeleteParentDevice_Click()

void ladder_diagram_app.Views.AddParentsWindow.DeleteParentDevice_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Removes a parent device when the Delete button is clicked.

Parameters

sender	The button that triggered the event.
e	The routed event arguments.

Definition at line 151 of file AddParentsWindow.xaml.cs.

6.2.3.4 GetMonitorInfo()

static bool ladder_diagram_app.Views.AddParentsWindow.GetMonitorInfo (
IntPtr hMonitor,
ref MONITORINFO lpmi) [private]

6.2.3.5 MonitorFromWindow()

static IntPtr ladder_diagram_app.Views.AddParentsWindow.MonitorFromWindow (
IntPtr hwnd,
uint dwFlags) [private]

6.2.3.6 Owner_PositionOrSizeChanged()

void ladder_diagram_app.Views.AddParentsWindow.Owner_PositionOrSizeChanged (
object sender,
EventArgs e) [inline], [private]

Handles changes in the owner window's position or size to reposition the dialog.

Parameters

sender	The object that triggered the event.
e	The event arguments.

Definition at line 238 of file AddParentsWindow.xaml.cs.

6.2.3.7 Owner_StateChanged()

void ladder_diagram_app.Views.AddParentsWindow.Owner_StateChanged (
object sender,
EventArgs e) [inline], [private]

Handles changes in the owner window's state (e.g., maximized/minimized) to reposition the dialog.

Parameters

sender	The object that triggered the event.
e	The event arguments.

Definition at line 248 of file AddParentsWindow.xaml.cs.

6.2.3.8 Save_Click()

void ladder_diagram_app.Views.AddParentsWindow.Save_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Saves the parent devices list and closes the window.

Parameters

sender	The button that triggered the event.
e	The routed event arguments.

Definition at line 164 of file AddParentsWindow.xaml.cs.

6.2.3.9 UpdatePosition()

void ladder_diagram_app.Views.AddParentsWindow.UpdatePosition () [inline], [private]

Updates the window position to center it relative to the owner window or monitor.

Definition at line 186 of file AddParentsWindow.xaml.cs.

6.2.4 Member Data Documentation

6.2.4.1 _parentDevices

readonly List<string> ladder_diagram_app.Views.AddParentsWindow._parentDevices [private]

Definition at line 50 of file AddParentsWindow.xaml.cs.

6.2.4.2 MONITOR_DEFAULTTONEAREST

const uint ladder_diagram_app.Views.AddParentsWindow.MONITOR_DEFAULTTONEAREST = 2 [static], [private]

Definition at line 24 of file AddParentsWindow.xaml.cs.

6.2.5 Property Documentation

6.2.5.1 ParentDevices

ObservableCollection<string> ladder_diagram_app.Views.AddParentsWindow.ParentDevices [get], [set]

Gets the observable collection of parent devices.

Definition at line 54 of file AddParentsWindow.xaml.cs.

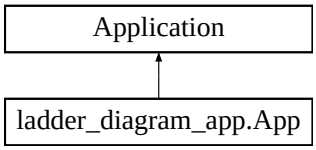
The documentation for this class was generated from the following file:

- ladder_diagram_app/Views/AddParentsWindow.xaml.cs

6.3 ladder_diagram_app.App Class Reference

Interaction logic for App.xaml.

Inheritance diagram for ladder_diagram_app.App:



6.3.1 Detailed Description

Interaction logic for App.xaml.

Definition at line 10 of file App.xaml.cs.

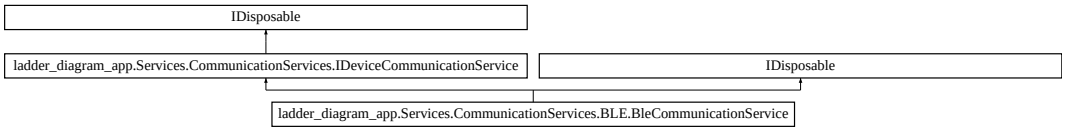
The documentation for this class was generated from the following file:

- ladder_diagram_app/App.xaml.cs

6.4 ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService Class Reference

Provides Bluetooth Low Energy (BLE) communication services for connecting to and interacting with a BLE device.

Inheritance diagram for ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService:



Public Member Functions

- async Task< bool > ConnectAsync (string deviceId)
  Connects to a BLE device asynchronously.
- async Task DisconnectAsync ()
  Disconnects from the BLE device and cleans up resources.
- async Task RequestConfigurationAsync ()
  Requests the configuration from the device.
- async Task< bool > SendConfigurationAsync (string configJson)
  Sends a JSON configuration to the device.
- void Dispose ()
  Disposes of the service and releases resources.

Properties

- bool IsConnected [get, private set]
  Gets a value indicating whether the service is connected to a device.
- string ConnectionType [get]
  Gets the type of connection, which is "BLE".

Events

- EventHandler< string >? ConfigurationReceived
- EventHandler< string >? MonitorDataReceived
- EventHandler< string >? OneWireDataReceived
- EventHandler< bool >? ConnectionStatusChanged

Events inherited from ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService

- EventHandler< string > ConfigurationReceived
  Occurs when configuration data is received from the device.
- EventHandler< string > MonitorDataReceived
  Occurs when monitor data is received from the device.
- EventHandler< string > OneWireDataReceived
  Occurs when one-wire data is received from the device.
- EventHandler< bool > ConnectionStatusChanged
  Occurs when the connection status changes.

Private Member Functions

- async Task< bool > ConnectToDeviceWithRetry (string deviceId, int maxRetries=5)
  Attempts to connect to a BLE device with retries.
- async Task< GattDeviceServicesResult?> GetServicesWithRetry (BluetoothLEDevice device, int maxRetries=5)
  Retrieves GATT services from the device with retries.
- async Task< bool > SetupCharacteristics (GattDeviceServicesResult servicesResult)
  Sets up the required GATT characteristics for communication.
- async Task ReadMonitorBle ()
  Continuously reads monitor data from the device.
- async Task ReadOneWireBle ()
  Continuously reads one-wire data from the device.

Private Attributes

- BluetoothLEDevice? _bleDevice
- GattCharacteristic? _readConfigurationCharacteristic
- GattCharacteristic? _writeConfigurationCharacteristic
- GattCharacteristic? _readMonitorCharacteristic
- GattCharacteristic? _readOneWireCharacteristic
- readonly Guid _serviceUuid = Guid.Parse("00001234-0000-1000-8000-00805f9b34fb")
- readonly Guid _readConfigurationCharUuid = Guid.Parse("0000FFF1-0000-1000-8000-00805f9b34fb")
- readonly Guid _writeConfigurationCharUuid = Guid.Parse("0000FFF2-0000-1000-8000-00805f9b34fb")
- readonly Guid _readMonitorCharUuid = Guid.Parse("0000FFF3-0000-1000-8000-00805f9b34fb")
- readonly Guid _readOneWireCharUuid = Guid.Parse("0000FFF4-0000-1000-8000-00805f9b34fb")
- StringBuilder _jsonConfigurationBuffer = new StringBuilder()
- StringBuilder _jsonMonitorBuffer = new StringBuilder()
- StringBuilder _jsonOneWireBuffer = new StringBuilder()
- bool _monitorTaskRunning = false
- bool _oneWireTaskRunning = false
- readonly int ChunkSize = 250

6.4.1 Detailed Description

Provides Bluetooth Low Energy (BLE) communication services for connecting to and interacting with a BLE device.

Definition at line 14 of file BleCommunicationService.cs.

6.4.2 Member Function Documentation

6.4.2.1 ConnectAsync()

async Task< bool > ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.ConnectAsync (
string deviceId) [inline]

Connects to a BLE device asynchronously.

Parameters

deviceId	The ID of the device to connect to.

Returns

True if connection is successful, otherwise false.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 105 of file BleCommunicationService.cs.

6.4.2.2 ConnectToDeviceWithRetry()

async Task< bool > ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.ConnectToDeviceWithRetry (
string deviceId,
int maxRetries = 5) [inline], [private]

Attempts to connect to a BLE device with retries.

Parameters

deviceId	The ID of the device to connect to.
maxRetries	Maximum number of retry attempts.

Returns

True if connection is successful, otherwise false.

Definition at line 58 of file BleCommunicationService.cs.

6.4.2.3 DisconnectAsync()

async Task ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.DisconnectAsync () [inline]

Disconnects from the BLE device and cleans up resources.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 167 of file BleCommunicationService.cs.

6.4.2.4 Dispose()

void ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.Dispose () [inline]

Disposes of the service and releases resources.

Definition at line 458 of file BleCommunicationService.cs.

6.4.2.5 GetServicesWithRetry()

async Task< GattDeviceServicesResult?> ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunication↩
Service.GetServicesWithRetry (
BluetoothLEDevice device,
int maxRetries = 5) [inline], [private]

Retrieves GATT services from the device with retries.

Parameters

device	The BLE device to query.
maxRetries	Maximum number of retry attempts.

Returns

The GATT services result, or null if unsuccessful.

Definition at line 86 of file BleCommunicationService.cs.

6.4.2.6 ReadMonitorBle()

async Task ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.ReadMonitorBle () [inline], [private]

Continuously reads monitor data from the device.

Definition at line 344 of file BleCommunicationService.cs.

6.4.2.7 ReadOneWireBle()

async Task ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.ReadOneWireBle () [inline], [private]

Continuously reads one-wire data from the device.

Definition at line 401 of file BleCommunicationService.cs.

6.4.2.8 RequestConfigurationAsync()

async Task ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.RequestConfiguration↩ Async () [inline]

Requests the configuration from the device.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 256 of file BleCommunicationService.cs.

6.4.2.9 SendConfigurationAsync()

async Task< bool > ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.Send↩ ConfigurationAsync (
string configJson) [inline]

Sends a JSON configuration to the device.

Parameters

configJson	The JSON configuration string to send.

Returns

True if sending is successful, otherwise false.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 309 of file BleCommunicationService.cs.

6.4.2.10 SetupCharacteristics()

async Task< bool > ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.Setup↩ Characteristics (
GattDeviceServicesResult servicesResult) [inline], [private]

Sets up the required GATT characteristics for communication.

Parameters

servicesResult	The GATT services result.

Returns

True if setup is successful, otherwise false.

Definition at line 208 of file BleCommunicationService.cs.

6.4.3 Member Data Documentation

6.4.3.1 _bleDevice

BluetoothLEDevice? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._bleDevice [private]

Definition at line 16 of file BleCommunicationService.cs.

6.4.3.2 _jsonConfigurationBuffer

StringBuilder ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._jsonConfiguration↩Buffer = new StringBuilder() [private]

Definition at line 43 of file BleCommunicationService.cs.

6.4.3.3 _jsonMonitorBuffer

StringBuilder ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._jsonMonitorBuffer = new StringBuilder() [private]

Definition at line 44 of file BleCommunicationService.cs.

6.4.3.4 _jsonOneWireBuffer

StringBuilder ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._jsonOneWireBuffer = new StringBuilder() [private]

Definition at line 45 of file BleCommunicationService.cs.

6.4.3.5 _monitorTaskRunning

bool ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._monitorTaskRunning = false [private]

Definition at line 47 of file BleCommunicationService.cs.

6.4.3.6 _oneWireTaskRunning

bool ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._oneWireTaskRunning = false [private]

Definition at line 48 of file BleCommunicationService.cs.

6.4.3.7 _readConfigurationCharacteristic

GattCharacteristic? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._readConfigurationCharacteristic [private]

Definition at line 17 of file BleCommunicationService.cs.

6.4.3.8 _readConfigurationCharUuid

readonly Guid ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._readConfigurationCharUuid = Guid.Parse("0000FFF1-0000-1000-8000-00805f9b34fb") [private]

Definition at line 23 of file BleCommunicationService.cs.

6.4.3.9 _readMonitorCharacteristic

GattCharacteristic? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._readMonitorCharacteristic [private]

Definition at line 19 of file BleCommunicationService.cs.

6.4.3.10 _readMonitorCharUuid

readonly Guid ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._readMonitorCharUuid = Guid.Parse("0000FFF3-0000-1000-8000-00805f9b34fb") [private]

Definition at line 25 of file BleCommunicationService.cs.

6.4.3.11 _readOneWireCharacteristic

GattCharacteristic? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._readOneWireCharacteristic [private]

Definition at line 20 of file BleCommunicationService.cs.

6.4.3.12 _readOneWireCharUuid

readonly Guid ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._readOneWireCharUuid = Guid.Parse("0000FFF4-0000-1000-8000-00805f9b34fb") [private]

Definition at line 26 of file BleCommunicationService.cs.

6.4.3.13 _serviceUuid

readonly Guid ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._serviceUuid = Guid.Parse("00001234-0000-1000-8000-00805f9b34fb") [private]

Definition at line 22 of file BleCommunicationService.cs.

6.4.3.14 _writeConfigurationCharacteristic

GattCharacteristic? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._write↩ ConfigurationCharacteristic [private]

Definition at line 18 of file BleCommunicationService.cs.

6.4.3.15 _writeConfigurationCharUuid

readonly Guid ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService._writeConfiguration↩ CharUuid = Guid.Parse("0000FFF2-0000-1000-8000-00805f9b34fb") [private]

Definition at line 24 of file BleCommunicationService.cs.

6.4.3.16 ChunkSize

readonly int ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.ChunkSize = 250 [private]

Definition at line 50 of file BleCommunicationService.cs.

6.4.4 Property Documentation

6.4.4.1 ConnectionType

string ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.ConnectionType [get]

Gets the type of connection, which is "BLE".

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 41 of file BleCommunicationService.cs.

6.4.4.2 IsConnected

bool ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.IsConnected [get], [private set]

Gets a value indicating whether the service is connected to a device.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 36 of file BleCommunicationService.cs.

6.4.5 Event Documentation

6.4.5.1 ConfigurationReceived

EventHandler<string>? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.↩
ConfigurationReceived

Definition at line 28 of file BleCommunicationService.cs.

6.4.5.2 ConnectionStatusChanged

EventHandler<bool>? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.↩
ConnectionStatusChanged

Definition at line 31 of file BleCommunicationService.cs.

6.4.5.3 MonitorDataReceived

EventHandler<string>? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.Monitor↩
DataReceived

Definition at line 29 of file BleCommunicationService.cs.

6.4.5.4 OneWireDataReceived

EventHandler<string>? ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService.One↩
WireDataReceived

Definition at line 30 of file BleCommunicationService.cs.

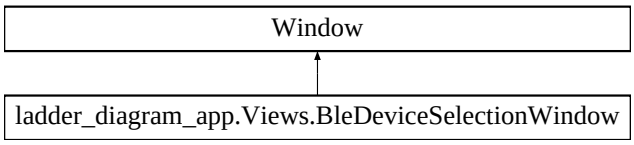
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CommunicationServices/BLE/BleCommunicationService.cs

6.5 ladder_diagram_app.Views.BleDeviceSelectionWindow Class Reference

Represents a window for selecting a Bluetooth Low Energy (BLE) device from a list.

Inheritance diagram for ladder_diagram_app.Views.BleDeviceSelectionWindow:



Public Member Functions

- BleDeviceSelectionWindow (ObservableCollection< DeviceInformation > devices, Window owner)
  Initializes a new instance of the BleDeviceSelectionWindow class.

Properties

- DeviceInformation? SelectedDevice [get, private set]
  Gets the selected BLE device information.

Private Member Functions

- void InitializeComponents ()
  Sets up the UI components for the BLE device selection window.

Private Attributes

- readonly ObservableCollection< DeviceInformation > _devices
  The collection of available BLE devices.

6.5.1 Detailed Description

Represents a window for selecting a Bluetooth Low Energy (BLE) device from a list.

Definition at line 13 of file BleDeviceSelectionWindow.cs.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 BleDeviceSelectionWindow()

ladder_diagram_app.Views.BleDeviceSelectionWindow.BleDeviceSelectionWindow (
ObservableCollection< DeviceInformation > devices,
Window owner) [inline]

Initializes a new instance of the BleDeviceSelectionWindow class.

Parameters

devices	The collection of BLE devices to display.
owner	The parent window that owns this dialog.

Definition at line 30 of file BleDeviceSelectionWindow.cs.

6.5.3 Member Function Documentation

6.5.3.1 InitializeComponents()

void ladder_diagram_app.Views.BleDeviceSelectionWindow.InitializeComponents () [inline], [private]

Sets up the UI components for the BLE device selection window.

Definition at line 40 of file BleDeviceSelectionWindow.cs.

6.5.4 Member Data Documentation

6.5.4.1 _devices

readonly ObservableCollection<DeviceInformation> ladder_diagram_app.Views.BleDeviceSelectionWindow._devices [private]

The collection of available BLE devices.

Definition at line 23 of file BleDeviceSelectionWindow.cs.

6.5.5 Property Documentation

6.5.5.1 SelectedDevice

DeviceInformation? ladder_diagram_app.Views.BleDeviceSelectionWindow.SelectedDevice [get], [private set]

Gets the selected BLE device information.

Definition at line 18 of file BleDeviceSelectionWindow.cs.

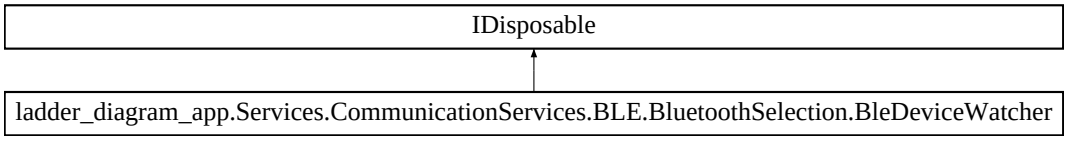
The documentation for this class was generated from the following file:

- ladder_diagram_app/Views/BleDeviceSelectionWindow.cs

6.6 ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher Class Reference

Monitors and manages Bluetooth Low Energy (BLE) devices using a device watcher.

Inheritance diagram for ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher:



Public Member Functions

- void InitializeBle ()

  Initializes and starts the BLE device watcher if not already running.
- void StopWatcher ()

  Stops the device watcher if it is running.
- void Dispose ()

  Disposes of the device watcher and releases resources.

Properties

- ObservableCollection< DeviceInformation > Devices [get]

  Gets the collection of discovered BLE devices.

Private Member Functions

- void DeviceWatcher_Added (DeviceWatcher sender, DeviceInformation deviceInfo)

  Handles the addition of a new BLE device.
- void DeviceWatcher_Updated (DeviceWatcher sender, DeviceInformationUpdate deviceInfo↩ Update)

  Handles updates to an existing BLE device.
- void DeviceWatcher_Removed (DeviceWatcher sender, DeviceInformationUpdate deviceInfo↩ Update)

  Handles the removal of a BLE device.

Private Attributes

- readonly ObservableCollection< DeviceInformation > _devices = new ObservableCollection<Device↩ Information>()
- DeviceWatcher? _deviceWatcher

6.6.1 Detailed Description

Monitors and manages Bluetooth Low Energy (BLE) devices using a device watcher.

Definition at line 10 of file BleDeviceWatcher.cs.

6.6.2 Member Function Documentation

6.6.2.1 DeviceWatcher_Added()

void ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher.DeviceWatcher_↩ Added (

DeviceWatcher sender,
DeviceInformation deviceInfo) [inline], [private]

Handles the addition of a new BLE device.

Parameters

sender	The device watcher that raised the event.
deviceInfo	Information about the added device.

Definition at line 54 of file BleDeviceWatcher.cs.

6.6.2.2 DeviceWatcher_Removed()

void ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher.DeviceWatcher_Removed (
DeviceWatcher sender,
DeviceInformationUpdate deviceInfoUpdate) [inline], [private]

Handles the removal of a BLE device.

Parameters

sender	The device watcher that raised the event.
deviceInfoUpdate	Information about the removed device.

Definition at line 102 of file BleDeviceWatcher.cs.

6.6.2.3 DeviceWatcher_Updated()

void ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher.DeviceWatcher_Updated (
DeviceWatcher sender,
DeviceInformationUpdate deviceInfoUpdate) [inline], [private]

Handles updates to an existing BLE device.

Parameters

sender	The device watcher that raised the event.
deviceInfoUpdate	Updated information about the device.

Definition at line 78 of file BleDeviceWatcher.cs.

6.6.2.4 Dispose()

void ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher.Dispose () [inline]

Disposes of the device watcher and releases resources.

Definition at line 135 of file BleDeviceWatcher.cs.

6.6.2.5 InitializeBle()

void ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher.InitializeBle () [inline]

Initializes and starts the BLE device watcher if not already running.

Definition at line 23 of file BleDeviceWatcher.cs.

6.6.2.6 StopWatcher()

void ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher.StopWatcher () [inline]

Stops the device watcher if it is running.

Definition at line 124 of file BleDeviceWatcher.cs.

6.6.3 Member Data Documentation

6.6.3.1 _devices

readonly ObservableCollection<DeviceInformation> ladder_diagram_app.Services.CommunicationServices.BLE.↩ BluetoothSelection.BleDeviceWatcher._devices = new ObservableCollection<DeviceInformation>() [private]

Definition at line 12 of file BleDeviceWatcher.cs.

6.6.3.2 _deviceWatcher

DeviceWatcher? ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher._↩ deviceWatcher [private]

Definition at line 13 of file BleDeviceWatcher.cs.

6.6.4 Property Documentation

6.6.4.1 Devices

ObservableCollection<DeviceInformation> ladder_diagram_app.Services.CommunicationServices.BLE.Bluetooth↩ Selection.BleDeviceWatcher.Devices [get]

Gets the collection of discovered BLE devices.

Definition at line 18 of file BleDeviceWatcher.cs.

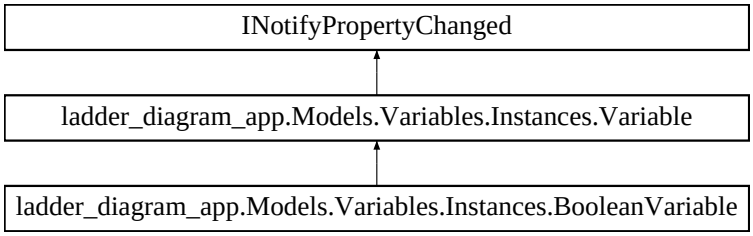
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CommunicationServices/BLE/BluetoothSelection/BleDeviceWatcher.cs

6.7 ladder_diagram_app.Models.Variables.Instances.BooleanVariable Class Reference

Represents a boolean variable in a ladder diagram, encapsulating a true/false value.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.BooleanVariable:



Public Member Functions

- BooleanVariable ()
  Initializes a new instance of the BooleanVariable class with default values.
- override Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Properties

- bool Value [get, set]
  Gets or sets the boolean value of the variable, notifying subscribers on change.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]
  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]
  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]
  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- bool _value
  Stores the boolean value of the variable.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.7.1 Detailed Description

Represents a boolean variable in a ladder diagram, encapsulating a true/false value.

Definition at line 6 of file BooleanVariable.cs.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 BooleanVariable()

ladder_diagram_app.Models.Variables.Instances.BooleanVariable.BooleanVariable () [inline]

Initializes a new instance of the BooleanVariable class with default values.

Definition at line 25 of file BooleanVariable.cs.

6.7.3 Member Function Documentation

6.7.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.BooleanVariable.ToExport↩ Dictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 35 of file BooleanVariable.cs.

6.7.4 Member Data Documentation

6.7.4.1 _value

bool ladder_diagram_app.Models.Variables.Instances.BooleanVariable._value [private]

Stores the boolean value of the variable.

Definition at line 11 of file BooleanVariable.cs.

6.7.5 Property Documentation

6.7.5.1 Value

bool ladder_diagram_app.Models.Variables.Instances.BooleanVariable.Value [get], [set]

Gets or sets the boolean value of the variable, notifying subscribers on change.

Definition at line 16 of file BooleanVariable.cs.

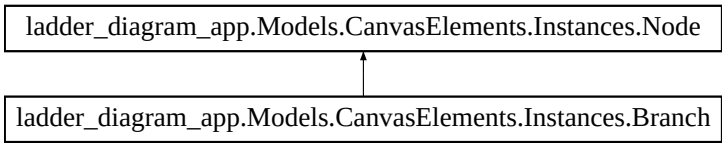
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/BooleanVariable.cs

6.8 ladder_diagram_app.Models.CanvasElements.Instances.Branch Class Reference

Represents a branch node in a ladder diagram, containing two lists of child nodes and visual lines for rendering.

Inheritance diagram for ladder_diagram_app.Models.CanvasElements.Instances.Branch:



Public Member Functions

- Branch ()
  Initializes a new instance of the Branch class with default image and lines.
- void HighlightBranch (bool isUpperLine)
  Highlights either the upper or lower line of the branch with a blue dashed style.
- void UnhighlightBranch ()
  Resets the highlighting of both upper and lower lines to default black.
- void HighlightBranchRecursive ()
  Recursively highlights the current branch and all sub-branches with red lines.
- void UnhighlightBranchRecursive ()
  Recursively resets the highlighting of the current branch and all sub-branches to default black lines.
- bool IsBranchNested (Branch targetBranch)
  Checks if the target branch is the same as this branch or nested within it.

Public Member Functions inherited from ladder_diagram_app.Models.CanvasElements.Instances.Node

- void HighlightNode ()

  Highlights the node by applying a red drop shadow effect to its image.
- void UnhighlightNode ()

  Removes the highlight effect from the node by clearing its image effect.

Properties

- List< Node > Nodes1 [get, set]

  Gets or sets the first list of child nodes in the branch.
- List< Node > Nodes2 [get, set]

  Gets or sets the second list of child nodes in the branch.
- double Y2 [get, set]

  Gets or sets the Y-coordinate of the lower line of the branch, calculated based on child nodes.
- Line UpperLine [get, set]

  Gets or sets the upper horizontal line, Lower Horizontal Line, Left Vertical Line and Right Vertical Line of the branch.
- Line LowerLine [get, set]
- Line LeftLine [get, set]
- Line RightLine [get, set]
- override double Y [get, set]

  Gets or sets the Y-coordinate of the branch, updating lines and Y2 when set.
- override double X [get, set]

  Gets or sets the X-coordinate of the branch, updating lines when set.
- override double Width [get]

  Gets the total width of the branch, based on the wider of Nodes1 or Nodes2.

Properties inherited from ladder_diagram_app.Models.CanvasElements.Instances.Node

- double X [get, set]

  Gets or sets the X-coordinate of the node.
- double Y [get, set]

  Gets or sets the Y-coordinate of the node.
- object? Parent [get, set]

  Gets or sets the parent object of the node, used to track hierarchical relationships.
- double Width [get]

  Gets the width of the node, typically based on its visual representation.
- virtual ? Image Image [get, set]

  Gets or sets the image representing the node visually, if applicable.

Private Member Functions

- void UpdateLines ()

  Updates the coordinates of the branch's lines based on current X, Y, and Y2 values.

Static Private Member Functions

- static double CalculateY2 (List< Node > nodes)

  Calculates the height (Y2 offset) of the branch based on its child nodes.
- static double CalculateTotalWidth (List< Node > nodes)

  Calculates the total width of a list of nodes.

Private Attributes

- double _y

  Stores the Y-coordinate of the branch.
- double _x

  Stores the X-coordinate of the branch.

6.8.1 Detailed Description

Represents a branch node in a ladder diagram, containing two lists of child nodes and visual lines for rendering.

Definition at line 12 of file Branch.cs.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Branch()

ladder_diagram_app.Models.CanvasElements.Instances.Branch.Branch () [inline]

Initializes a new instance of the Branch class with default image and lines.

Definition at line 50 of file Branch.cs.

6.8.3 Member Function Documentation

6.8.3.1 CalculateTotalWidth()

static double ladder_diagram_app.Models.CanvasElements.Instances.Branch.CalculateTotalWidth (
List< Node > nodes) [inline], [static], [private]

Calculates the total width of a list of nodes.

Parameters

nodes	The list of nodes to measure.

Returns

The sum of the widths of all nodes.

Definition at line 183 of file Branch.cs.

6.8.3.2 CalculateY2()

static double ladder_diagram_app.Models.CanvasElements.Instances.Branch.CalculateY2 (
List< Node > nodes) [inline], [static], [private]

Calculates the height (Y2 offset) of the branch based on its child nodes.

Parameters

nodes	The list of nodes to evaluate.

Returns

The calculated height, with a minimum of 125.

Definition at line 141 of file Branch.cs.

6.8.3.3 HighlightBranch()

void ladder_diagram_app.Models.CanvasElements.Instances.Branch.HighlightBranch (
bool isUpperLine) [inline]

Highlights either the upper or lower line of the branch with a blue dashed style.

Parameters

isUpperLine	True to highlight the upper line, false for the lower line.

Definition at line 197 of file Branch.cs.

6.8.3.4 HighlightBranchRecursive()

void ladder_diagram_app.Models.CanvasElements.Instances.Branch.HighlightBranchRecursive () [inline]

Recursively highlights the current branch and all sub-branches with red lines.

Definition at line 234 of file Branch.cs.

6.8.3.5 IsBranchNested()

bool ladder_diagram_app.Models.CanvasElements.Instances.Branch.IsBranchNested (
Branch targetBranch) [inline]

Checks if the target branch is the same as this branch or nested within it.

Parameters

targetBranch	The branch to check for nesting.

Returns

True if the target branch is this branch or a descendant, false otherwise.

Definition at line 278 of file Branch.cs.

6.8.3.6 UnhighlightBranch()

void ladder_diagram_app.Models.CanvasElements.Instances.Branch.UnhighlightBranch () [inline]

Resets the highlighting of both upper and lower lines to default black.

Definition at line 218 of file Branch.cs.

6.8.3.7 UnhighlightBranchRecursive()

void ladder_diagram_app.Models.CanvasElements.Instances.Branch.UnhighlightBranchRecursive () [inline]

Recursively resets the highlighting of the current branch and all sub-branches to default black lines.

Definition at line 255 of file Branch.cs.

6.8.3.8 UpdateLines()

void ladder_diagram_app.Models.CanvasElements.Instances.Branch.UpdateLines () [inline], [private]

Updates the coordinates of the branch's lines based on current X, Y, and Y2 values.

Definition at line 109 of file Branch.cs.

6.8.4 Member Data Documentation

6.8.4.1 _x

double ladder_diagram_app.Models.CanvasElements.Instances.Branch._x [private]

Stores the X-coordinate of the branch.

Definition at line 32 of file Branch.cs.

6.8.4.2 _y

double ladder_diagram_app.Models.CanvasElements.Instances.Branch._y [private]

Stores the Y-coordinate of the branch.

Definition at line 27 of file Branch.cs.

6.8.5 Property Documentation

6.8.5.1 LeftLine

Line ladder_diagram_app.Models.CanvasElements.Instances.Branch.LeftLine [get], [set]

Definition at line 44 of file Branch.cs.

6.8.5.2 LowerLine

Line ladder_diagram_app.Models.CanvasElements.Instances.Branch.LowerLine [get], [set]

Definition at line 43 of file Branch.cs.

6.8.5.3 Nodes1

List<Node> ladder_diagram_app.Models.CanvasElements.Instances.Branch.Nodes1 [get], [set]

Gets or sets the first list of child nodes in the branch.

Definition at line 17 of file Branch.cs.

6.8.5.4 Nodes2

List<Node> ladder_diagram_app.Models.CanvasElements.Instances.Branch.Nodes2 [get], [set]

Gets or sets the second list of child nodes in the branch.

Definition at line 22 of file Branch.cs.

6.8.5.5 RightLine

Line ladder_diagram_app.Models.CanvasElements.Instances.Branch.RightLine [get], [set]

Definition at line 45 of file Branch.cs.

6.8.5.6 UpperLine

Line ladder_diagram_app.Models.CanvasElements.Instances.Branch.UpperLine [get], [set]

Gets or sets the upper horizontal line, Lower Horizontal Line, Left Vertical Line and Right Vertical Line of the branch.

Definition at line 42 of file Branch.cs.

6.8.5.7 Width

override double ladder_diagram_app.Models.CanvasElements.Instances.Branch.Width [get]

Gets the total width of the branch, based on the wider of Nodes1 or Nodes2.

Definition at line 167 of file Branch.cs.

6.8.5.8 X

override double ladder_diagram_app.Models.CanvasElements.Instances.Branch.X [get], [set]

Gets or sets the X-coordinate of the branch, updating lines when set.

Definition at line 96 of file Branch.cs.

6.8.5.9 Y

override double ladder_diagram_app.Models.CanvasElements.Instances.Branch.Y [get], [set]

Gets or sets the Y-coordinate of the branch, updating lines and Y2 when set.

Definition at line 82 of file Branch.cs.

6.8.5.10 Y2

double ladder_diagram_app.Models.CanvasElements.Instances.Branch.Y2 [get], [set]

Gets or sets the Y-coordinate of the lower line of the branch, calculated based on child nodes.

Definition at line 37 of file Branch.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/CanvasElements/Instances/Branch.cs

6.9 ladder_diagram_app.Services.CanvasServices.CanvasElementFinder Class Reference

Provides methods to find canvas elements (wires, ladder elements, and branches) based on cursor position in a ladder diagram application.

Public Member Functions

- CanvasElementFinder (Func< WiresManager > getWiresManager)

  Initializes a new instance of the CanvasElementFinder class with a function to retrieve the WiresManager.
- Wire? FindClosestWire (Point cursorPosition)

  Finds the closest wire to the specified cursor position within a 20-pixel threshold.
- LadderElement? FindClosestElement (Point cursorPosition)

  Finds the closest ladder element to the specified cursor position by checking if the cursor is within the element's bounds.
- Branch? bool IsUpperLine FindClosestBranch (Point cursorPosition, Branch? selected↩ Branch=null)

Public Attributes

- Branch? ClosestBranch

  Finds the closest branch to the specified cursor position, considering upper or lower lines within a 20-pixel threshold.

Private Attributes

- readonly Func< WiresManager > _getWiresManager

6.9.1 Detailed Description

Provides methods to find canvas elements (wires, ladder elements, and branches) based on cursor position in a ladder diagram application.

Definition at line 12 of file CanvasElementFinder.cs.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 CanvasElementFinder()

ladder_diagram_app.Services.CanvasServices.CanvasElementFinder.CanvasElementFinder (
Func< WiresManager > getWiresManager) [inline]

Initializes a new instance of the CanvasElementFinder class with a function to retrieve the WiresManager.

Parameters

getWiresManager	A function that returns the WiresManager instance.

Exceptions

ArgumentNullException	Thrown if getWiresManager is null.

Definition at line 21 of file CanvasElementFinder.cs.

6.9.3 Member Function Documentation

6.9.3.1 FindClosestBranch()

Branch? bool IsUpperLine ladder_diagram_app.Services.CanvasServices.CanvasElementFinder.FindClosestBranch (
Point cursorPosition,
Branch? selectedBranch = null) [inline]

Definition at line 100 of file CanvasElementFinder.cs.

6.9.3.2 FindClosestElement()

LadderElement? ladder_diagram_app.Services.CanvasServices.CanvasElementFinder.FindClosestElement (
Point cursorPosition) [inline]

Finds the closest ladder element to the specified cursor position by checking if the cursor is within the element's bounds.

Parameters

cursorPosition	The cursor position on the canvas.

Returns

The closest LadderElement if found, otherwise null.

Definition at line 43 of file CanvasElementFinder.cs.

6.9.3.3 FindClosestWire()

Wire? ladder_diagram_app.Services.CanvasServices.CanvasElementFinder.FindClosestWire (
Point cursorPosition) [inline]

Finds the closest wire to the specified cursor position within a 20-pixel threshold.

Parameters

cursorPosition	The cursor position on the canvas.

Returns

The closest Wire if within threshold, otherwise null.

Definition at line 31 of file CanvasElementFinder.cs.

6.9.4 Member Data Documentation

6.9.4.1 _getWiresManager

readonly Func<WiresManager> ladder_diagram_app.Services.CanvasServices.CanvasElementFinder._getWiresManager [private]

Definition at line 14 of file CanvasElementFinder.cs.

6.9.4.2 ClosestBranch

Branch? ladder_diagram_app.Services.CanvasServices.CanvasElementFinder.ClosestBranch

Finds the closest branch to the specified cursor position, considering upper or lower lines within a 20-pixel threshold.

Parameters

cursorPosition	The cursor position on the canvas.
selectedBranch	An optional branch to exclude nested branches of.

Returns

A tuple containing the closest Branch and a boolean indicating if the upper line was closest.

Definition at line 100 of file CanvasElementFinder.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CanvasServices/CanvasElementFinder.cs

6.10 ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager Class Reference

Manages user interactions with the canvas in a ladder diagram application, including dragging, dropping, selecting, and deleting elements.

Public Member Functions

- CanvasInteractionManager (Canvas canvas, WiresManager wiresManager, CanvasElementFinder elementFinder, CanvasManager canvasManager, VariablesManager variablesManager)

  Initializes a new instance of the CanvasInteractionManager class.
- bool IsElementSelected ()

  Checks if an element or wire is currently selected.
- void HighlightPosition (Point currentPosition)

  Highlights the wire or branch closest to the specified position.
- void HandleDragOver (DragEventArgs e)

  Handles the drag-over event by highlighting the closest wire or branch.
- void HandleDrop (DragEventArgs e, Window owner)

  Handles the drop event by placing a new element or branch on the canvas.
- void HandleMouseMove (MouseEventArgs e)

  Handles mouse movement, initiating and updating drag operations for elements or wires.
- void HandleMouseLeftButtonUp (MouseButtonEventArgs e, Window owner)

  Handles mouse left button release, finalizing drag operations for elements or wires.
- void HandleMouseLeftButtonDown (MouseButtonEventArgs e, ListView elementListView)

  Handles mouse left button down, selecting elements, wires, or branches.
- void UnselectEverything ()

  Unselects all elements and wires, clearing highlights.
- void DeleteSelected (Window owner)

  Deletes the currently selected element or wire.

Private Attributes

- readonly Canvas _canvas
- readonly WiresManager _wiresManager
- readonly CanvasElementFinder _elementFinder
- readonly CanvasManager _canvasManager
- readonly VariablesManager _variablesManager
- Point _dragStartPosition
- bool _isDragging
- bool _isDraggingWire
- Node? _selectedNode
- Wire? _selectedWire
- Line? _wireLine
- object? _highlightedObject

6.10.1 Detailed Description

Manages user interactions with the canvas in a ladder diagram application, including dragging, dropping, selecting, and deleting elements.

Definition at line 16 of file CanvasInteractionManager.cs.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 CanvasInteractionManager()

ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.CanvasInteractionManager (
Canvas canvas,
WiresManager wiresManager,
CanvasElementFinder elementFinder,
CanvasManager canvasManager,
VariablesManager variablesManager) [inline]

Initializes a new instance of the CanvasInteractionManager class.

Parameters

canvas	The canvas to interact with.
wiresManager	Manages wires on the canvas.
elementFinder	Finds elements based on cursor position.
canvasManager	Manages canvas updates.
variablesManager	Manages variables for ladder elements.

Definition at line 45 of file CanvasInteractionManager.cs.

6.10.3 Member Function Documentation

6.10.3.1 DeleteSelected()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.DeleteSelected (
Window owner) [inline]

Deletes the currently selected element or wire.

Parameters

owner	The owner window for displaying notifications.

Definition at line 661 of file CanvasInteractionManager.cs.

6.10.3.2 HandleDragOver()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.HandleDragOver (
DragEventArgs e) [inline]

Handles the drag-over event by highlighting the closest wire or branch.

Parameters

e	The drag event arguments.

Definition at line 101 of file CanvasInteractionManager.cs.

6.10.3.3 HandleDrop()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.HandleDrop (
DragEventArgs e,
Window owner) [inline]

Handles the drop event by placing a new element or branch on the canvas.

Parameters

e	The drag event arguments.
owner	The owner window for displaying notifications.

Definition at line 111 of file CanvasInteractionManager.cs.

6.10.3.4 HandleMouseLeftButtonDown()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.HandleMouseLeftButtonDown (
MouseButtonEventArgs e,
ListView elementListView) [inline]

Handles mouse left button down, selecting elements, wires, or branches.

Parameters

e	The mouse button event arguments.
elementListView	The ListView to clear selection from.

Definition at line 562 of file CanvasInteractionManager.cs.

6.10.3.5 HandleMouseLeftButtonUp()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.HandleMouseLeftButtonUp (
MouseButtonEventArgs e,
Window owner) [inline]

Handles mouse left button release, finalizing drag operations for elements or wires.

Parameters

e	The mouse button event arguments.
owner	The owner window for displaying notifications.

Definition at line 366 of file CanvasInteractionManager.cs.

6.10.3.6 HandleMouseMove()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.HandleMouseMove (
MouseEventArgs e) [inline]

Handles mouse movement, initiating and updating drag operations for elements or wires.

Parameters

e	The mouse event arguments.

Definition at line 247 of file CanvasInteractionManager.cs.

6.10.3.7 HighlightPosition()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.HighlightPosition (
Point currentPosition) [inline]

Highlights the wire or branch closest to the specified position.

Parameters

currentPosition	The current cursor position on the canvas.

Definition at line 72 of file CanvasInteractionManager.cs.

6.10.3.8 IsElementSelected()

bool ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.IsElementSelected () [inline]

Checks if an element or wire is currently selected.

Returns

True if an element or wire is selected, otherwise false.

Definition at line 63 of file CanvasInteractionManager.cs.

6.10.3.9 UnselectEverything()

void ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager.UnselectEverything () [inline]

Unselects all elements and wires, clearing highlights.

Definition at line 630 of file CanvasInteractionManager.cs.

6.10.4 Member Data Documentation

6.10.4.1 _canvas

readonly Canvas ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._canvas [private]

Definition at line 18 of file CanvasInteractionManager.cs.

6.10.4.2 _canvasManager

readonly CanvasManager ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._canvasManager [private]

Definition at line 21 of file CanvasInteractionManager.cs.

6.10.4.3 _dragStartPosition

Point ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._dragStartPosition [private]

Definition at line 25 of file CanvasInteractionManager.cs.

6.10.4.4 _elementFinder

readonly CanvasElementFinder ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._elementFinder [private]

Definition at line 20 of file CanvasInteractionManager.cs.

6.10.4.5 _highlightedObject

object? ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._highlightedObject [private]

Definition at line 35 of file CanvasInteractionManager.cs.

6.10.4.6 _isDragging

bool ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._isDragging [private]

Definition at line 26 of file CanvasInteractionManager.cs.

6.10.4.7 _isDraggingWire

bool ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._isDraggingWire [private]

Definition at line 27 of file CanvasInteractionManager.cs.

6.10.4.8 _selectedNode

Node? ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._selectedNode [private]

Definition at line 30 of file CanvasInteractionManager.cs.

6.10.4.9 _selectedWire

Wire? ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._selectedWire [private]

Definition at line 31 of file CanvasInteractionManager.cs.

6.10.4.10 _variablesManager

readonly VariablesManager ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._variablesManager [private]

Definition at line 22 of file CanvasInteractionManager.cs.

6.10.4.11 _wireLine

Line? ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._wireLine [private]

Definition at line 32 of file CanvasInteractionManager.cs.

6.10.4.12 _wiresManager

readonly WiresManager ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager._wiresManager [private]

Definition at line 19 of file CanvasInteractionManager.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CanvasServices/CanvasInteractionManager.cs

6.11 ladder_diagram_app.Services.CanvasServices.CanvasManager Class Reference

Manages the rendering and layout of canvas elements in a ladder diagram application.

Public Member Functions

- CanvasManager (Canvas canvas, Grid gridCanvas, WiresManager wiresManager)
  Initializes a new instance of the CanvasManager class.
- void UpdateCanvas ()
  Updates the canvas by adjusting its size, positioning elements, and rendering wires and nodes.

Private Member Functions

- void UpdateElementsParameters (List< Node > nodes, double y1, double startX, Canvas canvas)
  Updates the positions and parameters of nodes (elements and branches) within a wire or branch.
- void DrawNodes (List< Node > nodes, double y1, double startX, Canvas canvas)
  Draws nodes (elements and branches) and their associated UI components on the canvas.

Private Attributes

- readonly Canvas _canvas
- readonly Grid _gridCanvas
- readonly WiresManager _wiresManager

6.11.1 Detailed Description

Manages the rendering and layout of canvas elements in a ladder diagram application.

Definition at line 11 of file CanvasManager.cs.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 CanvasManager()

ladder_diagram_app.Services.CanvasServices.CanvasManager.CanvasManager (
Canvas canvas,
Grid gridCanvas,
WiresManager wiresManager) [inline]

Initializes a new instance of the CanvasManager class.

Parameters

canvas	The canvas to render elements on.
gridCanvas	The grid containing the canvas for sizing reference.
wiresManager	Manages wires and their associated nodes.

Definition at line 23 of file CanvasManager.cs.

6.11.3 Member Function Documentation

6.11.3.1 DrawNodes()

void ladder_diagram_app.Services.CanvasServices.CanvasManager.DrawNodes (
List< Node > nodes,
double y1,
double startX,
Canvas canvas) [inline], [private]

Draws nodes (elements and branches) and their associated UI components on the canvas.

Parameters

nodes	The list of nodes to draw.
y1	The Y-coordinate of the parent wire or branch line.
startX	The starting X-coordinate for positioning nodes.
canvas	The canvas to draw on.

Definition at line 127 of file CanvasManager.cs.

6.11.3.2 UpdateCanvas()

void ladder_diagram_app.Services.CanvasServices.CanvasManager.UpdateCanvas () [inline]

Updates the canvas by adjusting its size, positioning elements, and rendering wires and nodes.

Definition at line 33 of file CanvasManager.cs.

6.11.3.3 UpdateElementsParameters()

void ladder_diagram_app.Services.CanvasServices.CanvasManager.UpdateElementsParameters (
List< Node > nodes,
double y1,
double startX,
Canvas canvas) [inline], [private]

Updates the positions and parameters of nodes (elements and branches) within a wire or branch.

Parameters

nodes	The list of nodes to update.
y1	The Y-coordinate of the parent wire or branch line.
startX	The starting X-coordinate for positioning nodes.
canvas	The canvas for positioning reference.

Definition at line 82 of file CanvasManager.cs.

6.11.4 Member Data Documentation

6.11.4.1 _canvas

readonly Canvas ladder_diagram_app.Services.CanvasServices.CanvasManager._canvas [private]

Definition at line 13 of file CanvasManager.cs.

6.11.4.2 _gridCanvas

readonly Grid ladder_diagram_app.Services.CanvasServices.CanvasManager._gridCanvas [private]

Definition at line 14 of file CanvasManager.cs.

6.11.4.3 _wiresManager

readonly WiresManager ladder_diagram_app.Services.CanvasServices.CanvasManager._wiresManager [private]

Definition at line 15 of file CanvasManager.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CanvasServices/CanvasManager.cs

6.12 ladder_diagram_app.Services.CommunicationServices.CommunicationServiceFactory Class Reference

Factory class for creating instances of IDeviceCommunicationService based on the specified connection type.

Static Public Member Functions

- static IDeviceCommunicationService CreateService (string connectionType)
  Creates a communication service instance for the specified connection type.

6.12.1 Detailed Description

Factory class for creating instances of IDeviceCommunicationService based on the specified connection type.

Definition at line 9 of file CommunicationServiceFactory.cs.

6.12.2 Member Function Documentation

6.12.2.1 CreateService()

static IDeviceCommunicationService ladder_diagram_app.Services.CommunicationServices.CommunicationService↩
Factory.CreateService (
string connectionType) [inline], [static]

Creates a communication service instance for the specified connection type.

Parameters

connectionType	The type of connection (”MQTT” or ”BLE”).

Returns

An instance of IDeviceCommunicationService.

Exceptions

ArgumentException	Thrown when an unsupported connection type is provided.

Definition at line 17 of file CommunicationServiceFactory.cs.

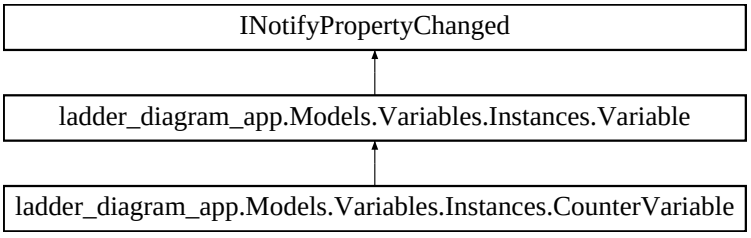
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CommunicationServices/CommunicationServiceFactory.cs

6.13 ladder_diagram_app.Models.Variables.Instances.CounterVariable Class Reference

Represents a counter variable in a ladder diagram, encapsulating preset and current values, count direction, and output states.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.CounterVariable:



Public Member Functions

- CounterVariable ()

  Initializes a new instance of the CounterVariable class with default values.
- override Dictionary< string, object > ToExportDictionary ()

  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()

  Converts the variable's properties to a dictionary for export purposes.

Properties

- double PV [get, set]

  Gets or sets the preset value (PV) of the counter, notifying subscribers on change.
- double CV [get, set]

  Gets or sets the current value (CV) of the counter, notifying subscribers on change.
- bool CU [get, set]

  Gets or sets the count up (CU) state, resetting count down (CD) if set to true, and notifying subscribers on change.
- bool CD [get, set]

  Gets or sets the count down (CD) state, resetting count up (CU) if set to true, and notifying subscribers on change.
- bool QU [get, set]

  Gets or sets the up output (QU) state, notifying subscribers on change.
- bool QD [get, set]

  Gets or sets the down output (QD) state, notifying subscribers on change.
- string Value [get, set]

  Gets or sets the display value of the counter, notifying subscribers on change.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]

  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]

  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]

  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- double _pv
  Stores the preset value of the counter.
- double _cv
  Stores the current value of the counter.
- bool _cu
  Stores the count up state.
- bool _cd
  Stores the count down state.
- bool _qu
  Stores the up output state.
- bool _qd
  Stores the down output state.
- string _value
  Stores the display value of the counter.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.13.1 Detailed Description

Represents a counter variable in a ladder diagram, encapsulating preset and current values, count direction, and output states.

Definition at line 6 of file CounterVariable.cs.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 CounterVariable()

ladder_diagram_app.Models.Variables.Instances.CounterVariable.CounterVariable () [inline]

Initializes a new instance of the CounterVariable class with default values.

Definition at line 117 of file CounterVariable.cs.

6.13.3 Member Function Documentation

6.13.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.CounterVariable.ToExport↩
Dictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 128 of file CounterVariable.cs.

6.13.4 Member Data Documentation

6.13.4.1 _cd

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable._cd [private]

Stores the count down state.

Definition at line 26 of file CounterVariable.cs.

6.13.4.2 _cu

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable._cu [private]

Stores the count up state.

Definition at line 21 of file CounterVariable.cs.

6.13.4.3 _cv

double ladder_diagram_app.Models.Variables.Instances.CounterVariable._cv [private]

Stores the current value of the counter.

Definition at line 16 of file CounterVariable.cs.

6.13.4.4 _pv

double ladder_diagram_app.Models.Variables.Instances.CounterVariable._pv [private]

Stores the preset value of the counter.

Definition at line 11 of file CounterVariable.cs.

6.13.4.5 _qd

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable._qd [private]

Stores the down output state.

Definition at line 36 of file CounterVariable.cs.

6.13.4.6 _qu

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable._qu [private]

Stores the up output state.

Definition at line 31 of file CounterVariable.cs.

6.13.4.7 _value

string ladder_diagram_app.Models.Variables.Instances.CounterVariable._value [private]

Stores the display value of the counter.

Definition at line 41 of file CounterVariable.cs.

6.13.5 Property Documentation

6.13.5.1 CD

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable.CD [get], [set]

Gets or sets the count down (CD) state, resetting count up (CU) if set to true, and notifying subscribers on change.

Definition at line 77 of file CounterVariable.cs.

6.13.5.2 CU

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable.CU [get], [set]

Gets or sets the count up (CU) state, resetting count down (CD) if set to true, and notifying subscribers on change.

Definition at line 64 of file CounterVariable.cs.

6.13.5.3 CV

double ladder_diagram_app.Models.Variables.Instances.CounterVariable.CV [get], [set]

Gets or sets the current value (CV) of the counter, notifying subscribers on change.

Definition at line 55 of file CounterVariable.cs.

6.13.5.4 PV

double ladder_diagram_app.Models.Variables.Instances.CounterVariable.PV [get], [set]

Gets or sets the preset value (PV) of the counter, notifying subscribers on change.

Definition at line 46 of file CounterVariable.cs.

6.13.5.5 QD

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable.QD [get], [set]

Gets or sets the down output (QD) state, notifying subscribers on change.

Definition at line 99 of file CounterVariable.cs.

6.13.5.6 QU

bool ladder_diagram_app.Models.Variables.Instances.CounterVariable.QU [get], [set]

Gets or sets the up output (QU) state, notifying subscribers on change.

Definition at line 90 of file CounterVariable.cs.

6.13.5.7 Value

string ladder_diagram_app.Models.Variables.Instances.CounterVariable.Value [get], [set]

Gets or sets the display value of the counter, notifying subscribers on change.

Definition at line 108 of file CounterVariable.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/CounterVariable.cs

6.14 ladder_diagram_app.Models.DeviceElement.Device Class Reference

Represents a device in a ladder diagram application, encapsulating its properties and configuration.

Public Member Functions

- Device ()

  Initializes a new instance of the Device class with default empty values.
- bool IsDeviceLoaded ()

  Checks if the device is loaded by verifying if the device name is set.
- void AddParentDevices (Window owner)

  Opens a dialog window to add parent devices to the current device.
- void UpdateFrom (Device device)

  Updates the current device's properties from another device instance.
- bool Validate ()

  Validates the device's properties to ensure they meet basic requirements.
- string DeviceInfo ()

  Generates a string representation of the device's properties.

Properties

- string device_name [get, set]

  Gets or sets the name of the device.
- double logic_voltage [get, set]

  Gets or sets the logic voltage of the device.
- List< int > digital_inputs [get, set]

  Gets or sets the list of digital input pins.
- List< string > digital_inputs_names [get, set]

  Gets or sets the names of the digital input pins.
- List< int > digital_outputs [get, set]

  Gets or sets the list of digital output pins.
- List< string > digital_outputs_names [get, set]

  Gets or sets the names of the digital output pins.
- List< int > analog_inputs [get, set]

  Gets or sets the list of analog input pins.
- List< string > analog_inputs_names [get, set]

  Gets or sets the names of the analog input pins.
- List< int > dac_outputs [get, set]

  Gets or sets the list of DAC output pins.
- List< string > dac_outputs_names [get, set]

  Gets or sets the names of the DAC output pins.
- List< int > one_wire_inputs [get, set]

  Gets or sets the list of one-wire input pins.
- List< List< string > > one_wire_inputs_names [get, set]

  Gets or sets the names of the one-wire input devices.
- List< List< string > > one_wire_inputs_devices_types [get, set]

  Gets or sets the types of one-wire input devices.
- List< List< string > > one_wire_inputs_devices_addresses [get, set]

  Gets or sets the addresses of one-wire input devices.
- int pwm_channels [get, set]

  Gets or sets the number of PWM channels.
- int max_hardware_timers [get, set]

  Gets or sets the maximum number of hardware timers.
- bool has_rtos [get, set]

Gets or sets a value indicating whether the device supports RTOS.

- List< int > UART [get, set]

  Gets or sets the list of UART channels.
- List< int > I2C [get, set]

  Gets or sets the list of I2C channels.
- List< int > SPI [get, set]

  Gets or sets the list of SPI channels.
- bool USB [get, set]

  Gets or sets a value indicating whether the device supports USB.
- List< string > parent_devices [get, set]

  Gets or sets the list of parent device names.

Static Private Member Functions

- static string FormatListOfLists (List< List< string > > listOfLists)

  Formats a list of lists into a string representation for display.

6.14.1 Detailed Description

Represents a device in a ladder diagram application, encapsulating its properties and configuration.

Definition at line 9 of file Device.cs.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Device()

ladder_diagram_app.Models.DeviceElement.Device.Device () [inline]

Initializes a new instance of the Device class with default empty values.

Definition at line 124 of file Device.cs.

6.14.3 Member Function Documentation

6.14.3.1 AddParentDevices()

void ladder_diagram_app.Models.DeviceElement.Device.AddParentDevices (
Window owner) [inline]

Opens a dialog window to add parent devices to the current device.

Parameters

owner	The owner window for the dialog.

Definition at line 163 of file Device.cs.

6.14.3.2 DeviceInfo()

string ladder_diagram_app.Models.DeviceElement.Device.DeviceInfo () [inline]

Generates a string representation of the device's properties.

Returns

A formatted string containing device information, or a message if no device is loaded.

Definition at line 223 of file Device.cs.

6.14.3.3 FormatListOfLists()

static string ladder_diagram_app.Models.DeviceElement.Device.FormatListOfLists (
List< List< string > > listOfLists) [inline], [static], [private]

Formats a list of lists into a string representation for display.

Parameters

listOfLists	The list of lists to format.

Returns

A formatted string with semicolon-separated inner lists.

Definition at line 259 of file Device.cs.

6.14.3.4 IsDeviceLoaded()

bool ladder_diagram_app.Models.DeviceElement.Device.IsDeviceLoaded () [inline]

Checks if the device is loaded by verifying if the device name is set.

Returns

True if the device name is not null or empty, false otherwise.

Definition at line 154 of file Device.cs.

6.14.3.5 UpdateFrom()

void ladder_diagram_app.Models.DeviceElement.Device.UpdateFrom (
Device device) [inline]

Updates the current device's properties from another device instance.

Parameters

device	The source device to copy properties from.

Definition at line 173 of file Device.cs.

6.14.3.6 Validate()

bool ladder_diagram_app.Models.DeviceElement.Device.Validate () [inline]

Validates the device's properties to ensure they meet basic requirements.

Returns

True if the device is valid, false otherwise.

Definition at line 212 of file Device.cs.

6.14.4 Property Documentation

6.14.4.1 analog_inputs

List<int> ladder_diagram_app.Models.DeviceElement.Device.analog_inputs [get], [set]

Gets or sets the list of analog input pins.

Definition at line 44 of file Device.cs.

6.14.4.2 analog_inputs_names

List<string> ladder_diagram_app.Models.DeviceElement.Device.analog_inputs_names [get], [set]

Gets or sets the names of the analog input pins.

Definition at line 49 of file Device.cs.

6.14.4.3 dac_outputs

List<int> ladder_diagram_app.Models.DeviceElement.Device.dac_outputs [get], [set]

Gets or sets the list of DAC output pins.

Definition at line 54 of file Device.cs.

6.14.4.4 dac_outputs_names

List<string> ladder_diagram_app.Models.DeviceElement.Device.dac_outputs_names [get], [set]

Gets or sets the names of the DAC output pins.

Definition at line 59 of file Device.cs.

6.14.4.5 device_name

string ladder_diagram_app.Models.DeviceElement.Device.device_name [get], [set]

Gets or sets the name of the device.

Definition at line 14 of file Device.cs.

6.14.4.6 digital_inputs

List<int> ladder_diagram_app.Models.DeviceElement.Device.digital_inputs [get], [set]

Gets or sets the list of digital input pins.

Definition at line 24 of file Device.cs.

6.14.4.7 digital_inputs_names

List<string> ladder_diagram_app.Models.DeviceElement.Device.digital_inputs_names [get], [set]

Gets or sets the names of the digital input pins.

Definition at line 29 of file Device.cs.

6.14.4.8 digital_outputs

List<int> ladder_diagram_app.Models.DeviceElement.Device.digital_outputs [get], [set]

Gets or sets the list of digital output pins.

Definition at line 34 of file Device.cs.

6.14.4.9 digital_outputs_names

List<string> ladder_diagram_app.Models.DeviceElement.Device.digital_outputs_names [get], [set]

Gets or sets the names of the digital output pins.

Definition at line 39 of file Device.cs.

6.14.4.10 has_rtos

bool ladder_diagram_app.Models.DeviceElement.Device.has_rtos [get], [set]

Gets or sets a value indicating whether the device supports RTOS.

Definition at line 94 of file Device.cs.

6.14.4.11 I2C

List<int> ladder_diagram_app.Models.DeviceElement.Device.I2C [get], [set]

Gets or sets the list of I2C channels.

Definition at line 104 of file Device.cs.

6.14.4.12 logic_voltage

double ladder_diagram_app.Models.DeviceElement.Device.logic_voltage [get], [set]

Gets or sets the logic voltage of the device.

Definition at line 19 of file Device.cs.

6.14.4.13 max_hardware_timers

int ladder_diagram_app.Models.DeviceElement.Device.max_hardware_timers [get], [set]

Gets or sets the maximum number of hardware timers.

Definition at line 89 of file Device.cs.

6.14.4.14 one_wire_inputs

List<int> ladder_diagram_app.Models.DeviceElement.Device.one_wire_inputs [get], [set]

Gets or sets the list of one-wire input pins.

Definition at line 64 of file Device.cs.

6.14.4.15 one_wire_inputs_devices_addresses

List<List<string> > ladder_diagram_app.Models.DeviceElement.Device.one_wire_inputs_devices_addresses [get], [set]

Gets or sets the addresses of one-wire input devices.

Definition at line 79 of file Device.cs.

6.14.4.16 one_wire_inputs_devices_types

List<List<string> > ladder_diagram_app.Models.DeviceElement.Device.one_wire_inputs_devices_types [get], [set]

Gets or sets the types of one-wire input devices.

Definition at line 74 of file Device.cs.

6.14.4.17 one_wire_inputs_names

List<List<string> > ladder_diagram_app.Models.DeviceElement.Device.one_wire_inputs_names [get], [set]

Gets or sets the names of the one-wire input devices.

Definition at line 69 of file Device.cs.

6.14.4.18 parent_devices

List<string> ladder_diagram_app.Models.DeviceElement.Device.parent_devices [get], [set]

Gets or sets the list of parent device names.

Definition at line 119 of file Device.cs.

6.14.4.19 pwm_channels

int ladder_diagram_app.Models.DeviceElement.Device.pwm_channels [get], [set]

Gets or sets the number of PWM channels.

Definition at line 84 of file Device.cs.

6.14.4.20 SPI

List<int> ladder_diagram_app.Models.DeviceElement.Device.SPI [get], [set]

Gets or sets the list of SPI channels.

Definition at line 109 of file Device.cs.

6.14.4.21 UART

List<int> ladder_diagram_app.Models.DeviceElement.Device.UART [get], [set]

Gets or sets the list of UART channels.

Definition at line 99 of file Device.cs.

6.14.4.22 USB

bool ladder_diagram_app.Models.DeviceElement.Device.USB [get], [set]

Gets or sets a value indicating whether the device supports USB.

Definition at line 114 of file Device.cs.

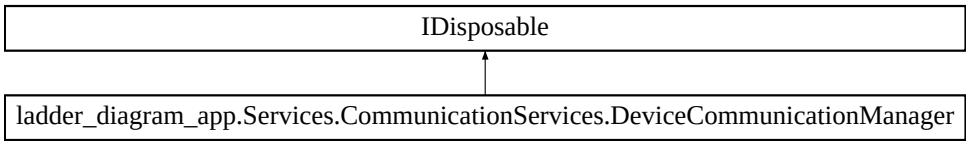
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/DeviceElement/Device.cs

6.15 ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager Class Reference

Manages device communication, handling connection, disconnection, and configuration exchange for MQTT and BLE protocols.

Inheritance diagram for ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager:



Public Member Functions

- DeviceCommunicationManager (Action< string > onConfigurationReceived, Action< bool > onConnectionStatusChanged, Action< string > onMonitorDataReceived, Action< string > onOneWireDataReceived)

  Initializes a new instance of the DeviceCommunicationManager class.
- async Task ConnectAsync (Window owner, string connectionType)

  Connects to a device asynchronously using the specified connection type.
- async Task DisconnectAsync (Window owner)

  Disconnects from the device asynchronously.
- async Task SendConfigurationAsync (string jsonConfig, Window owner)

  Sends a JSON configuration to the connected device.
- void Dispose ()

  Disposes of the communication service and BLE device watcher, releasing resources.

Properties

- IDeviceCommunicationService? _communicationService [get, set]

Private Member Functions

- string GetDeviceId (Window owner, string connectionType)

  Retrieves the device ID based on the connection type (MQTT or BLE).

Private Attributes

- readonly BleDeviceWatcher _bleDeviceWatcher
- readonly Action< string > _onConfigurationReceived
- readonly Action< bool > _onConnectionStatusChanged
- readonly Action< string > _onMonitorDataReceived
- readonly Action< string > _onOneWireDataReceived

6.15.1 Detailed Description

Manages device communication, handling connection, disconnection, and configuration exchange for MQTT and BLE protocols.

Definition at line 11 of file DeviceCommunicationManager.cs.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 DeviceCommunicationManager()

ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager.DeviceCommunicationManager (
Action< string > onConfigurationReceived,
Action< bool > onConnectionStatusChanged,
Action< string > onMonitorDataReceived,
Action< string > onOneWireDataReceived) [inline]

Initializes a new instance of the DeviceCommunicationManager class.

Parameters

onConfigurationReceived	Callback for handling received configuration data.
onConnectionStatusChanged	Callback for handling connection status changes.
onMonitorDataReceived	Callback for handling received monitor data.
onOneWireDataReceived	Callback for handling received one-wire data.

Exceptions

ArgumentNullException	Thrown if any callback is null.

Definition at line 28 of file DeviceCommunicationManager.cs.

6.15.3 Member Function Documentation

6.15.3.1 ConnectAsync()

async Task ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager.ConnectAsync (
Window owner,
string connectionType) [inline]

Connects to a device asynchronously using the specified connection type.

Parameters

owner	The owner window for displaying dialogs.
connectionType	The type of connection (”MQTT” or ”BLE”).

Definition at line 77 of file DeviceCommunicationManager.cs.

6.15.3.2 DisconnectAsync()

async Task ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager.DisconnectAsync (
Window owner) [inline]

Disconnects from the device asynchronously.

Parameters

owner	The owner window for displaying dialogs.

Definition at line 119 of file DeviceCommunicationManager.cs.

6.15.3.3 Dispose()

void ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager.Dispose () [inline]

Disposes of the communication service and BLE device watcher, releasing resources.

Definition at line 151 of file DeviceCommunicationManager.cs.

6.15.3.4 GetDeviceId()

string ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager.GetDeviceId (
Window owner,
string connectionType) [inline], [private]

Retrieves the device ID based on the connection type (MQTT or BLE).

Parameters

owner	The owner window for displaying dialogs.
connectionType	The type of connection (”MQTT” or ”BLE”).

Returns

The device ID, or an empty string if no device is selected.

Definition at line 47 of file DeviceCommunicationManager.cs.

6.15.3.5 SendConfigurationAsync()

async Task ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager.SendConfiguration↩
Async (
string jsonConfig,
Window owner) [inline]

Sends a JSON configuration to the connected device.

Parameters

jsonConfig	The JSON configuration string to send.
owner	The owner window for displaying dialogs.

Definition at line 136 of file DeviceCommunicationManager.cs.

6.15.4 Member Data Documentation

6.15.4.1 _bleDeviceWatcher

readonly BleDeviceWatcher ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager._bleDeviceWatcher [private]

Definition at line 14 of file DeviceCommunicationManager.cs.

6.15.4.2 _onConfigurationReceived

readonly Action<string> ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager._onConfigurationReceived [private]

Definition at line 15 of file DeviceCommunicationManager.cs.

6.15.4.3 _onConnectionStatusChanged

readonly Action<bool> ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager._onConnectionStatusChanged [private]

Definition at line 16 of file DeviceCommunicationManager.cs.

6.15.4.4 _onMonitorDataReceived

readonly Action<string> ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager._onMonitorDataReceived [private]

Definition at line 17 of file DeviceCommunicationManager.cs.

6.15.4.5 _onOneWireDataReceived

readonly Action<string> ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager._onOneWireDataReceived [private]

Definition at line 18 of file DeviceCommunicationManager.cs.

6.15.5 Property Documentation

6.15.5.1 _communicationService

IDeviceCommunicationService? ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager.↩
_communicationService [get], [set]

Definition at line 13 of file DeviceCommunicationManager.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CommunicationServices/DeviceCommunicationManager.cs

6.16 ladder_diagram_app.Models.DeviceElement.DevicePinManager Class Reference

Manages pin mapping options for a device, providing collections of available digital, analog, and one-wire pin names.

Public Member Functions

- DevicePinManager ()
  Initializes a new instance of the DevicePinManager class with empty pin option collections.
- void UpdateDevicePinOptions (Device device)
  Updates the pin option collections based on the provided device's pin names.

Properties

- ObservableCollection< string > DigitalInputOptions [get]
  Gets the collection of digital input pin names available for mapping.
- ObservableCollection< string > DigitalOutputOptions [get]
  Gets the collection of digital output pin names available for mapping.
- ObservableCollection< string > AnalogInputOptions [get]
  Gets the collection of analog input pin names available for mapping.
- ObservableCollection< string > AnalogOutputOptions [get]
  Gets the collection of analog output (DAC) pin names available for mapping.
- ObservableCollection< string > OneWireInputOptions [get]
  Gets the collection of one-wire input pin names available for mapping.

6.16.1 Detailed Description

Manages pin mapping options for a device, providing collections of available digital, analog, and one-wire pin names.

Definition at line 8 of file DevicePinManager.cs.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 DevicePinManager()

ladder_diagram_app.Models.DeviceElement.DevicePinManager.DevicePinManager () [inline]

Initializes a new instance of the DevicePinManager class with empty pin option collections.

Definition at line 38 of file DevicePinManager.cs.

6.16.3 Member Function Documentation

6.16.3.1 UpdateDevicePinOptions()

void ladder_diagram_app.Models.DeviceElement.DevicePinManager.UpdateDevicePinOptions (
Device device) [inline]

Updates the pin option collections based on the provided device's pin names.

Parameters

device	The device whose pin names are used to update the options.

Definition at line 51 of file DevicePinManager.cs.

6.16.4 Property Documentation

6.16.4.1 AnalogInputOptions

ObservableCollection<string> ladder_diagram_app.Models.DeviceElement.DevicePinManager.AnalogInputOptions [get]

Gets the collection of analog input pin names available for mapping.

Definition at line 23 of file DevicePinManager.cs.

6.16.4.2 AnalogOutputOptions

ObservableCollection<string> ladder_diagram_app.Models.DeviceElement.DevicePinManager.AnalogOutputOptions [get]

Gets the collection of analog output (DAC) pin names available for mapping.

Definition at line 28 of file DevicePinManager.cs.

6.16.4.3 DigitalInputOptions

ObservableCollection<string> ladder_diagram_app.Models.DeviceElement.DevicePinManager.DigitalInputOptions [get]

Gets the collection of digital input pin names available for mapping.

Definition at line 13 of file DevicePinManager.cs.

6.16.4.4 DigitalOutputOptions

ObservableCollection<string> ladder_diagram_app.Models.DeviceElement.DevicePinManager.DigitalOutputOptions [get]

Gets the collection of digital output pin names available for mapping.

Definition at line 18 of file DevicePinManager.cs.

6.16.4.5 OneWireInputOptions

ObservableCollection<string> ladder_diagram_app.Models.DeviceElement.DevicePinManager.OneWireInputOptions [get]

Gets the collection of one-wire input pin names available for mapping.

Definition at line 33 of file DevicePinManager.cs.

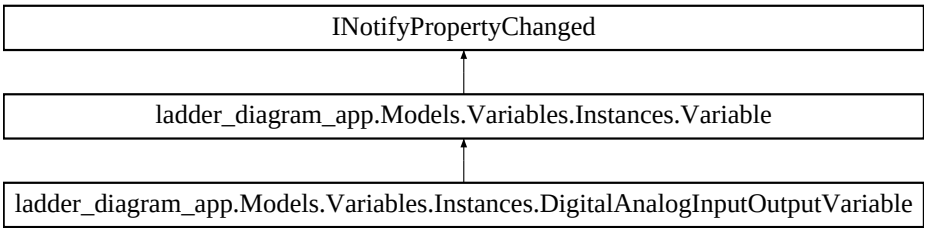
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/DeviceElement/DevicePinManager.cs

6.17 ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutputVariable Class Reference

Represents a digital or analog input/output variable in a ladder diagram, associated with a specific pin.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutputVariable:



Public Member Functions

- DigitalAnalogInputOutputVariable ()
  Initializes a new instance of the DigitalAnalogInputOutputVariable class with an empty pin name.
- override Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Properties

- string PinName [get, set]
  Gets or sets the name of the pin associated with the variable, notifying subscribers on change.
- bool IsValid [get]
  Gets a value indicating whether the variable is valid based on the presence of a pin name.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]
  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]
  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]
  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- string _pinName
  Stores the name of the pin associated with the variable.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.17.1 Detailed Description

Represents a digital or analog input/output variable in a ladder diagram, associated with a specific pin.

Definition at line 6 of file DigitalAnalogInputOutputVariable.cs.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 DigitalAnalogInputOutputVariable()

ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutputVariable.DigitalAnalogInputOutputVariable () [inline]

Initializes a new instance of the DigitalAnalogInputOutputVariable class with an empty pin name.

Definition at line 16 of file DigitalAnalogInputOutputVariable.cs.

6.17.3 Member Function Documentation

6.17.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutput↩ Variable.ToExportDictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 39 of file DigitalAnalogInputOutputVariable.cs.

6.17.4 Member Data Documentation

6.17.4.1 _pinName

string ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutputVariable._pinName [private]

Stores the name of the pin associated with the variable.

Definition at line 11 of file DigitalAnalogInputOutputVariable.cs.

6.17.5 Property Documentation

6.17.5.1 IsValid

bool ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutputVariable.IsValid [get]

Gets a value indicating whether the variable is valid based on the presence of a pin name.

Definition at line 33 of file DigitalAnalogInputOutputVariable.cs.

6.17.5.2 PinName

string ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutputVariable.PinName [get], [set]

Gets or sets the name of the pin associated with the variable, notifying subscribers on change.

Definition at line 24 of file DigitalAnalogInputOutputVariable.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/DigitalAnalogInputOutputVariable.cs

6.18 ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportData Class Reference

Properties

- Device? Device [get, set]
- List< Dictionary< string, object > >? Variables [get, set]
- List< ExportWire >? Wires [get, set]

6.18.1 Detailed Description

Definition at line 321 of file ImportExportService.cs.

6.18.2 Property Documentation

6.18.2.1 Device

Device? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportData.Device [get], [set]

Definition at line 323 of file ImportExportService.cs.

6.18.2.2 Variables

List<Dictionary<string, object> >? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportData.Variables [get], [set]

Definition at line 324 of file ImportExportService.cs.

6.18.2.3 Wires

List<ExportWire>? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportData.Wires [get], [set]

Definition at line 325 of file ImportExportService.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/ImportExportServices/ImportExportService.cs

6.19 ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNode Class Reference

Properties

- string? Type [get, set]
- string? ElementType [get, set]
- List< string >? ComboBoxValues [get, set]
- List< ExportNode >? Nodes1 [get, set]
- List< ExportNode >? Nodes2 [get, set]

6.19.1 Detailed Description

Definition at line 333 of file ImportExportService.cs.

6.19.2 Property Documentation

6.19.2.1 ComboBoxValues

List<string>? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNode.ComboBoxValues [get], [set]

Definition at line 337 of file ImportExportService.cs.

6.19.2.2 ElementType

string? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNode.ElementType [get], [set]

Definition at line 336 of file ImportExportService.cs.

6.19.2.3 Nodes1

List<ExportNode>? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNode.Nodes1 [get], [set]

Definition at line 338 of file ImportExportService.cs.

6.19.2.4 Nodes2

List<ExportNode>? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNode.Nodes2 [get], [set]

Definition at line 339 of file ImportExportService.cs.

6.19.2.5 Type

string? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNode.Type [get], [set]

Definition at line 335 of file ImportExportService.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/ImportExportServices/ImportExportService.cs

6.20 ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportWire Class Reference

Properties

- List< ExportNode >? Nodes [get, set]

6.20.1 Detailed Description

Definition at line 328 of file ImportExportService.cs.

6.20.2 Property Documentation

6.20.2.1 Nodes

List<ExportNode>? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportWire.Nodes [get], [set]

Definition at line 330 of file ImportExportService.cs.

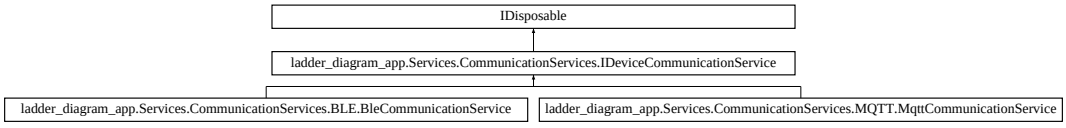
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/ImportExportServices/ImportExportService.cs

6.21 ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService Interface Reference

Defines the contract for device communication services, supporting connection management and data exchange.

Inheritance diagram for ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService:



Public Member Functions

- Task< bool > ConnectAsync (string deviceIdentifier)

  Connects to a device asynchronously using the specified identifier.
- Task DisconnectAsync ()

  Disconnects from the device asynchronously.
- Task< bool > SendConfigurationAsync (string configJson)

  Sends a JSON configuration to the device asynchronously.
- Task RequestConfigurationAsync ()

  Requests the configuration from the device asynchronously.

Properties

- bool IsConnected [get]

  Gets a value indicating whether the service is connected to a device.
- string ConnectionType [get]

  Gets the type of connection (e.g., "MQTT" or "BLE").

Events

- EventHandler< string > ConfigurationReceived

  Occurs when configuration data is received from the device.
- EventHandler< string > MonitorDataReceived

  Occurs when monitor data is received from the device.
- EventHandler< string > OneWireDataReceived

  Occurs when one-wire data is received from the device.
- EventHandler< bool > ConnectionStatusChanged

  Occurs when the connection status changes.

6.21.1 Detailed Description

Defines the contract for device communication services, supporting connection management and data exchange.

Definition at line 6 of file IDeviceCommunicationService.cs.

6.21.2 Member Function Documentation

6.21.2.1 ConnectAsync()

Task< bool > ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.ConnectAsync (string deviceIdentifier)

Connects to a device asynchronously using the specified identifier.

Parameters

deviceIdentifier	The identifier of the device to connect to.

Returns

True if connection is successful, otherwise false.

Implemented in ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService, and ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.

6.21.2.2 DisconnectAsync()

Task ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.DisconnectAsync ()

Disconnects from the device asynchronously.

Implemented in ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService, and ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.

6.21.2.3 RequestConfigurationAsync()

Task ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.RequestConfigurationAsync ()

Requests the configuration from the device asynchronously.

Implemented in ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService, and ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.

6.21.2.4 SendConfigurationAsync()

Task< bool > ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.SendConfiguration↩
Async (
string configJson)

Sends a JSON configuration to the device asynchronously.

Parameters

configJson	The JSON configuration string to send.

Returns

True if sending is successful, otherwise false.

Implemented in ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService, and ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.

6.21.3 Property Documentation

6.21.3.1 ConnectionType

string ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.ConnectionType [get]

Gets the type of connection (e.g., ”MQTT” or ”BLE”).

Implemented in ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService, and ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.

Definition at line 36 of file IDeviceCommunicationService.cs.

6.21.3.2 IsConnected

bool ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.IsConnected [get]

Gets a value indicating whether the service is connected to a device.

Implemented in ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService, and ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.

Definition at line 31 of file IDeviceCommunicationService.cs.

6.21.4 Event Documentation

6.21.4.1 ConfigurationReceived

EventHandler<string> ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.Configuration↩Received

Occurs when configuration data is received from the device.

Definition at line 11 of file IDeviceCommunicationService.cs.

6.21.4.2 ConnectionStatusChanged

EventHandler<bool> ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.Connection↩StatusChanged

Occurs when the connection status changes.

Definition at line 26 of file IDeviceCommunicationService.cs.

6.21.4.3 MonitorDataReceived

EventHandler<string> ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.Monitor↩DataReceived

Occurs when monitor data is received from the device.

Definition at line 16 of file IDeviceCommunicationService.cs.

6.21.4.4 OneWireDataReceived

EventHandler<string> ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.OneWire↩DataReceived

Occurs when one-wire data is received from the device.

Definition at line 21 of file IDeviceCommunicationService.cs.

The documentation for this interface was generated from the following file:

- ladder_diagram_app/Services/CommunicationServices/IDeviceCommunicationService.cs

6.22 ladder_diagram_app.Services.ImportExportServices.ImportExportService Class Reference

Handles importing and exporting of ladder diagram configurations to and from JSON format.

Classes

- class ExportData
- class ExportNode
- class ExportWire

Public Member Functions

- ImportExportService (VariablesManager variablesManager, Device device, WiresManager wiresManager, CanvasManager canvasManager, DevicePinManager devicePinManager)

  Initializes a new instance of the ImportExportService class.
- string? ExportToJson (MainWindow owner)

  Exports the current configuration to a JSON string.
- void ImportFromJson (string jsonString, MainWindow owner)

  Imports a configuration from a JSON string and updates the application state.

Private Member Functions

- ExportData? PrepareExportData (MainWindow owner)

  Prepares the data for export, validating variables and collecting device, variables, and wire information.
- List< ExportNode > ExportNodes (List< Node > nodes)

  Converts a list of nodes to a list of exportable node data.
- List< Node > ImportNodes (List< ExportNode > exportNodes, Wire? parentWire=null)

  Converts a list of exportable node data back into a list of nodes.

Private Attributes

- readonly VariablesManager _variablesManager
- readonly Device _device
- readonly WiresManager _wiresManager
- readonly CanvasManager _canvasManager
- readonly DevicePinManager _devicePinManager
- bool _abortExport

6.22.1 Detailed Description

Handles importing and exporting of ladder diagram configurations to and from JSON format.

Definition at line 15 of file ImportExportService.cs.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 ImportExportService()

```
ladder_diagram_app.Services.ImportExportServices.ImportExportService.ImportExportService (
        VariablesManager variablesManager,
        Device device,
        WiresManager wiresManager,
        CanvasManager canvasManager,
        DevicePinManager devicePinManager)   [inline]
```

Initializes a new instance of the ImportExportService class.

Parameters

variablesManager	Manages variables for ladder elements.
device	The device configuration.
wiresManager	Manages wires on the canvas.
canvasManager	Manages canvas rendering.
devicePinManager	Manages device pin configurations.

Definition at line 32 of file ImportExportService.cs.

6.22.3 Member Function Documentation

6.22.3.1 ExportNodes()

List< ExportNode > ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNodes (
List< Node > nodes) [inline], [private]

Converts a list of nodes to a list of exportable node data.

Parameters

nodes	The nodes to export.

Returns

A list of exportable node data.

Definition at line 233 of file ImportExportService.cs.

6.22.3.2 ExportToJson()

string? ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportToJson (
MainWindow owner) [inline]

Exports the current configuration to a JSON string.

Parameters

owner	The main window for displaying notifications.

Returns

The JSON string representing the configuration, or null if export fails.

Definition at line 51 of file ImportExportService.cs.

6.22.3.3 ImportFromJson()

void ladder_diagram_app.Services.ImportExportServices.ImportExportService.ImportFromJson (
string jsonString,
MainWindow owner) [inline]

Imports a configuration from a JSON string and updates the application state.

Parameters

jsonString	The JSON string to import.
owner	The main window for displaying notifications.

Definition at line 72 of file ImportExportService.cs.

6.22.3.4 ImportNodes()

List< Node > ladder_diagram_app.Services.ImportExportServices.ImportExportService.ImportNodes (
List< ExportNode > exportNodes,
Wire? parentWire = null) [inline], [private]

Converts a list of exportable node data back into a list of nodes.

Parameters

exportNodes	The exportable node data.
parentWire	The parent wire, if applicable.

Returns

A list of nodes.

Definition at line 270 of file ImportExportService.cs.

6.22.3.5 PrepareExportData()

ExportData? ladder_diagram_app.Services.ImportExportServices.ImportExportService.PrepareExportData (
MainWindow owner) [inline], [private]

Prepares the data for export, validating variables and collecting device, variables, and wire information.

Parameters

owner	The main window for displaying notifications.

Returns

The prepared export data, or null if validation fails.

Definition at line 198 of file ImportExportService.cs.

6.22.4 Member Data Documentation

6.22.4.1 _abortExport

bool ladder_diagram_app.Services.ImportExportServices.ImportExportService._abortExport [private]

Definition at line 22 of file ImportExportService.cs.

6.22.4.2 _canvasManager

readonly CanvasManager ladder_diagram_app.Services.ImportExportServices.ImportExportService._canvasManager [private]

Definition at line 20 of file ImportExportService.cs.

6.22.4.3 _device

readonly Device ladder_diagram_app.Services.ImportExportServices.ImportExportService._device [private]

Definition at line 18 of file ImportExportService.cs.

6.22.4.4 _devicePinManager

readonly DevicePinManager ladder_diagram_app.Services.ImportExportServices.ImportExportService._devicePinManager [private]

Definition at line 21 of file ImportExportService.cs.

6.22.4.5 _variablesManager

readonly VariablesManager ladder_diagram_app.Services.ImportExportServices.ImportExportService._variablesManager [private]

Definition at line 17 of file ImportExportService.cs.

6.22.4.6 _wiresManager

readonly WiresManager ladder_diagram_app.Services.ImportExportServices.ImportExportService._wiresManager [private]

Definition at line 19 of file ImportExportService.cs.

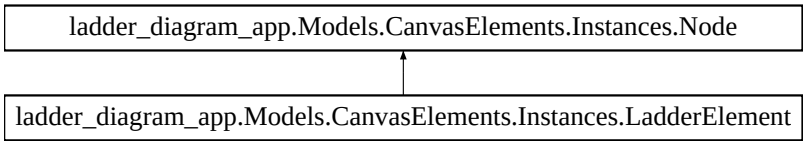
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/ImportExportServices/ImportExportService.cs

6.23 ladder_diagram_app.Models.CanvasElements.Instances.LadderElement Class Reference

Represents a ladder diagram element (e.g., contact, coil, timer) with an associated image and variable selection ComboBoxes.

Inheritance diagram for ladder_diagram_app.Models.CanvasElements.Instances.LadderElement:



Public Member Functions

- LadderElement (string type, ObservableCollection< string > variablesListContacts, ObservableCollection< string > variablesListCoils, ObservableCollection< string > variablesListMath, ObservableCollection< string > variablesListCompare, ObservableCollection< string > variablesListCounter, ObservableCollection< string > variablesListTimer, ObservableCollection< string > variablesListReset)

  Initializes a new instance of the LadderElement class with the specified type and variable lists.

Public Member Functions inherited from ladder_diagram_app.Models.CanvasElements.Instances.Node

- void HighlightNode ()

  Highlights the node by applying a red drop shadow effect to its image.
- void UnhighlightNode ()

  Removes the highlight effect from the node by clearing its image effect.

Properties

- string Type [get, set]

  Gets or sets the type of the ladder element (e.g., NOContact, Coil, AddMath).
- override double X [get, set]

  Gets or sets the X-coordinate of the ladder element.
- override double Y [get, set]

  Gets or sets the Y-coordinate of the ladder element.
- override double Width [get]

  Gets the width of the ladder element, based on the maximum of the image or ComboBox widths plus margins.
- IReadOnlyList< ComboBox > VariableComboBoxes [get]

  Gets a read-only list of the ComboBoxes used for variable selection.
- int ComboBoxCount [get]

  Gets the number of ComboBoxes associated with this ladder element.

Properties inherited from ladder_diagram_app.Models.CanvasElements.Instances.Node

- double X [get, set]

  Gets or sets the X-coordinate of the node.
- double Y [get, set]

  Gets or sets the Y-coordinate of the node.
- object? Parent [get, set]

  Gets or sets the parent object of the node, used to track hierarchical relationships.
- double Width [get]

  Gets the width of the node, typically based on its visual representation.
- virtual ? Image Image [get, set]

  Gets or sets the image representing the node visually, if applicable.

Private Attributes

- readonly List< ComboBox > _variableComboBoxes = []

  Stores the list of ComboBoxes for variable selection.

6.23.1 Detailed Description

Represents a ladder diagram element (e.g., contact, coil, timer) with an associated image and variable selection ComboBoxes.

Definition at line 10 of file LadderElement.cs.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 LadderElement()

ladder_diagram_app.Models.CanvasElements.Instances.LadderElement.LadderElement (
string type,
ObservableCollection< string > variablesListContacts,
ObservableCollection< string > variablesListCoils,
ObservableCollection< string > variablesListMath,
ObservableCollection< string > variablesListCompare,
ObservableCollection< string > variablesListCounter,
ObservableCollection< string > variablesListTimer,
ObservableCollection< string > variablesListReset) [inline]

Initializes a new instance of the LadderElement class with the specified type and variable lists.

Parameters

type	The type of the ladder element (e.g., NOContact, Coil).
variablesListContacts	List of variables for contact elements.
variablesListCoils	List of variables for coil elements.
variablesListMath	List of variables for math elements.
variablesListCompare	List of variables for compare elements.
variablesListCounter	List of variables for counter elements.
variablesListTimer	List of variables for timer elements.
variablesListReset	List of variables for reset elements.

Definition at line 49 of file LadderElement.cs.

6.23.3 Member Data Documentation

6.23.3.1 _variableComboBoxes

readonly List<ComboBox> ladder_diagram_app.Models.CanvasElements.Instances.LadderElement._variableCombo↩
Boxes = [] [private]

Stores the list of ComboBoxes for variable selection.

Definition at line 15 of file LadderElement.cs.

6.23.4 Property Documentation

6.23.4.1 ComboBoxCount

int ladder_diagram_app.Models.CanvasElements.Instances.LadderElement.ComboBoxCount [get]

Gets the number of ComboBoxes associated with this ladder element.

Definition at line 176 of file LadderElement.cs.

6.23.4.2 Type

string ladder_diagram_app.Models.CanvasElements.Instances.LadderElement.Type [get], [set]

Gets or sets the type of the ladder element (e.g., NOContact, Coil, AddMath).

Definition at line 20 of file LadderElement.cs.

6.23.4.3 VariableComboBoxes

IReadOnlyList<ComboBox> ladder_diagram_app.Models.CanvasElements.Instances.LadderElement.VariableCombo↩
Boxes [get]

Gets a read-only list of the ComboBoxes used for variable selection.

Definition at line 171 of file LadderElement.cs.

6.23.4.4 Width

override double ladder_diagram_app.Models.CanvasElements.Instances.LadderElement.Width [get]

Gets the width of the ladder element, based on the maximum of the image or ComboBox widths plus margins.

Definition at line 35 of file LadderElement.cs.

6.23.4.5 X

override double ladder_diagram_app.Models.CanvasElements.Instances.LadderElement.X [get], [set]

Gets or sets the X-coordinate of the ladder element.

Definition at line 25 of file LadderElement.cs.

6.23.4.6 Y

override double ladder_diagram_app.Models.CanvasElements.Instances.LadderElement.Y [get], [set]

Gets or sets the Y-coordinate of the ladder element.

Definition at line 30 of file LadderElement.cs.

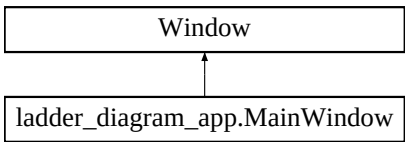
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/CanvasElements/Instances/LadderElement.cs

6.24 ladder_diagram_app.MainWindow Class Reference

Main application window for managing ladder diagrams, device communication, and variables.

Inheritance diagram for ladder_diagram_app.MainWindow:



Public Member Functions

- MainWindow ()
  Initializes a new instance of the MainWindow class.

Public Attributes

- readonly DevicePinManager _devicePinManager
  Manages device pin configurations.

Properties

- List< string > AdcSensorTypes = ["TM7711", "HX710B"] [get]
  List of supported ADC sensor types.
- List< string > AdcSensorSamplingRates = ["10Hz", "40Hz", "Temperature"] [get]
  List of supported ADC sensor sampling rates.
- DevicePinManager DevicePinManager [get]
  Gets the device pin manager.
- VariablesManager VariablesManager [get]
  Gets the variables manager.

Private Member Functions

- void MainWindow_PreviewKeyDown (object sender, KeyEventArgs e)

  Handles keyboard input for deleting elements or adding variables.
- void ButtonImport_Click (object sender, RoutedEventArgs e)

  Imports a project from a JSON file, replacing current content after confirmation.
- void ButtonExport_Click (object sender, RoutedEventArgs e)

  Exports the current project to a JSON file.
- void ButtonAddVariable_Click (object? sender, RoutedEventArgs? e)

  Adds a new variable based on input name and type.
- void ButtonDeleteVariable_Click (object sender, RoutedEventArgs e)

  Deletes a variable when the delete button is clicked.
- void ButtonChangeBoolean_Click (object sender, RoutedEventArgs e)

  Toggles the boolean value of a variable when its button is clicked.
- void TextBoxVariable_TextChanged (object sender, RoutedEventArgs e)

  Updates a variable's value when its text box changes, triggered by Enter or focus loss.
- void ComboBoxVariable_SelectionChanged (object sender, RoutedEventArgs e)

  Updates a variable's value when its combo box selection changes.
- void TextBoxVariable_PreviewTextInput (object sender, TextCompositionEventArgs e)

  Restricts text box input to valid numbers or a minus sign.
- void ButtonDeviceInfo_Click (object sender, RoutedEventArgs e)

  Displays device information in a notification window.
- void ButtonParentDevice_Click (object sender, RoutedEventArgs e)

  Opens a window to manage parent devices.
- void ButtonAddWire_Click (object sender, RoutedEventArgs e)

  Adds a new wire to the canvas and updates the display.
- void Button_PreviewMouseLeftButtonDown (object sender, MouseEventArgs e)

  Initiates drag-and-drop from the toolbar for adding elements.
- void Canvas_DragOver (object sender, DragEventArgs e)

  Highlights valid drop positions during drag operations on the canvas.
- void Canvas_Drop (object sender, DragEventArgs e)

  Handles dropping elements onto the canvas.
- void MainCanvas_MouseMove (object sender, MouseEventArgs e)

  Handles mouse movement for dragging elements on the canvas.
- void MainCanvas_MouseLeftButtonUp (object sender, MouseButtonEventArgs e)

  Handles mouse release for placing elements on the canvas.
- void MainCanvas_MouseLeftButtonDown (object sender, MouseButtonEventArgs e)

  Handles mouse click for selecting elements or starting a drag operation.
- void ButtonDelete_Click (object? sender, RoutedEventArgs? e)

  Deletes the selected canvas element or wire.
- async void ButtonConnect_Click (object sender, RoutedEventArgs e)

  Initiates connection to the device via MQTT or BLE.
- async void ButtonDisconnect_Click (object sender, RoutedEventArgs e)

  Disconnects from the device.
- async void ButtonSendToDevice_Click (object sender, RoutedEventArgs e)

  Sends the current configuration to the device.
- void OnConfigurationReceived (string jsonConfig)

  Handles received device configuration and updates the application state.
- void OnConnectionStatusChanged (bool isConnected)

  Updates UI based on device connection status changes.
- void MonitorExpander_Collapsed (object sender, RoutedEventArgs e)

Adjusts the grid row height when the monitor expander is collapsed.

- void ActionButton_Click (object sender, RoutedEventArgs e)

  Handles adding or deleting one-wire sensors.
- async void Window_Closing (object? sender, System.ComponentModel.CancelEventArgs e)

  Ensures proper cleanup when the application window is closing.

Private Attributes

- readonly Device _device
- readonly VariablesManager _variablesManager
- readonly WiresManager _wiresManager
- readonly CanvasManager _canvasManager
- readonly CanvasElementFinder _canvasElementFinder
- readonly CanvasInteractionManager _canvasInteractionManager
- readonly DeviceCommunicationManager _deviceCommunicationManager
- readonly MonitorDataService _monitorDataService
- readonly OneWireDataService _oneWireDataService
- readonly ImportExportService _importExportService

6.24.1 Detailed Description

Main application window for managing ladder diagrams, device communication, and variables.

Definition at line 26 of file MainWindow.xaml.cs.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 MainWindow()

ladder_diagram_app.MainWindow.MainWindow () [inline]

Initializes a new instance of the MainWindow class.

Definition at line 76 of file MainWindow.xaml.cs.

6.24.3 Member Function Documentation

6.24.3.1 ActionButton_Click()

void ladder_diagram_app.MainWindow.ActionButton_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Handles adding or deleting one-wire sensors.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 573 of file MainWindow.xaml.cs.

6.24.3.2 Button_PreviewMouseLeftButtonDown()

void ladder_diagram_app.MainWindow.Button_PreviewMouseLeftButtonDown (
object sender,
MouseEventArgs e) [inline], [private]

Initiates drag-and-drop from the toolbar for adding elements.

Parameters

sender	The sender object.
e	The mouse event arguments.

Definition at line 408 of file MainWindow.xaml.cs.

6.24.3.3 ButtonAddVariable_Click()

void ladder_diagram_app.MainWindow.ButtonAddVariable_Click (
object? sender,
RoutedEventArgs? e) [inline], [private]

Adds a new variable based on input name and type.

Parameters

sender	The sender object, can be null for keyboard-triggered calls.
e	The routed event arguments, can be null for keyboard-triggered calls.

Definition at line 263 of file MainWindow.xaml.cs.

6.24.3.4 ButtonAddWire_Click()

void ladder_diagram_app.MainWindow.ButtonAddWire_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Adds a new wire to the canvas and updates the display.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 396 of file MainWindow.xaml.cs.

6.24.3.5 ButtonChangeBoolean_Click()

void ladder_diagram_app.MainWindow.ButtonChangeBoolean_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Toggles the boolean value of a variable when its button is clicked.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 291 of file MainWindow.xaml.cs.

6.24.3.6 ButtonConnect_Click()

async void ladder_diagram_app.MainWindow.ButtonConnect_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Initiates connection to the device via MQTT or BLE.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 482 of file MainWindow.xaml.cs.

6.24.3.7 ButtonDelete_Click()

void ladder_diagram_app.MainWindow.ButtonDelete_Click (
object? sender,
RoutedEventArgs? e) [inline], [private]

Deletes the selected canvas element or wire.

Parameters

sender	The sender object, can be null for keyboard-triggered calls.
e	The routed event arguments, can be null for keyboard-triggered calls.

Definition at line 471 of file MainWindow.xaml.cs.

6.24.3.8 ButtonDeleteVariable_Click()

void ladder_diagram_app.MainWindow.ButtonDeleteVariable_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Deletes a variable when the delete button is clicked.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 278 of file MainWindow.xaml.cs.

6.24.3.9 ButtonDeviceInfo_Click()

void ladder_diagram_app.MainWindow.ButtonDeviceInfo_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Displays device information in a notification window.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 374 of file MainWindow.xaml.cs.

6.24.3.10 ButtonDisconnect_Click()

async void ladder_diagram_app.MainWindow.ButtonDisconnect_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Disconnects from the device.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 493 of file MainWindow.xaml.cs.

6.24.3.11 ButtonExport_Click()

void ladder_diagram_app.MainWindow.ButtonExport_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Exports the current project to a JSON file.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 228 of file MainWindow.xaml.cs.

6.24.3.12 ButtonImport_Click()

void ladder_diagram_app.MainWindow.ButtonImport_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Imports a project from a JSON file, replacing current content after confirmation.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 193 of file MainWindow.xaml.cs.

6.24.3.13 ButtonParentDevice_Click()

void ladder_diagram_app.MainWindow.ButtonParentDevice_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Opens a window to manage parent devices.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 385 of file MainWindow.xaml.cs.

6.24.3.14 ButtonSendToDevice_Click()

async void ladder_diagram_app.MainWindow.ButtonSendToDevice_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Sends the current configuration to the device.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 503 of file MainWindow.xaml.cs.

6.24.3.15 Canvas_DragOver()

void ladder_diagram_app.MainWindow.Canvas_DragOver (
object sender,
DragEventArgs e) [inline], [private]

Highlights valid drop positions during drag operations on the canvas.

Parameters

sender	The sender object.
e	The drag event arguments.

Definition at line 421 of file MainWindow.xaml.cs.

6.24.3.16 Canvas_Drop()

void ladder_diagram_app.MainWindow.Canvas_Drop (
object sender,
DragEventArgs e) [inline], [private]

Handles dropping elements onto the canvas.

Parameters

sender	The sender object.
e	The drag event arguments.

Definition at line 431 of file MainWindow.xaml.cs.

6.24.3.17 ComboBoxVariable_SelectionChanged()

void ladder_diagram_app.MainWindow.ComboBoxVariable_SelectionChanged (
object sender,
RoutedEventArgs e) [inline], [private]

Updates a variable's value when its combo box selection changes.

Parameters

sender	The sender object.
e	The selection changed event arguments.

Definition at line 328 of file MainWindow.xaml.cs.

6.24.3.18 MainCanvas_MouseLeftButtonDown()

void ladder_diagram_app.MainWindow.MainCanvas_MouseLeftButtonDown (
object sender,
MouseButtonEventArgs e) [inline], [private]

Handles mouse click for selecting elements or starting a drag operation.

Parameters

sender	The sender object.
e	The mouse button event arguments.

Definition at line 461 of file MainWindow.xaml.cs.

6.24.3.19 MainCanvas_MouseLeftButtonUp()

void ladder_diagram_app.MainWindow.MainCanvas_MouseLeftButtonUp (
object sender,
MouseButtonEventArgs e) [inline], [private]

Handles mouse release for placing elements on the canvas.

Parameters

sender	The sender object.
e	The mouse button event arguments.

Definition at line 451 of file MainWindow.xaml.cs.

6.24.3.20 MainCanvas_MouseMove()

void ladder_diagram_app.MainWindow.MainCanvas_MouseMove (
object sender,
MouseEventArgs e) [inline], [private]

Handles mouse movement for dragging elements on the canvas.

Parameters

sender	The sender object.
e	The mouse event arguments.

Definition at line 441 of file MainWindow.xaml.cs.

6.24.3.21 MainWindow_PreviewKeyDown()

void ladder_diagram_app.MainWindow.MainWindow_PreviewKeyDown (
object sender,
KeyEventArgs e) [inline], [private]

Handles keyboard input for deleting elements or adding variables.

Parameters

sender	The sender object.
e	The key event arguments.

Definition at line 158 of file MainWindow.xaml.cs.

6.24.3.22 MonitorExpander_Collapsed()

void ladder_diagram_app.MainWindow.MonitorExpander_Collapsed (
object sender,
RoutedEventArgs e) [inline], [private]

Adjusts the grid row height when the monitor expander is collapsed.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 557 of file MainWindow.xaml.cs.

6.24.3.23 OnConfigurationReceived()

void ladder_diagram_app.MainWindow.OnConfigurationReceived (
string jsonConfig) [inline], [private]

Handles received device configuration and updates the application state.

Parameters

jsonConfig	The JSON configuration string.

Definition at line 515 of file MainWindow.xaml.cs.

6.24.3.24 OnConnectionStatusChanged()

void ladder_diagram_app.MainWindow.OnConnectionStatusChanged (
bool isConnected) [inline], [private]

Updates UI based on device connection status changes.

Parameters

isConnected	True if connected, false otherwise.

Definition at line 526 of file MainWindow.xaml.cs.

6.24.3.25 TextBoxVariable_PreviewTextInput()

void ladder_diagram_app.MainWindow.TextBoxVariable_PreviewTextInput (
object sender,
TextCompositionEventArgs e) [inline], [private]

Restricts text box input to valid numbers or a minus sign.

Parameters

sender	The sender object.
e	The text composition event arguments.

Definition at line 342 of file MainWindow.xaml.cs.

6.24.3.26 TextBoxVariable_TextChanged()

void ladder_diagram_app.MainWindow.TextBoxVariable_TextChanged (
object sender,
RoutedEventArgs e) [inline], [private]

Updates a variable's value when its text box changes, triggered by Enter or focus loss.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 304 of file MainWindow.xaml.cs.

6.24.3.27 Window_Closing()

async void ladder_diagram_app.MainWindow.Window_Closing (
object? sender,
System.ComponentModel.CancelEventArgs e) [inline], [private]

Ensures proper cleanup when the application window is closing.

Parameters

sender	The sender object, can be null.
e	The cancel event arguments.

Definition at line 584 of file MainWindow.xaml.cs.

6.24.4 Member Data Documentation

6.24.4.1 _canvasElementFinder

readonly CanvasElementFinder ladder_diagram_app.MainWindow._canvasElementFinder [private]

Definition at line 60 of file MainWindow.xaml.cs.

6.24.4.2 _canvasInteractionManager

readonly CanvasInteractionManager ladder_diagram_app.MainWindow._canvasInteractionManager [private]

Definition at line 61 of file MainWindow.xaml.cs.

6.24.4.3 _canvasManager

readonly CanvasManager ladder_diagram_app.MainWindow._canvasManager [private]

Definition at line 59 of file MainWindow.xaml.cs.

6.24.4.4 _device

readonly Device ladder_diagram_app.MainWindow._device [private]

Definition at line 38 of file MainWindow.xaml.cs.

6.24.4.5 _deviceCommunicationManager

readonly DeviceCommunicationManager ladder_diagram_app.MainWindow._deviceCommunicationManager [private]

Definition at line 64 of file MainWindow.xaml.cs.

6.24.4.6 _devicePinManager

readonly DevicePinManager ladder_diagram_app.MainWindow._devicePinManager

Manages device pin configurations.

Definition at line 42 of file MainWindow.xaml.cs.

6.24.4.7 _importExportService

readonly ImportExportService ladder_diagram_app.MainWindow._importExportService [private]

Definition at line 71 of file MainWindow.xaml.cs.

6.24.4.8 _monitorDataService

readonly MonitorDataService ladder_diagram_app.MainWindow._monitorDataService [private]

Definition at line 67 of file MainWindow.xaml.cs.

6.24.4.9 _oneWireDataService

readonly OneWireDataService ladder_diagram_app.MainWindow._oneWireDataService [private]

Definition at line 68 of file MainWindow.xaml.cs.

6.24.4.10 _variablesManager

readonly VariablesManager ladder_diagram_app.MainWindow._variablesManager [private]

Definition at line 49 of file MainWindow.xaml.cs.

6.24.4.11 _wiresManager

readonly WiresManager ladder_diagram_app.MainWindow._wiresManager [private]

Definition at line 56 of file MainWindow.xaml.cs.

6.24.5 Property Documentation

6.24.5.1 AdcSensorSamplingRates

List<string> ladder_diagram_app.MainWindow.AdcSensorSamplingRates = [”10Hz”, ”40Hz”, ”Temperature”] [get]

List of supported ADC sensor sampling rates.

Definition at line 35 of file MainWindow.xaml.cs.

6.24.5.2 AdcSensorTypes

List<string> ladder_diagram_app.MainWindow.AdcSensorTypes = [”TM7711”, ”HX710B”] [get]

List of supported ADC sensor types.

Definition at line 31 of file MainWindow.xaml.cs.

6.24.5.3 DevicePinManager

DevicePinManager ladder_diagram_app.MainWindow.DevicePinManager [get]

Gets the device pin manager.

Definition at line 46 of file MainWindow.xaml.cs.

6.24.5.4 VariablesManager

VariablesManager ladder_diagram_app.MainWindow.VariablesManager [get]

Gets the variables manager.

Definition at line 53 of file MainWindow.xaml.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/MainWindow.xaml.cs

6.25 ladder_diagram_app.Services.MonitorServices.MonitorDataService Class Reference

Processes and displays monitor data received from a device in the main window.

Classes

- class MonitorVariable
  Represents a variable in the monitor data with properties for various variable types.

Public Member Functions

- MonitorDataService (MainWindow mainWindow)
  Initializes a new instance of the MonitorDataService class.
- void OnMonitorDataReceived (string monitorData)
  Handles incoming monitor data, deserializes it, and updates the main window's monitor display.

Private Attributes

- readonly MainWindow _mainWindow

6.25.1 Detailed Description

Processes and displays monitor data received from a device in the main window.

Definition at line 9 of file MonitorDataService.cs.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 MonitorDataService()

ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorDataService (
MainWindow mainWindow) [inline]

Initializes a new instance of the MonitorDataService class.

Parameters

mainWindow	The main window where monitor data will be displayed.

Definition at line 17 of file MonitorDataService.cs.

6.25.3 Member Function Documentation

6.25.3.1 OnMonitorDataReceived()

void ladder_diagram_app.Services.MonitorServices.MonitorDataService.OnMonitorDataReceived (
string monitorData) [inline]

Handles incoming monitor data, deserializes it, and updates the main window's monitor display.

Parameters

monitorData	The JSON string containing monitor data.

Definition at line 26 of file MonitorDataService.cs.

6.25.4 Member Data Documentation

6.25.4.1 _mainWindow

readonly MainWindow ladder_diagram_app.Services.MonitorServices.MonitorDataService._mainWindow [private]

Definition at line 11 of file MonitorDataService.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/MonitorServices/MonitorDataService.cs

6.26 ladder_diagram_app.Views.AddParentsWindow.MONITORINFO Struct Reference

Contains information about a monitor's size and work area.

Public Attributes

- int cbSize
- RECT rcMonitor
- RECT rcWork
- uint dwFlags

6.26.1 Detailed Description

Contains information about a monitor's size and work area.

Definition at line 42 of file AddParentsWindow.xaml.cs.

6.26.2 Member Data Documentation

6.26.2.1 cbSize

int ladder_diagram_app.Views.AddParentsWindow.MONITORINFO.cbSize

Definition at line 44 of file AddParentsWindow.xaml.cs.

6.26.2.2 dwFlags

uint ladder_diagram_app.Views.AddParentsWindow.MONITORINFO.dwFlags

Definition at line 47 of file AddParentsWindow.xaml.cs.

6.26.2.3 rcMonitor

RECT ladder_diagram_app.Views.AddParentsWindow.MONITORINFO.rcMonitor

Definition at line 45 of file AddParentsWindow.xaml.cs.

6.26.2.4 rcWork

RECT ladder_diagram_app.Views.AddParentsWindow.MONITORINFO.rcWork

Definition at line 46 of file AddParentsWindow.xaml.cs.

The documentation for this struct was generated from the following file:

- ladder_diagram_app/Views/AddParentsWindow.xaml.cs

6.27 ladder_diagram_app.Views.NotificationWindow.MONITORINFO Struct Reference

Contains information about a monitor's size and work area.

Public Attributes

- int cbSize
- RECT rcMonitor
- RECT rcWork
- uint dwFlags

6.27.1 Detailed Description

Contains information about a monitor's size and work area.

Definition at line 54 of file NotificationWindow.xaml.cs.

6.27.2 Member Data Documentation

6.27.2.1 cbSize

int ladder_diagram_app.Views.NotificationWindow.MONITORINFO.cbSize

Definition at line 56 of file NotificationWindow.xaml.cs.

6.27.2.2 dwFlags

uint ladder_diagram_app.Views.NotificationWindow.MONITORINFO.dwFlags

Definition at line 59 of file NotificationWindow.xaml.cs.

6.27.2.3 rcMonitor

RECT ladder_diagram_app.Views.NotificationWindow.MONITORINFO.rcMonitor

Definition at line 57 of file NotificationWindow.xaml.cs.

6.27.2.4 rcWork

RECT ladder_diagram_app.Views.NotificationWindow.MONITORINFO.rcWork

Definition at line 58 of file NotificationWindow.xaml.cs.

The documentation for this struct was generated from the following file:

- ladder_diagram_app/Views/NotificationWindow.xaml.cs

6.28 ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable Class Reference

Represents a variable in the monitor data with properties for various variable types.

Public Member Functions

- override string ToString ()

  Returns a string representation of the monitor variable, including all non-null properties.

Properties

- string? Type [get, set]
- string? Name [get, set]
- string? Pin [get, set]
- object? Value [get, set]
- string? SensorType [get, set]
- string? PD_SCK [get, set]
- string? DOUT [get, set]
- double? MapLow [get, set]
- double? MapHigh [get, set]
- double? Gain [get, set]
- string? SamplingRate [get, set]
- double? PV [get, set]
- double? CV [get, set]
- bool? CU [get, set]
- bool? CD [get, set]
- bool? QU [get, set]
- bool? QD [get, set]
- double? PT [get, set]
- double? ET [get, set]
- bool? IN [get, set]
- bool? Q [get, set]

6.28.1 Detailed Description

Represents a variable in the monitor data with properties for various variable types.

Definition at line 51 of file MonitorDataService.cs.

6.28.2 Member Function Documentation

6.28.2.1 ToString()

override string ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.ToString () [inline]

Returns a string representation of the monitor variable, including all non-null properties.

Definition at line 81 of file MonitorDataService.cs.

6.28.3 Property Documentation

6.28.3.1 CD

bool? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.CD [get], [set]

Definition at line 69 of file MonitorDataService.cs.

6.28.3.2 CU

bool? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.CU [get], [set]

Definition at line 68 of file MonitorDataService.cs.

6.28.3.3 CV

double? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.CV [get], [set]

Definition at line 67 of file MonitorDataService.cs.

6.28.3.4 DOUT

string? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.DOUT [get], [set]

Definition at line 60 of file MonitorDataService.cs.

6.28.3.5 ET

double? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.ET [get], [set]

Definition at line 74 of file MonitorDataService.cs.

6.28.3.6 Gain

double? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.Gain [get], [set]

Definition at line 63 of file MonitorDataService.cs.

6.28.3.7 IN

bool? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.IN [get], [set]

Definition at line 75 of file MonitorDataService.cs.

6.28.3.8 MapHigh

double? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.MapHigh [get], [set]

Definition at line 62 of file MonitorDataService.cs.

6.28.3.9 MapLow

double? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.MapLow [get], [set]

Definition at line 61 of file MonitorDataService.cs.

6.28.3.10 Name

string? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.Name [get], [set]

Definition at line 54 of file MonitorDataService.cs.

6.28.3.11 PD_SCK

string? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.PD_SCK [get], [set]

Definition at line 59 of file MonitorDataService.cs.

6.28.3.12 Pin

string? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.Pin [get], [set]

Definition at line 55 of file MonitorDataService.cs.

6.28.3.13 PT

double? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.PT [get], [set]

Definition at line 73 of file MonitorDataService.cs.

6.28.3.14 PV

double? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.PV [get], [set]

Definition at line 66 of file MonitorDataService.cs.

6.28.3.15 Q

bool? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.Q [get], [set]

Definition at line 76 of file MonitorDataService.cs.

6.28.3.16 QD

bool? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.QD [get], [set]

Definition at line 71 of file MonitorDataService.cs.

6.28.3.17 QU

bool? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.QU [get], [set]

Definition at line 70 of file MonitorDataService.cs.

6.28.3.18 SamplingRate

string? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.SamplingRate [get], [set]

Definition at line 64 of file MonitorDataService.cs.

6.28.3.19 SensorType

string? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.SensorType [get], [set]

Definition at line 58 of file MonitorDataService.cs.

6.28.3.20 Type

string? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.Type [get], [set]

Definition at line 53 of file MonitorDataService.cs.

6.28.3.21 Value

object? ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable.Value [get], [set]

Definition at line 56 of file MonitorDataService.cs.

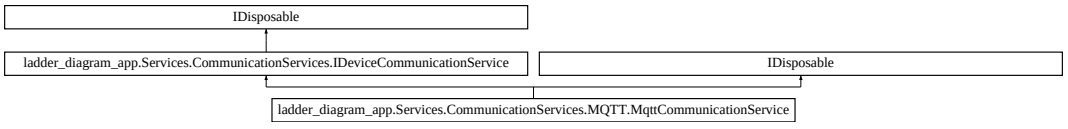
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/MonitorServices/MonitorDataService.cs

6.29 ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService Class Reference

Provides MQTT communication services for connecting to and interacting with a device using MQTT protocol.

Inheritance diagram for ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService:



Public Member Functions

- MqttCommunicationService ()

  Initializes a new instance of the MqttCommunicationService class.
- async Task< bool > ConnectAsync (string deviceId)

  Connects to the MQTT broker and initiates communication with the specified device.
- async Task DisconnectAsync ()

  Disconnects from the MQTT broker and cleans up resources.
- async Task RequestConnectionAsync ()

  Requests a connection to the device via MQTT.
- async Task RequestConfigurationAsync ()

  Requests the device's configuration via MQTT.
- async Task< bool > SendConfigurationAsync (string configJson)

  Sends a JSON configuration to the device in chunks.
- void Dispose ()

  Disposes of the service and releases resources.

Static Public Attributes

- static readonly string BrokerAddress = ConfigurationManager.AppSettings["MqttBrokerAddress"] ?? throw new ConfigurationErrorsException("MqttBrokerAddress is missing in App.config")
- static readonly int BrokerPort = int.Parse(ConfigurationManager.AppSettings["MqttBrokerPort"] ?? throw new ConfigurationErrorsException("MqttBrokerPort is missing in App.config"))
- static readonly? string BrokerUsername = ConfigurationManager.AppSettings["MqttBroker↩ Username"]
- static readonly? string BrokerPassword = ConfigurationManager.AppSettings["MqttBroker↩ Password"]

Protected Member Functions

- virtual void Dispose (bool disposing)

  Disposes of the service, releasing resources.

Properties

- bool IsConnected [get, private set]

  Gets a value indicating whether the service is connected to the device.
- string ConnectionType [get]

  Gets the type of connection, which is "MQTT".

Events

- EventHandler< string >? ConfigurationReceived
- EventHandler< string >? MonitorDataReceived
- EventHandler< string >? OneWireDataReceived
- EventHandler< bool >? ConnectionStatusChanged

Events inherited from ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService

- EventHandler< string > ConfigurationReceived

  Occurs when configuration data is received from the device.
- EventHandler< string > MonitorDataReceived

  Occurs when monitor data is received from the device.
- EventHandler< string > OneWireDataReceived

  Occurs when one-wire data is received from the device.
- EventHandler< bool > ConnectionStatusChanged

  Occurs when the connection status changes.

Private Member Functions

- void SetupEventHandlers ()

  Sets up MQTT client event handlers for connection, disconnection, and message receipt.
- async Task SubscribeToTopics ()

  Subscribes to relevant MQTT topics for the device.
- async Task UnsubscribeFromTopics ()

  Unsubscribes from all MQTT topics for the device.
- void HandleIncomingMessage (string topic, string message)

  Handles incoming MQTT messages based on their topic.
- void InitializePresentTimer ()

  Initializes a timer to periodically send "Present" messages to maintain connection.

Private Attributes

- readonly IMqttClient _mqttClient
- string _macAddress = string.Empty
- System.Timers.? Timer _presentTimer
- DateTime? _lastMonitorMessageTime = null
- bool _disposed = false

Static Private Attributes

- const string MqttTopicConnectionRequest = "/connection_request"
- const string MqttTopicConnectionResponse = "/connection_response"
- const string MqttTopicMonitor = "/monitor"
- const string MqttTopicOneWire = "/one_wire"
- const string MqttTopicConfigRequest = "/config_request"
- const string MqttTopicConfigResponse = "/config_response"
- const string MqttTopicConfig = "/config_device"
- const int ChunkSize = 800

6.29.1 Detailed Description

Provides MQTT communication services for connecting to and interacting with a device using MQTT protocol.

Definition at line 13 of file MqttCommunicationService.cs.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 MqttCommunicationService()

ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttCommunicationService () [inline]

Initializes a new instance of the MqttCommunicationService class.

Definition at line 54 of file MqttCommunicationService.cs.

6.29.3 Member Function Documentation

6.29.3.1 ConnectAsync()

async Task< bool > ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.ConnectAsync (
string deviceId) [inline]

Connects to the MQTT broker and initiates communication with the specified device.

Parameters

deviceId	The MAC address of the device to connect to.

Returns

True if connection is successful, otherwise false.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 94 of file MqttCommunicationService.cs.

6.29.3.2 DisconnectAsync()

async Task ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.DisconnectAsync () [inline]

Disconnects from the MQTT broker and cleans up resources.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 149 of file MqttCommunicationService.cs.

6.29.3.3 Dispose() [1/2]

void ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.Dispose () [inline]

Disposes of the service and releases resources.

Definition at line 410 of file MqttCommunicationService.cs.

6.29.3.4 Dispose() [2/2]

virtual void ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.Dispose (
bool disposing) [inline], [protected], [virtual]

Disposes of the service, releasing resources.

Parameters

disposing	Indicates whether to dispose managed resources.

Definition at line 382 of file MqttCommunicationService.cs.

6.29.3.5 HandleIncomingMessage()

void ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.HandleIncomingMessage (
string topic,
string message) [inline], [private]

Handles incoming MQTT messages based on their topic.

Parameters

topic	The topic of the message.
message	The message payload.

Definition at line 227 of file MqttCommunicationService.cs.

6.29.3.6 InitializePresentTimer()

void ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.InitializePresentTimer () [inline], [private]

Initializes a timer to periodically send "Present" messages to maintain connection.

Definition at line 301 of file MqttCommunicationService.cs.

6.29.3.7 RequestConfigurationAsync()

async Task ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.RequestConfigurationAsync () [inline]

Requests the device's configuration via MQTT.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 279 of file MqttCommunicationService.cs.

6.29.3.8 RequestConnectionAsync()

async Task ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.RequestConnectionAsync () [inline]

Requests a connection to the device via MQTT.

Definition at line 257 of file MqttCommunicationService.cs.

6.29.3.9 SendConfigurationAsync()

async Task< bool > ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.SendConfigurationAsync (
string configJson) [inline]

Sends a JSON configuration to the device in chunks.

Parameters

configJson	The JSON configuration string to send.

Returns

True if sending is successful, otherwise false.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 337 of file MqttCommunicationService.cs.

6.29.3.10 SetupEventHandlers()

void ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.SetupEventHandlers () [inline], [private]

Sets up MQTT client event handlers for connection, disconnection, and message receipt.

Definition at line 64 of file MqttCommunicationService.cs.

6.29.3.11 SubscribeToTopics()

async Task ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.SubscribeToTopics () [inline], [private]

Subscribes to relevant MQTT topics for the device.

Definition at line 182 of file MqttCommunicationService.cs.

6.29.3.12 UnsubscribeFromTopics()

async Task ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.UnsubscribeFrom↩Topics () [inline], [private]

Unsubscribes from all MQTT topics for the device.

Definition at line 204 of file MqttCommunicationService.cs.

6.29.4 Member Data Documentation

6.29.4.1 _disposed

bool ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService._disposed = false [private]

Definition at line 49 of file MqttCommunicationService.cs.

6.29.4.2 _lastMonitorMessageTime

DateTime? ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService._lastMonitorMessageTime = null [private]

Definition at line 47 of file MqttCommunicationService.cs.

6.29.4.3 _macAddress

string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService._macAddress = string.Empty [private]

Definition at line 16 of file MqttCommunicationService.cs.

6.29.4.4 _mqttClient

readonly IMqttClient ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService._mqttClient [private]

Definition at line 15 of file MqttCommunicationService.cs.

6.29.4.5 _presentTimer

System.Timers.? Timer ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService._presentTimer [private]

Definition at line 46 of file MqttCommunicationService.cs.

6.29.4.6 BrokerAddress

readonly string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.BrokerAddress = ConfigurationManager.AppSettings["MqttBrokerAddress"] ?? throw new ConfigurationErrorsException("MqttBrokerAddress is missing in App.config") [static]

Definition at line 18 of file MqttCommunicationService.cs.

6.29.4.7 BrokerPassword

readonly? string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.BrokerPassword = ConfigurationManager.AppSettings["MqttBrokerPassword"] [static]

Definition at line 21 of file MqttCommunicationService.cs.

6.29.4.8 BrokerPort

readonly int ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.BrokerPort = int.Parse(ConfigurationManager.AppSettings["MqttBrokerPort"] ?? throw new ConfigurationErrorsException("MqttBrokerPort is missing in App.config")) [static]

Definition at line 19 of file MqttCommunicationService.cs.

6.29.4.9 BrokerUsername

readonly? string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.BrokerUsername = ConfigurationManager.AppSettings["MqttBrokerUsername"] [static]

Definition at line 20 of file MqttCommunicationService.cs.

6.29.4.10 ChunkSize

const int ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.ChunkSize = 800 [static], [private]

Definition at line 48 of file MqttCommunicationService.cs.

6.29.4.11 MqttTopicConfig

const string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttTopicConfig = "/config_device" [static], [private]

Definition at line 29 of file MqttCommunicationService.cs.

6.29.4.12 MqttTopicConfigRequest

const string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttTopicConfigRequest = "/config_request" [static], [private]

Definition at line 27 of file MqttCommunicationService.cs.

6.29.4.13 MqttTopicConfigResponse

const string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttTopicConfigResponse = "/config_response" [static], [private]

Definition at line 28 of file MqttCommunicationService.cs.

6.29.4.14 MqttTopicConnectionRequest

const string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttTopicConnectionRequest = "/connection_request" [static], [private]

Definition at line 23 of file MqttCommunicationService.cs.

6.29.4.15 MqttTopicConnectionResponse

const string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttTopicConnectionResponse = "/connection_response" [static], [private]

Definition at line 24 of file MqttCommunicationService.cs.

6.29.4.16 MqttTopicMonitor

const string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttTopicMonitor = "/monitor" [static], [private]

Definition at line 25 of file MqttCommunicationService.cs.

6.29.4.17 MqttTopicOneWire

const string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.MqttTopicOneWire = "/one_wire" [static], [private]

Definition at line 26 of file MqttCommunicationService.cs.

6.29.5 Property Documentation

6.29.5.1 ConnectionType

string ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.ConnectionType [get]

Gets the type of connection, which is "MQTT".

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 44 of file MqttCommunicationService.cs.

6.29.5.2 IsConnected

bool ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.IsConnected [get], [private set]

Gets a value indicating whether the service is connected to the device.

Implements ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService.

Definition at line 39 of file MqttCommunicationService.cs.

6.29.6 Event Documentation

6.29.6.1 ConfigurationReceived

EventHandler<string>? ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.ConfigurationReceived

Definition at line 31 of file MqttCommunicationService.cs.

6.29.6.2 ConnectionStatusChanged

EventHandler<bool>? ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.↩
ConnectionStatusChanged

Definition at line 34 of file MqttCommunicationService.cs.

6.29.6.3 MonitorDataReceived

EventHandler<string>? ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.↩
MonitorDataReceived

Definition at line 32 of file MqttCommunicationService.cs.

6.29.6.4 OneWireDataReceived

EventHandler<string>? ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService.One↩
WireDataReceived

Definition at line 33 of file MqttCommunicationService.cs.

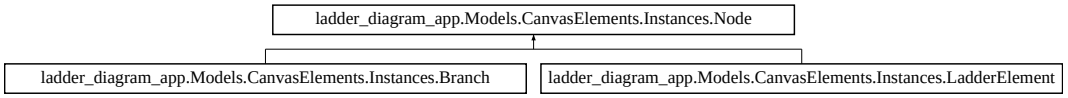
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/CommunicationServices/MQTT/MqttCommunicationService.cs

6.30 ladder_diagram_app.Models.CanvasElements.Instances.Node Class Reference

Abstract base class for nodes in a ladder diagram, providing common properties and methods for positioning and highlighting.

Inheritance diagram for ladder_diagram_app.Models.CanvasElements.Instances.Node:



Public Member Functions

- void HighlightNode ()

  Highlights the node by applying a red drop shadow effect to its image.
- void UnhighlightNode ()

  Removes the highlight effect from the node by clearing its image effect.

Properties

- double X [get, set]

  Gets or sets the X-coordinate of the node.
- double Y [get, set]

  Gets or sets the Y-coordinate of the node.
- object? Parent [get, set]

  Gets or sets the parent object of the node, used to track hierarchical relationships.
- double Width [get]

  Gets the width of the node, typically based on its visual representation.
- virtual ? Image Image [get, set]

  Gets or sets the image representing the node visually, if applicable.

6.30.1 Detailed Description

Abstract base class for nodes in a ladder diagram, providing common properties and methods for positioning and highlighting.

Definition at line 10 of file Node.cs.

6.30.2 Member Function Documentation

6.30.2.1 HighlightNode()

void ladder_diagram_app.Models.CanvasElements.Instances.Node.HighlightNode () [inline]

Highlights the node by applying a red drop shadow effect to its image.

Definition at line 41 of file Node.cs.

6.30.2.2 UnhighlightNode()

void ladder_diagram_app.Models.CanvasElements.Instances.Node.UnhighlightNode () [inline]

Removes the highlight effect from the node by clearing its image effect.

Definition at line 60 of file Node.cs.

6.30.3 Property Documentation

6.30.3.1 Image

virtual ? Image ladder_diagram_app.Models.CanvasElements.Instances.Node.Image [get], [set]

Gets or sets the image representing the node visually, if applicable.

Definition at line 35 of file Node.cs.

6.30.3.2 Parent

object? ladder_diagram_app.Models.CanvasElements.Instances.Node.Parent [get], [set]

Gets or sets the parent object of the node, used to track hierarchical relationships.

Definition at line 25 of file Node.cs.

6.30.3.3 Width

double ladder_diagram_app.Models.CanvasElements.Instances.Node.Width [get], [abstract]

Gets the width of the node, typically based on its visual representation.

Definition at line 30 of file Node.cs.

6.30.3.4 X

double ladder_diagram_app.Models.CanvasElements.Instances.Node.X [get], [set], [abstract]

Gets or sets the X-coordinate of the node.

Definition at line 15 of file Node.cs.

6.30.3.5 Y

double ladder_diagram_app.Models.CanvasElements.Instances.Node.Y [get], [set], [abstract]

Gets or sets the Y-coordinate of the node.

Definition at line 20 of file Node.cs.

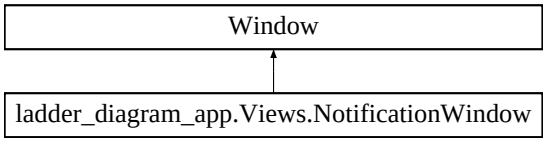
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/CanvasElements/Instances/Node.cs

6.31 ladder_diagram_app.Views.NotificationWindow Class Reference

A customizable notification window that supports various button configurations and input fields.

Inheritance diagram for ladder_diagram_app.Views.NotificationWindow:



Classes

- struct MONITORINFO

  Contains information about a monitor's size and work area.
- struct RECT

  Represents a rectangle with left, top, right, and bottom coordinates.

Public Member Functions

- NotificationWindow (string poruka, Window owner, NotificationButtons buttons=NotificationButtons.None, string[] inputLabels=null)

  Initializes a new instance of the NotificationWindow class.

Properties

- bool? Result [get]

  Gets the dialog result (true for Yes/Confirm, false for No/Cancel, null if not set).
- string[] InputResults [get]

  Gets the array of input field values for input-based dialogs.

Private Member Functions

- static IntPtr MonitorFromWindow (IntPtr hwnd, uint dwFlags)
- static bool GetMonitorInfo (IntPtr hMonitor, ref MONITORINFO lpmi)
- bool AreInputsValid (NotificationButtons buttons)

  Validates input fields to ensure they are not empty for input-based dialogs.
- void AdjustLabelWidths ()

  Adjusts the width of visible input labels to match the widest label for consistent alignment.
- void UpdatePosition ()

  Updates the position of the notification window to center it relative to the owner window or monitor.
- void Owner_PositionOrSizeChanged (object sender, EventArgs e)

  Handles changes in the owner window's position or size to reposition the notification window.
- void Owner_StateChanged (object sender, EventArgs e)

  Handles changes in the owner window's state (e.g., maximized/minimized) to reposition the notification window.

Private Attributes

- bool? _result = null
- string[] _inputResults = null

Static Private Attributes

- const uint MONITOR_DEFAULTTONEAREST = 2

6.31.1 Detailed Description

A customizable notification window that supports various button configurations and input fields.

Definition at line 27 of file NotificationWindow.xaml.cs.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 NotificationWindow()

ladder_diagram_app.Views.NotificationWindow.NotificationWindow (
string poruka,
Window owner,
NotificationButtons buttons = NotificationButtons::None,
string[] inputLabels = null) [inline]

Initializes a new instance of the NotificationWindow class.

Parameters

poruka	The message to display in the notification.
owner	The parent window that owns this dialog.
buttons	The button configuration for the dialog.
inputLabels	Optional labels for input fields (used for OneInput, TwoInputs, ThreeInputs).

Definition at line 82 of file NotificationWindow.xaml.cs.

6.31.3 Member Function Documentation

6.31.3.1 AdjustLabelWidths()

void ladder_diagram_app.Views.NotificationWindow.AdjustLabelWidths () [inline], [private]

Adjusts the width of visible input labels to match the widest label for consistent alignment.

Definition at line 333 of file NotificationWindow.xaml.cs.

6.31.3.2 AreInputsValid()

bool ladder_diagram_app.Views.NotificationWindow.AreInputsValid (
NotificationButtons buttons) [inline], [private]

Validates input fields to ensure they are not empty for input-based dialogs.

Parameters

buttons	The button configuration to validate.

Returns

True if all required inputs are valid; otherwise, false.

Definition at line 310 of file NotificationWindow.xaml.cs.

6.31.3.3 GetMonitorInfo()

static bool ladder_diagram_app.Views.NotificationWindow.GetMonitorInfo (
IntPtr hMonitor,
ref MONITORINFO lpmi) [private]

6.31.3.4 MonitorFromWindow()

static IntPtr ladder_diagram_app.Views.NotificationWindow.MonitorFromWindow (
IntPtr hwnd,
uint dwFlags) [private]

6.31.3.5 Owner_PositionOrSizeChanged()

void ladder_diagram_app.Views.NotificationWindow.Owner_PositionOrSizeChanged (
object sender,
EventArgs e) [inline], [private]

Handles changes in the owner window's position or size to reposition the notification window.

Definition at line 409 of file NotificationWindow.xaml.cs.

6.31.3.6 Owner_StateChanged()

void ladder_diagram_app.Views.NotificationWindow.Owner_StateChanged (
object sender,
EventArgs e) [inline], [private]

Handles changes in the owner window's state (e.g., maximized/minimized) to reposition the notification window.

Definition at line 417 of file NotificationWindow.xaml.cs.

6.31.3.7 UpdatePosition()

void ladder_diagram_app.Views.NotificationWindow.UpdatePosition () [inline], [private]

Updates the position of the notification window to center it relative to the owner window or monitor.

Definition at line 359 of file NotificationWindow.xaml.cs.

6.31.4 Member Data Documentation

6.31.4.1 _inputResults

string [] ladder_diagram_app.Views.NotificationWindow._inputResults = null [private]

Definition at line 63 of file NotificationWindow.xaml.cs.

6.31.4.2 _result

bool? ladder_diagram_app.Views.NotificationWindow._result = null [private]

Definition at line 62 of file NotificationWindow.xaml.cs.

6.31.4.3 MONITOR_DEFAULTTONEAREST

const uint ladder_diagram_app.Views.NotificationWindow.MONITOR_DEFAULTTONEAREST = 2 [static], [private]

Definition at line 36 of file NotificationWindow.xaml.cs.

6.31.5 Property Documentation

6.31.5.1 InputResults

string [] ladder_diagram_app.Views.NotificationWindow.InputResults [get]

Gets the array of input field values for input-based dialogs.

Definition at line 73 of file NotificationWindow.xaml.cs.

6.31.5.2 Result

bool? ladder_diagram_app.Views.NotificationWindow.Result [get]

Gets the dialog result (true for Yes/Confirm, false for No/Cancel, null if not set).

Definition at line 68 of file NotificationWindow.xaml.cs.

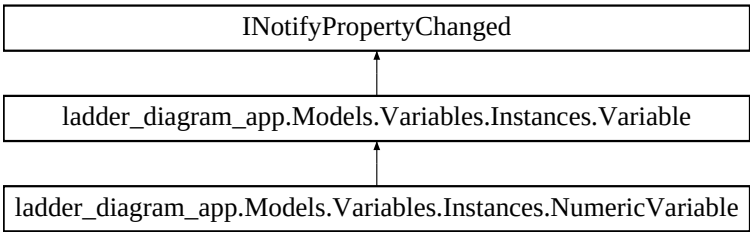
The documentation for this class was generated from the following file:

- ladder_diagram_app/Views/NotificationWindow.xaml.cs

6.32 ladder_diagram_app.Models.Variables.Instances.NumericVariable Class Reference

Represents a numeric variable in a ladder diagram, encapsulating a double-precision value.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.NumericVariable:



Public Member Functions

- NumericVariable ()
  Initializes a new instance of the NumericVariable class with default values.
- override Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Properties

- double Value [get, set]
  Gets or sets the numeric value of the variable, notifying subscribers on change.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]
  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]
  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]
  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- double _value
  Stores the numeric value of the variable.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.32.1 Detailed Description

Represents a numeric variable in a ladder diagram, encapsulating a double-precision value.

Definition at line 6 of file NumericVariable.cs.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 NumericVariable()

ladder_diagram_app.Models.Variables.Instances.NumericVariable.NumericVariable () [inline]

Initializes a new instance of the NumericVariable class with default values.

Definition at line 25 of file NumericVariable.cs.

6.32.3 Member Function Documentation

6.32.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.NumericVariable.ToExport↩ Dictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 35 of file NumericVariable.cs.

6.32.4 Member Data Documentation

6.32.4.1 _value

double ladder_diagram_app.Models.Variables.Instances.NumericVariable._value [private]

Stores the numeric value of the variable.

Definition at line 11 of file NumericVariable.cs.

6.32.5 Property Documentation

6.32.5.1 Value

double ladder_diagram_app.Models.Variables.Instances.NumericVariable.Value [get], [set]

Gets or sets the numeric value of the variable, notifying subscribers on change.

Definition at line 16 of file NumericVariable.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/NumericVariable.cs

6.33 ladder_diagram_app.Services.MonitorServices.OneWireDataService Class Reference

Manages one-wire sensor data processing and UI updates for the main window.

Classes

- class OneWireSensor

  Represents a one-wire sensor with address and type information.
- class OneWireSensorViewModel

  Represents a view model for one-wire sensors, used for UI display.

Public Member Functions

- OneWireDataService (MainWindow mainWindow, Device device)

  Initializes a new instance of the OneWireDataService class.
- void DeleteLastOneWireMessage ()

  Clears the last stored one-wire message.
- void OnOneWireDataReceived (string oneWireData)

  Processes incoming one-wire data and updates the UI asynchronously.
- void ActionButton_Click (object sender, RoutedEventArgs e)

  Handles the click event for action buttons to add or remove one-wire sensors.

Private Member Functions

- void ProcessOneWireMessage (string message)

  Processes a one-wire message, parsing it and refreshing the sensor list if it differs from the last message.
- void RefreshOneWireSensors (JsonElement? root=null)

  Refreshes the list of one-wire sensors displayed in the UI based on device configuration and optional MQTT data.

Private Attributes

- readonly MainWindow _mainWindow
- readonly Device _device
- string? _lastOneWireMessage

6.33.1 Detailed Description

Manages one-wire sensor data processing and UI updates for the main window.

Definition at line 13 of file OneWireDataService.cs.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 OneWireDataService()

ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireDataService (
MainWindow mainWindow,
Device device) [inline]

Initializes a new instance of the OneWireDataService class.

Parameters

main↩ Window	The main window for UI updates and notifications.
device	The device configuration containing one-wire input data.

Definition at line 24 of file OneWireDataService.cs.

6.33.3 Member Function Documentation

6.33.3.1 ActionButton_Click()

void ladder_diagram_app.Services.MonitorServices.OneWireDataService.ActionButton_Click (
object sender,
RoutedEventArgs e) [inline]

Handles the click event for action buttons to add or remove one-wire sensors.

Parameters

sender	The button that triggered the event.
e	The routed event arguments.

Definition at line 154 of file OneWireDataService.cs.

6.33.3.2 DeleteLastOneWireMessage()

void ladder_diagram_app.Services.MonitorServices.OneWireDataService.DeleteLastOneWireMessage () [inline]

Clears the last stored one-wire message.

Definition at line 34 of file OneWireDataService.cs.

6.33.3.3 OnOneWireDataReceived()

void ladder_diagram_app.Services.MonitorServices.OneWireDataService.OnOneWireDataReceived (
string oneWireData) [inline]

Processes incoming one-wire data and updates the UI asynchronously.

Parameters

oneWireData	The JSON string containing one-wire sensor data.

Definition at line 43 of file OneWireDataService.cs.

6.33.3.4 ProcessOneWireMessage()

void ladder_diagram_app.Services.MonitorServices.OneWireDataService.ProcessOneWireMessage (
string message) [inline], [private]

Processes a one-wire message, parsing it and refreshing the sensor list if it differs from the last message.

Parameters

message	The JSON string to process.

Definition at line 58 of file OneWireDataService.cs.

6.33.3.5 RefreshOneWireSensors()

void ladder_diagram_app.Services.MonitorServices.OneWireDataService.RefreshOneWireSensors (
JsonElement? root = null) [inline], [private]

Refreshes the list of one-wire sensors displayed in the UI based on device configuration and optional MQTT data.

Parameters

root	The JSON root element containing MQTT data, if available.

Definition at line 82 of file OneWireDataService.cs.

6.33.4 Member Data Documentation

6.33.4.1 _device

readonly Device ladder_diagram_app.Services.MonitorServices.OneWireDataService._device [private]

Definition at line 16 of file OneWireDataService.cs.

6.33.4.2 _lastOneWireMessage

string? ladder_diagram_app.Services.MonitorServices.OneWireDataService._lastOneWireMessage [private]

Definition at line 17 of file OneWireDataService.cs.

6.33.4.3 _mainWindow

readonly MainWindow ladder_diagram_app.Services.MonitorServices.OneWireDataService._mainWindow [private]

Definition at line 15 of file OneWireDataService.cs.

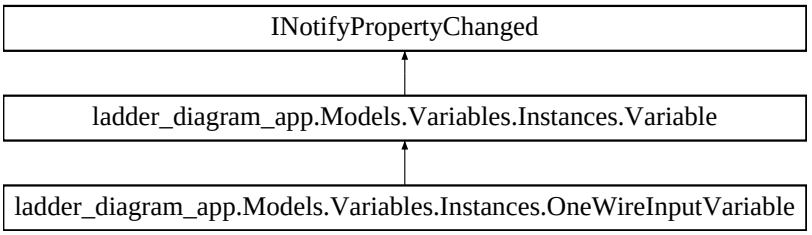
The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/MonitorServices/OneWireDataService.cs

6.34 ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable Class Reference

Represents a one-wire input variable in a ladder diagram, associated with a specific pin.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable:



Public Member Functions

- OneWireInputVariable ()
  Initializes a new instance of the OneWireInputVariable class with an empty pin name.
- override Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Properties

- string PinName [get, set]
  Gets or sets the name of the pin associated with the one-wire input, notifying subscribers on change.
- bool IsValid [get]
  Gets a value indicating whether the variable is valid based on the presence of a pin name.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]
  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]
  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]
  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- string _pinName
  Stores the name of the pin associated with the one-wire input.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.34.1 Detailed Description

Represents a one-wire input variable in a ladder diagram, associated with a specific pin.

Definition at line 6 of file OneWireInputVariable.cs.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 OneWireInputVariable()

ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable.OneWireInputVariable () [inline]

Initializes a new instance of the OneWireInputVariable class with an empty pin name.

Definition at line 16 of file OneWireInputVariable.cs.

6.34.3 Member Function Documentation

6.34.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable.ToExport↩
Dictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 39 of file OneWireInputVariable.cs.

6.34.4 Member Data Documentation

6.34.4.1 _pinName

string ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable._pinName [private]

Stores the name of the pin associated with the one-wire input.

Definition at line 11 of file OneWireInputVariable.cs.

6.34.5 Property Documentation

6.34.5.1 IsValid

bool ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable.IsValid [get]

Gets a value indicating whether the variable is valid based on the presence of a pin name.

Definition at line 33 of file OneWireInputVariable.cs.

6.34.5.2 PinName

string ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable.PinName [get], [set]

Gets or sets the name of the pin associated with the one-wire input, notifying subscribers on change.

Definition at line 24 of file OneWireInputVariable.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/OneWireInputVariable.cs

6.35 ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensor Class Reference

Represents a one-wire sensor with address and type information.

Public Member Functions

- OneWireSensor (string address)

Properties

- string Address [get, set]
- string Type [get, set]

Private Member Functions

- string GetSensorType (string address)
  Determines the sensor type based on the family code in the address.

6.35.1 Detailed Description

Represents a one-wire sensor with address and type information.

Definition at line 283 of file OneWireDataService.cs.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 OneWireSensor()

ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensor.OneWireSensor (
string address) [inline]

Definition at line 288 of file OneWireDataService.cs.

6.35.3 Member Function Documentation

6.35.3.1 GetSensorType()

string ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensor.GetSensorType (
string address) [inline], [private]

Determines the sensor type based on the family code in the address.

Parameters

address	The sensor address.

Returns

The sensor type or "Unknown" if not recognized.

Definition at line 299 of file OneWireDataService.cs.

6.35.4 Property Documentation

6.35.4.1 Address

string ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensor.Address [get], [set]

Definition at line 285 of file OneWireDataService.cs.

6.35.4.2 Type

string ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensor.Type [get], [set]

Definition at line 286 of file OneWireDataService.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/MonitorServices/OneWireDataService.cs

6.36 ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel Class Reference

Represents a view model for one-wire sensors, used for UI display.

Properties

- int Pin [get, set]
- string? Address [get, set]
- string? Type [get, set]
- string? SensorName [get, set]
- bool IsInDevice [get, set]
- bool IsFromMqtt [get, set]
- bool IsInDeviceAndFromMqtt [get]
- bool IsInDeviceAndNotFromMqtt [get]
- bool IsNotInDeviceAndFromMqtt [get]

6.36.1 Detailed Description

Represents a view model for one-wire sensors, used for UI display.

Definition at line 267 of file OneWireDataService.cs.

6.36.2 Property Documentation

6.36.2.1 Address

string? ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.Address [get], [set]

Definition at line 270 of file OneWireDataService.cs.

6.36.2.2 IsFromMqtt

bool ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.IsFromMqtt [get], [set]

Definition at line 274 of file OneWireDataService.cs.

6.36.2.3 IsInDevice

bool ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.IsInDevice [get], [set]

Definition at line 273 of file OneWireDataService.cs.

6.36.2.4 IsInDeviceAndFromMqtt

bool ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.IsInDeviceAnd↩FromMqtt [get]

Definition at line 275 of file OneWireDataService.cs.

6.36.2.5 IsInDeviceAndNotFromMqtt

bool ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.IsInDeviceAndNot↩FromMqtt [get]

Definition at line 276 of file OneWireDataService.cs.

6.36.2.6 IsNotInDeviceAndFromMqtt

bool ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.IsNotInDeviceAnd↩FromMqtt [get]

Definition at line 277 of file OneWireDataService.cs.

6.36.2.7 Pin

int ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.Pin [get], [set]

Definition at line 269 of file OneWireDataService.cs.

6.36.2.8 SensorName

string? ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.SensorName [get], [set]

Definition at line 272 of file OneWireDataService.cs.

6.36.2.9 Type

string? ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel.Type [get], [set]

Definition at line 271 of file OneWireDataService.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Services/MonitorServices/OneWireDataService.cs

6.37 ladder_diagram_app.Views.AddParentsWindow.RECT Struct Reference

Represents a rectangle with left, top, right, and bottom coordinates.

Public Attributes

- int Left
- int Top
- int Right
- int Bottom

6.37.1 Detailed Description

Represents a rectangle with left, top, right, and bottom coordinates.

Definition at line 30 of file AddParentsWindow.xaml.cs.

6.37.2 Member Data Documentation

6.37.2.1 Bottom

int ladder_diagram_app.Views.AddParentsWindow.RECT.Bottom

Definition at line 35 of file AddParentsWindow.xaml.cs.

6.37.2.2 Left

int ladder_diagram_app.Views.AddParentsWindow.RECT.Left

Definition at line 32 of file AddParentsWindow.xaml.cs.

6.37.2.3 Right

int ladder_diagram_app.Views.AddParentsWindow.RECT.Right

Definition at line 34 of file AddParentsWindow.xaml.cs.

6.37.2.4 Top

int ladder_diagram_app.Views.AddParentsWindow.RECT.Top

Definition at line 33 of file AddParentsWindow.xaml.cs.

The documentation for this struct was generated from the following file:

- ladder_diagram_app/Views/AddParentsWindow.xaml.cs

6.38 ladder_diagram_app.Views.NotificationWindow.RECT Struct Reference

Represents a rectangle with left, top, right, and bottom coordinates.

Public Attributes

- int Left
- int Top
- int Right
- int Bottom

6.38.1 Detailed Description

Represents a rectangle with left, top, right, and bottom coordinates.

Definition at line 42 of file NotificationWindow.xaml.cs.

6.38.2 Member Data Documentation

6.38.2.1 Bottom

int ladder_diagram_app.Views.NotificationWindow.RECT.Bottom

Definition at line 47 of file NotificationWindow.xaml.cs.

6.38.2.2 Left

int ladder_diagram_app.Views.NotificationWindow.RECT.Left

Definition at line 44 of file NotificationWindow.xaml.cs.

6.38.2.3 Right

int ladder_diagram_app.Views.NotificationWindow.RECT.Right

Definition at line 46 of file NotificationWindow.xaml.cs.

6.38.2.4 Top

int ladder_diagram_app.Views.NotificationWindow.RECT.Top

Definition at line 45 of file NotificationWindow.xaml.cs.

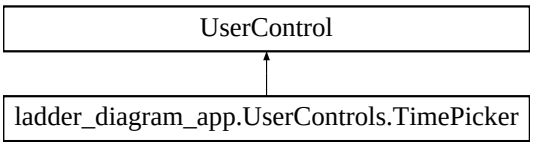
The documentation for this struct was generated from the following file:

- ladder_diagram_app/Views/NotificationWindow.xaml.cs

6.39 ladder_diagram_app.UserControls.TimePicker Class Reference

A user control for selecting and displaying time in a HH:mm:ss format.

Inheritance diagram for ladder_diagram_app.UserControls.TimePicker:



Public Member Functions

- TimePicker ()
  Initializes a new instance of the TimePicker class.

Static Public Attributes

- static readonly DependencyProperty SelectedTimeProperty
  Dependency property for the selected time, stored as a double in the format HHmmss.0.

Properties

- double SelectedTime [get, set]
  Gets or sets the selected time as a double in the format HHmmss.0.

Private Member Functions

- void InitializeComboBoxes ()
  Populates the hours, minutes, and seconds combo boxes with valid values.
- void UpdateDisplayFromSelectedTime ()
  Updates the combo boxes and time display based on the current SelectedTime value.
- void TimeDisplay_MouseLeftButtonUp (object sender, System.Windows.Input.MouseButton↩ EventArgs e)
  Opens the configuration popup when the time display is clicked.
- void ComboBox_SelectionChanged (object sender, SelectionChangedEventArgs e)
  Updates the time preview when a combo box selection changes.
- void ApplyButton_Click (object sender, RoutedEventArgs e)
  Applies the selected time and closes the configuration popup.
- void UpdateTimePreview ()
  Updates the SelectedTime and time display based on the combo box selections.

Static Private Member Functions

- static void OnSelectedTimeChanged (DependencyObject d, DependencyPropertyChangedEvent↩ Args e)
  Handles changes to the SelectedTime property and updates the UI.

6.39.1 Detailed Description

A user control for selecting and displaying time in a HH:mm:ss format.

Definition at line 10 of file TimePicker.xaml.cs.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 TimePicker()

ladder_diagram_app.UserControls.TimePicker.TimePicker () [inline]

Initializes a new instance of the TimePicker class.

Definition at line 37 of file TimePicker.xaml.cs.

6.39.3 Member Function Documentation

6.39.3.1 ApplyButton_Click()

void ladder_diagram_app.UserControls.TimePicker.ApplyButton_Click (
object sender,
RoutedEventArgs e) [inline], [private]

Applies the selected time and closes the configuration popup.

Parameters

sender	The sender object.
e	The routed event arguments.

Definition at line 115 of file TimePicker.xaml.cs.

6.39.3.2 ComboBox_SelectionChanged()

void ladder_diagram_app.UserControls.TimePicker.ComboBox_SelectionChanged (
object sender,
SelectionChangedEventArgs e) [inline], [private]

Updates the time preview when a combo box selection changes.

Parameters

sender	The sender object.
e	The selection changed event arguments.

Definition at line 105 of file TimePicker.xaml.cs.

6.39.3.3 InitializeComboBoxes()

void ladder_diagram_app.UserControls.TimePicker.InitializeComboBoxes () [inline], [private]

Populates the hours, minutes, and seconds combo boxes with valid values.

Definition at line 46 of file TimePicker.xaml.cs.

6.39.3.4 OnSelectedTimeChanged()

static void ladder_diagram_app.UserControls.TimePicker.OnSelectedTimeChanged (
DependencyObject d,
DependencyPropertyChangedEventArgs e) [inline], [static], [private]

Handles changes to the SelectedTime property and updates the UI.

Parameters

d	The dependency object.
e	The event arguments.

Definition at line 64 of file TimePicker.xaml.cs.

6.39.3.5 TimeDisplay_MouseLeftButtonUp()

void ladder_diagram_app.UserControls.TimePicker.TimeDisplay_MouseLeftButtonUp (
object sender,
System.Windows.Input.MouseButtonEventArgs e) [inline], [private]

Opens the configuration popup when the time display is clicked.

Parameters

sender	The sender object.
e	The mouse button event arguments.

Definition at line 94 of file TimePicker.xaml.cs.

6.39.3.6 UpdateDisplayFromSelectedTime()

void ladder_diagram_app.UserControls.TimePicker.UpdateDisplayFromSelectedTime () [inline], [private]

Updates the combo boxes and time display based on the current SelectedTime value.

Definition at line 73 of file TimePicker.xaml.cs.

6.39.3.7 UpdateTimePreview()

void ladder_diagram_app.UserControls.TimePicker.UpdateTimePreview () [inline], [private]

Updates the SelectedTime and time display based on the combo box selections.

Definition at line 124 of file TimePicker.xaml.cs.

6.39.4 Member Data Documentation

6.39.4.1 SelectedTimeProperty

readonly DependencyProperty ladder_diagram_app.UserControls.TimePicker.SelectedTimeProperty [static]

Initial value:

```
=
        DependencyProperty.Register(
            nameof(SelectedTime),
            typeof(double),
            typeof(TimePicker),
            new FrameworkPropertyMetadata(
                0.0,
                FrameworkPropertyMetadataOptions.BindsTwoWayByDefault,
                OnSelectedTimeChanged))
```

Dependency property for the selected time, stored as a double in the format HHmmss.0.

Definition at line 15 of file TimePicker.xaml.cs.

6.39.5 Property Documentation

6.39.5.1 SelectedTime

double ladder_diagram_app.UserControls.TimePicker.SelectedTime [get], [set]

Gets or sets the selected time as a double in the format HHmmss.0.

Definition at line 28 of file TimePicker.xaml.cs.

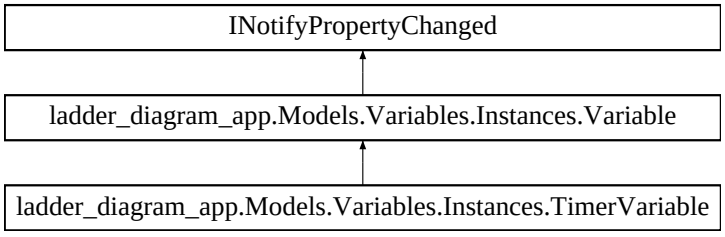
The documentation for this class was generated from the following file:

- ladder_diagram_app/UserControls/TimePicker.xaml.cs

6.40 ladder_diagram_app.Models.Variables.Instances.TimerVariable Class Reference

Represents a timer variable in a ladder diagram, encapsulating preset time, elapsed time, input, and output states.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.TimerVariable:



Public Member Functions

- TimerVariable ()
  Initializes a new instance of the TimerVariable class with default values.
- override Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Properties

- double PT [get, set]
  Gets or sets the preset time (PT) of the timer, notifying subscribers on change.
- double ET [get, set]
  Gets or sets the elapsed time (ET) of the timer, notifying subscribers on change.
- bool IN [get, set]
  Gets or sets the input (IN) state of the timer, notifying subscribers on change.
- bool Q [get, set]
  Gets or sets the output (Q) state of the timer, notifying subscribers on change.
- string Value [get, set]
  Gets or sets the display value of the timer, notifying subscribers on change.
- bool IsValid [get]
  Gets a value indicating whether the variable is valid based on the preset time.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]
  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]
  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]
  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- double _pt
  Stores the preset time (PT) of the timer.
- double _et
  Stores the elapsed time (ET) of the timer.
- bool _in
  Stores the input (IN) state of the timer.
- bool _q
  Stores the output (Q) state of the timer.
- string _value
  Stores the display value of the timer.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.40.1 Detailed Description

Represents a timer variable in a ladder diagram, encapsulating preset time, elapsed time, input, and output states.

Definition at line 6 of file TimerVariable.cs.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 TimerVariable()

ladder_diagram_app.Models.Variables.Instances.TimerVariable.TimerVariable () [inline]

Initializes a new instance of the TimerVariable class with default values.

Definition at line 81 of file TimerVariable.cs.

6.40.3 Member Function Documentation

6.40.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.TimerVariable.ToExportDictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 96 of file TimerVariable.cs.

6.40.4 Member Data Documentation

6.40.4.1 _et

double ladder_diagram_app.Models.Variables.Instances.TimerVariable._et [private]

Stores the elapsed time (ET) of the timer.

Definition at line 16 of file TimerVariable.cs.

6.40.4.2 _in

bool ladder_diagram_app.Models.Variables.Instances.TimerVariable._in [private]

Stores the input (IN) state of the timer.

Definition at line 21 of file TimerVariable.cs.

6.40.4.3 _pt

double ladder_diagram_app.Models.Variables.Instances.TimerVariable._pt [private]

Stores the preset time (PT) of the timer.

Definition at line 11 of file TimerVariable.cs.

6.40.4.4 _q

bool ladder_diagram_app.Models.Variables.Instances.TimerVariable._q [private]

Stores the output (Q) state of the timer.

Definition at line 26 of file TimerVariable.cs.

6.40.4.5 _value

string ladder_diagram_app.Models.Variables.Instances.TimerVariable._value [private]

Stores the display value of the timer.

Definition at line 31 of file TimerVariable.cs.

6.40.5 Property Documentation

6.40.5.1 ET

double ladder_diagram_app.Models.Variables.Instances.TimerVariable.ET [get], [set]

Gets or sets the elapsed time (ET) of the timer, notifying subscribers on change.

Definition at line 45 of file TimerVariable.cs.

6.40.5.2 IN

bool ladder_diagram_app.Models.Variables.Instances.TimerVariable.IN [get], [set]

Gets or sets the input (IN) state of the timer, notifying subscribers on change.

Definition at line 54 of file TimerVariable.cs.

6.40.5.3 IsValid

bool ladder_diagram_app.Models.Variables.Instances.TimerVariable.IsValid [get]

Gets a value indicating whether the variable is valid based on the preset time.

Definition at line 90 of file TimerVariable.cs.

6.40.5.4 PT

double ladder_diagram_app.Models.Variables.Instances.TimerVariable.PT [get], [set]

Gets or sets the preset time (PT) of the timer, notifying subscribers on change.

Definition at line 36 of file TimerVariable.cs.

6.40.5.5 Q

bool ladder_diagram_app.Models.Variables.Instances.TimerVariable.Q [get], [set]

Gets or sets the output (Q) state of the timer, notifying subscribers on change.

Definition at line 63 of file TimerVariable.cs.

6.40.5.6 Value

string ladder_diagram_app.Models.Variables.Instances.TimerVariable.Value [get], [set]

Gets or sets the display value of the timer, notifying subscribers on change.

Definition at line 72 of file TimerVariable.cs.

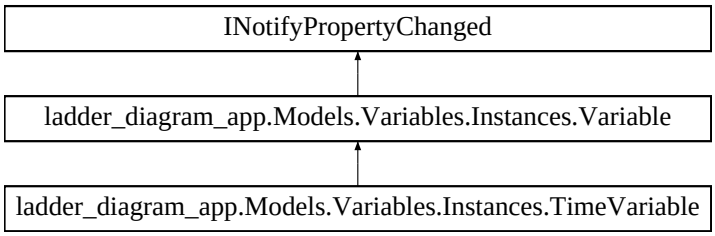
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/TimerVariable.cs

6.41 ladder_diagram_app.Models.Variables.Instances.TimeVariable Class Reference

Represents a time variable in a ladder diagram, encapsulating a double-precision time value.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.TimeVariable:



Public Member Functions

- TimeVariable ()
  Initializes a new instance of the TimeVariable class with default values.
- override Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Public Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Properties

- double Value [get, set]
  Gets or sets the time value of the variable, notifying subscribers on change.

Properties inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- string Name [get, set]
  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]
  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]
  Gets or sets a value indicating whether the variable can be deleted.

Private Attributes

- double _value
  Stores the time value of the variable.

Additional Inherited Members

Protected Member Functions inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Events inherited from ladder_diagram_app.Models.Variables.Instances.Variable

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

6.41.1 Detailed Description

Represents a time variable in a ladder diagram, encapsulating a double-precision time value.

Definition at line 6 of file TimeVariable.cs.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 TimeVariable()

ladder_diagram_app.Models.Variables.Instances.TimeVariable.TimeVariable () [inline]

Initializes a new instance of the TimeVariable class with default values.

Definition at line 25 of file TimeVariable.cs.

6.41.3 Member Function Documentation

6.41.3.1 ToExportDictionary()

override Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.TimeVariable.ToExportDictionary () [inline]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

Definition at line 35 of file TimeVariable.cs.

6.41.4 Member Data Documentation

6.41.4.1 _value

double ladder_diagram_app.Models.Variables.Instances.TimeVariable._value [private]

Stores the time value of the variable.

Definition at line 11 of file TimeVariable.cs.

6.41.5 Property Documentation

6.41.5.1 Value

double ladder_diagram_app.Models.Variables.Instances.TimeVariable.Value [get], [set]

Gets or sets the time value of the variable, notifying subscribers on change.

Definition at line 16 of file TimeVariable.cs.

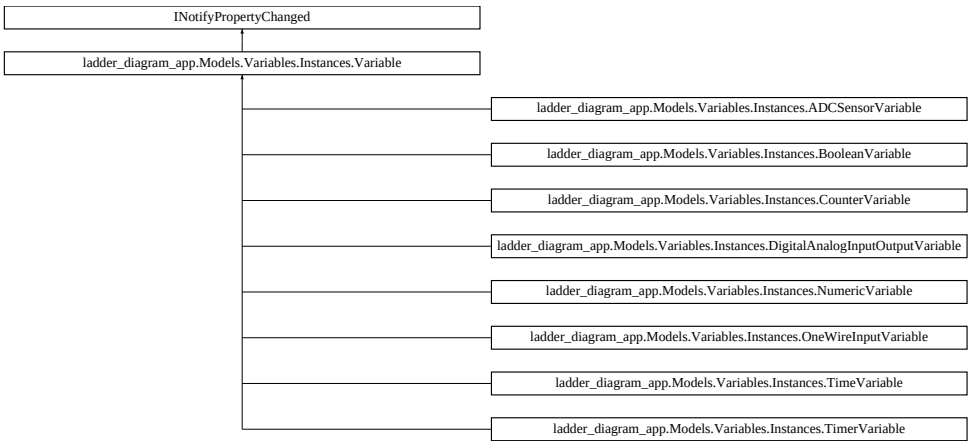
The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/TimeVariable.cs

6.42 ladder_diagram_app.Models.Variables.Instances.Variable Class Reference

Abstract base class for variables in a ladder diagram, providing common properties and change notification.

Inheritance diagram for ladder_diagram_app.Models.Variables.Instances.Variable:



Public Member Functions

- Dictionary< string, object > ToExportDictionary ()
  Converts the variable's properties to a dictionary for export purposes.

Protected Member Functions

- Variable ()
  Initializes a new instance of the Variable class with default empty values.
- virtual void OnPropertyChanged ([CallerMemberName] string? propertyName=null)
  Raises the PropertyChanged event for the specified property.
- bool SetField< T > (ref T field, T value, [CallerMemberName] string? propertyName=null)
  Updates a field and raises the PropertyChanged event if the value changes.

Properties

- string Name [get, set]
  Gets or sets the name of the variable, notifying subscribers on change.
- string Type [get, set]
  Gets or sets the type of the variable, notifying subscribers on change.
- bool IsDeletable = true [get, set]
  Gets or sets a value indicating whether the variable can be deleted.

Events

- PropertyChangedEventHandler? PropertyChanged
  Event raised when a property value changes.

Private Attributes

- string _name
  Stores the name of the variable.
- string _type
  Stores the type of the variable.

6.42.1 Detailed Description

Abstract base class for variables in a ladder diagram, providing common properties and change notification.

Definition at line 9 of file Variable.cs.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 Variable()

ladder_diagram_app.Models.Variables.Instances.Variable.Variable () [inline], [protected]

Initializes a new instance of the Variable class with default empty values.

Definition at line 24 of file Variable.cs.

6.42.3 Member Function Documentation

6.42.3.1 OnPropertyChanged()

virtual void ladder_diagram_app.Models.Variables.Instances.Variable.OnPropertyChanged (
[CallerMemberName] string? propertyName = null) [inline], [protected], [virtual]

Raises the PropertyChanged event for the specified property.

Parameters

property↩ Name	The name of the property that changed, automatically inferred if not specified.

Definition at line 62 of file Variable.cs.

6.42.3.2 SetField< T >()

bool ladder_diagram_app.Models.Variables.Instances.Variable.SetField< T > (
ref T field,
T value,
[CallerMemberName] string? propertyName = null) [inline], [protected]

Updates a field and raises the PropertyChanged event if the value changes.

Template Parameters

T	The type of the field.

Parameters

field	The backing field to update.
value	The new value for the field.
property↩ Name	The name of the property, automatically inferred if not specified.

Returns

True if the field was updated, false if the value was unchanged.

Definition at line 75 of file Variable.cs.

6.42.3.3 ToExportDictionary()

Dictionary< string, object > ladder_diagram_app.Models.Variables.Instances.Variable.ToExportDictionary () [abstract]

Converts the variable's properties to a dictionary for export purposes.

Returns

A dictionary containing the variable's properties and their values.

6.42.4 Member Data Documentation

6.42.4.1 _name

string ladder_diagram_app.Models.Variables.Instances.Variable._name [private]

Stores the name of the variable.

Definition at line 14 of file Variable.cs.

6.42.4.2 _type

string ladder_diagram_app.Models.Variables.Instances.Variable._type [private]

Stores the type of the variable.

Definition at line 19 of file Variable.cs.

6.42.5 Property Documentation

6.42.5.1 IsDeletable

bool ladder_diagram_app.Models.Variables.Instances.Variable.IsDeletable = true [get], [set]

Gets or sets a value indicating whether the variable can be deleted.

Definition at line 51 of file Variable.cs.

6.42.5.2 Name

string ladder_diagram_app.Models.Variables.Instances.Variable.Name [get], [set]

Gets or sets the name of the variable, notifying subscribers on change.

Definition at line 33 of file Variable.cs.

6.42.5.3 Type

string ladder_diagram_app.Models.Variables.Instances.Variable.Type [get], [set]

Gets or sets the type of the variable, notifying subscribers on change.

Definition at line 42 of file Variable.cs.

6.42.6 Event Documentation

6.42.6.1 PropertyChanged

PropertyChangedEventHandler? ladder_diagram_app.Models.Variables.Instances.Variable.PropertyChanged

Event raised when a property value changes.

Definition at line 56 of file Variable.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/Instances/Variable.cs

6.43 ladder_diagram_app.Models.Variables.VariablesManager Class Reference

Manages variables in a ladder diagram application, associating them with a device and maintaining lists for UI components.

Public Member Functions

- VariablesManager (Device device)

  Initializes a new instance of the VariablesManager class with a specified device.
- void ClearVariablesList ()

  Clears all variable lists.
- void AddVariable (string name, string type, Window owner, string pinName="", string sensorType="", string pdSck="", string dout="", string samplingRate="", double mapLow=0.0, double mapHigh=100.0, double gain=1.0, bool boolValue=false, double numValue=0.0, double pv=0.0, double cv=0.0, bool cu=true, bool cd=false, double pt=0.0, double et=0.0, double timeValue=0.0)

  Adds a new variable to the list with specified properties.
- void DeleteVariable (Variable variable, Window owner)

  Deletes a variable from the list, handling expanded properties if necessary.
- void VariableBooleanClick (Variable variable)

  Toggles boolean values or expands/collapses variable properties when clicked.
- void VariableTextBoxChange (Variable variable, string inputText)

  Updates the double value of a variable or its parameters based on text input.
- void VariableComboBoxChange (Variable variable, string selectedValue)

  Updates ComboBox values for variables or their parameters.
- bool ValidateVariables (Window owner)

  Validates all variables to ensure they meet export requirements.

Properties

- Device Device [get, set]

  Gets or sets the device associated with the variables.
- ObservableCollection< Variable > VariablesList = [] [get]

  Gets the collection of all variables.
- ObservableCollection< string > VariablesListContacts = [] [get]

  Gets the collection of variable names for contact elements in ladder diagrams.
- ObservableCollection< string > VariablesListCoils = [] [get]

  Gets the collection of variable names for coil elements in ladder diagrams.
- ObservableCollection< string > VariablesListMath = [] [get]

  Gets the collection of variable names for mathematical operations.
- ObservableCollection< string > VariablesListCompare = [] [get]

  Gets the collection of variable names for comparison operations.
- ObservableCollection< string > VariablesListCounter = [] [get]

  Gets the collection of variable names for counter operations.
- ObservableCollection< string > VariablesListTimer = [] [get]

  Gets the collection of variable names for timer operations.
- ObservableCollection< string > VariablesListReset = [] [get]

  Gets the collection of variable names for reset operations.

Private Member Functions

- void VariablesList_CollectionChanged (object? sender, NotifyCollectionChangedEventArgs e)
  Handles changes to the VariablesList collection, updating related collections.
- void AddVariableToCollections (Variable variable)
  Adds a variable to the appropriate collections based on its type.
- void RemoveVariableFromCollections (Variable variable)
  Removes a variable from the appropriate collections based on its type.

6.43.1 Detailed Description

Manages variables in a ladder diagram application, associating them with a device and maintaining lists for UI components.

Definition at line 15 of file VariablesManager.cs.

6.43.2 Constructor & Destructor Documentation

6.43.2.1 VariablesManager()

ladder_diagram_app.Models.Variables.VariablesManager.VariablesManager (
Device device) [inline]

Initializes a new instance of the VariablesManager class with a specified device.

Parameters

device	The device to associate with the variables.

Definition at line 66 of file VariablesManager.cs.

6.43.3 Member Function Documentation

6.43.3.1 AddVariable()

void ladder_diagram_app.Models.Variables.VariablesManager.AddVariable (
string name,
string type,
Window owner,
string pinName = "",
string sensorType = "",
string pdSck = "",
string dout = "",
string samplingRate = "",
double mapLow = 0::0,
double mapHigh = 100::0,
double gain = 1::0,
bool boolValue = false,
double numValue = 0::0,
double pv = 0::0,
double cv = 0::0,
bool cu = true,
bool cd = false,
double pt = 0::0,
double et = 0::0,
double timeValue = 0::0) [inline]

Adds a new variable to the list with specified properties.

Parameters

name	The name of the variable.
type	The type of the variable (e.g., Digital Input, Boolean).
owner	The owner window for displaying notifications.
pinName	The pin name for input/output variables.
sensorType	The sensor type for ADC sensors.
pdSck	The PD_SCK pin for ADC sensors.
dout	The DOUT pin for ADC sensors.
samplingRate	The sampling rate for ADC sensors.
mapLow	The low mapping value for ADC sensors.
mapHigh	The high mapping value for ADC sensors.
gain	The gain value for ADC sensors.
boolValue	The boolean value for Boolean variables.
numValue	The numeric value for Number variables.
pv	The preset value for Counter variables.
cv	The current value for Counter variables.
cu	The count-up flag for Counter variables.
cd	The count-down flag for Counter variables.
pt	The preset time for Timer variables.
et	The elapsed time for Timer variables.
timeValue	The time value for Time variables.

Definition at line 111 of file VariablesManager.cs.

6.43.3.2 AddVariableToCollections()

void ladder_diagram_app.Models.Variables.VariablesManager.AddVariableToCollections (
Variable variable) [inline], [private]

Adds a variable to the appropriate collections based on its type.

Parameters

variable	The variable to add.

Definition at line 573 of file VariablesManager.cs.

6.43.3.3 ClearVariablesList()

void ladder_diagram_app.Models.Variables.VariablesManager.ClearVariablesList () [inline]

Clears all variable lists.

Definition at line 76 of file VariablesManager.cs.

6.43.3.4 DeleteVariable()

void ladder_diagram_app.Models.Variables.VariablesManager.DeleteVariable (
Variable variable,
Window owner) [inline]

Deletes a variable from the list, handling expanded properties if necessary.

Parameters

variable	The variable to delete.
owner	The owner window for displaying notifications.

Definition at line 263 of file VariablesManager.cs.

6.43.3.5 RemoveVariableFromCollections()

void ladder_diagram_app.Models.Variables.VariablesManager.RemoveVariableFromCollections (
Variable variable) [inline], [private]

Removes a variable from the appropriate collections based on its type.

Parameters

variable	The variable to remove.

Definition at line 645 of file VariablesManager.cs.

6.43.3.6 ValidateVariables()

bool ladder_diagram_app.Models.Variables.VariablesManager.ValidateVariables (
Window owner) [inline]

Validates all variables to ensure they meet export requirements.

Parameters

owner	The owner window for displaying notifications.

Returns

True if all variables are valid, false otherwise.

Definition at line 719 of file VariablesManager.cs.

6.43.3.7 VariableBooleanClick()

void ladder_diagram_app.Models.Variables.VariablesManager.VariableBooleanClick (
Variable variable) [inline]

Toggles boolean values or expands/collapses variable properties when clicked.

Parameters

variable	The variable to modify.

Definition at line 300 of file VariablesManager.cs.

6.43.3.8 VariableComboBoxChange()

void ladder_diagram_app.Models.Variables.VariablesManager.VariableComboBoxChange (
Variable variable,
string selectedValue) [inline]

Updates ComboBox values for variables or their parameters.

Parameters

variable	The variable to update.
selectedValue	The selected ComboBox value.

Definition at line 482 of file VariablesManager.cs.

6.43.3.9 VariablesList_CollectionChanged()

void ladder_diagram_app.Models.Variables.VariablesManager.VariablesList_CollectionChanged (
object? sender,
NotifyCollectionChangedEventArgs e) [inline], [private]

Handles changes to the VariablesList collection, updating related collections.

Parameters

sender	The source of the event.
e	The event arguments.

Definition at line 545 of file VariablesManager.cs.

6.43.3.10 VariableTextBoxChange()

void ladder_diagram_app.Models.Variables.VariablesManager.VariableTextBoxChange (
Variable variable,
string inputText) [inline]

Updates the double value of a variable or its parameters based on text input.

Parameters

variable	The variable to update.
inputText	The text input to parse as a double.

Definition at line 402 of file VariablesManager.cs.

6.43.4 Property Documentation

6.43.4.1 Device

Device ladder_diagram_app.Models.Variables.VariablesManager.Device [get], [set]

Gets or sets the device associated with the variables.

Definition at line 20 of file VariablesManager.cs.

6.43.4.2 VariablesList

ObservableCollection<Variable> ladder_diagram_app.Models.Variables.VariablesManager.VariablesList = [] [get]

Gets the collection of all variables.

Definition at line 25 of file VariablesManager.cs.

6.43.4.3 VariablesListCoils

ObservableCollection<string> ladder_diagram_app.Models.Variables.VariablesManager.VariablesListCoils = [] [get]

Gets the collection of variable names for coil elements in ladder diagrams.

Definition at line 35 of file VariablesManager.cs.

6.43.4.4 VariablesListCompare

ObservableCollection<string> ladder_diagram_app.Models.Variables.VariablesManager.VariablesListCompare = [] [get]

Gets the collection of variable names for comparison operations.

Definition at line 45 of file VariablesManager.cs.

6.43.4.5 VariablesListContacts

ObservableCollection<string> ladder_diagram_app.Models.Variables.VariablesManager.VariablesListContacts = [] [get]

Gets the collection of variable names for contact elements in ladder diagrams.

Definition at line 30 of file VariablesManager.cs.

6.43.4.6 VariablesListCounter

ObservableCollection<string> ladder_diagram_app.Models.Variables.VariablesManager.VariablesListCounter = [] [get]

Gets the collection of variable names for counter operations.

Definition at line 50 of file VariablesManager.cs.

6.43.4.7 VariablesListMath

ObservableCollection<string> ladder_diagram_app.Models.Variables.VariablesManager.VariablesListMath = [] [get]

Gets the collection of variable names for mathematical operations.

Definition at line 40 of file VariablesManager.cs.

6.43.4.8 VariablesListReset

ObservableCollection<string> ladder_diagram_app.Models.Variables.VariablesManager.VariablesListReset = [] [get]

Gets the collection of variable names for reset operations.

Definition at line 60 of file VariablesManager.cs.

6.43.4.9 VariablesListTimer

ObservableCollection<string> ladder_diagram_app.Models.Variables.VariablesManager.VariablesListTimer = [] [get]

Gets the collection of variable names for timer operations.

Definition at line 55 of file VariablesManager.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/Variables/VariablesManager.cs

6.44 ladder_diagram_app.Models.CanvasElements.Instances.Wire Class Reference

Represents a wire in a ladder diagram, connecting nodes with a visual line.

Public Member Functions

- Wire ()
  Initializes a new instance of the Wire class with an empty node list and default line.
- void SelectWire ()
  Highlights the wire by setting its line to a red, thicker style.
- void UnselectWire ()
  Resets the wire's line to its default black style.
- void HighlightWire ()
  Highlights the wire with a blue, dashed, thicker style.
- void UnhighlightWire ()
  Resets the wire's line to its default black style, removing the dashed effect.

Properties

- double Width [get, set]
  Gets or sets the width of the wire.
- List< Node > Nodes [get, set]
  Gets or sets the list of nodes connected by the wire.
- Line WireLine [get, set]
  Gets or sets the line representing the wire visually.
- double Y [get, set]
  Gets or sets the Y-coordinate of the wire, updating the wire line when set.
- double Height [get]
  Gets the height of the wire, calculated based on the deepest branch Y2 value plus a margin.

Private Member Functions

- void UpdateWireLine ()

  Updates the coordinates of the wire line based on current Y and Width values.
- double GetMaxY2FromBranches (List< Node > nodes)

  Recursively calculates the maximum Y2 value from branches in the node list.

Private Attributes

- double _y

  Stores the Y-coordinate of the wire.

6.44.1 Detailed Description

Represents a wire in a ladder diagram, connecting nodes with a visual line.

Definition at line 9 of file Wire.cs.

6.44.2 Constructor & Destructor Documentation

6.44.2.1 Wire()

ladder_diagram_app.Models.CanvasElements.Instances.Wire.Wire () [inline]

Initializes a new instance of the Wire class with an empty node list and default line.

Definition at line 34 of file Wire.cs.

6.44.3 Member Function Documentation

6.44.3.1 GetMaxY2FromBranches()

double ladder_diagram_app.Models.CanvasElements.Instances.Wire.GetMaxY2FromBranches (
List< Node > nodes) [inline], [private]

Recursively calculates the maximum Y2 value from branches in the node list.

Parameters

nodes	The list of nodes to evaluate.

Returns

The maximum Y2 value, or the wire's Y if no branches are found.

Definition at line 88 of file Wire.cs.

6.44.3.2 HighlightWire()

void ladder_diagram_app.Models.CanvasElements.Instances.Wire.HighlightWire () [inline]

Highlights the wire with a blue, dashed, thicker style.

Definition at line 126 of file Wire.cs.

6.44.3.3 SelectWire()

void ladder_diagram_app.Models.CanvasElements.Instances.Wire.SelectWire () [inline]

Highlights the wire by setting its line to a red, thicker style.

Definition at line 108 of file Wire.cs.

6.44.3.4 UnhighlightWire()

void ladder_diagram_app.Models.CanvasElements.Instances.Wire.UnhighlightWire () [inline]

Resets the wire's line to its default black style, removing the dashed effect.

Definition at line 136 of file Wire.cs.

6.44.3.5 UnselectWire()

void ladder_diagram_app.Models.CanvasElements.Instances.Wire.UnselectWire () [inline]

Resets the wire's line to its default black style.

Definition at line 117 of file Wire.cs.

6.44.3.6 UpdateWireLine()

void ladder_diagram_app.Models.CanvasElements.Instances.Wire.UpdateWireLine () [inline], [private]

Updates the coordinates of the wire line based on current Y and Width values.

Definition at line 64 of file Wire.cs.

6.44.4 Member Data Documentation

6.44.4.1 _y

double ladder_diagram_app.Models.CanvasElements.Instances.Wire._y [private]

Stores the Y-coordinate of the wire.

Definition at line 14 of file Wire.cs.

6.44.5 Property Documentation

6.44.5.1 Height

double ladder_diagram_app.Models.CanvasElements.Instances.Wire.Height [get]

Gets the height of the wire, calculated based on the deepest branch Y2 value plus a margin.

Definition at line 75 of file Wire.cs.

6.44.5.2 Nodes

List<Node> ladder_diagram_app.Models.CanvasElements.Instances.Wire.Nodes [get], [set]

Gets or sets the list of nodes connected by the wire.

Definition at line 24 of file Wire.cs.

6.44.5.3 Width

double ladder_diagram_app.Models.CanvasElements.Instances.Wire.Width [get], [set]

Gets or sets the width of the wire.

Definition at line 19 of file Wire.cs.

6.44.5.4 WireLine

Line ladder_diagram_app.Models.CanvasElements.Instances.Wire.WireLine [get], [set]

Gets or sets the line representing the wire visually.

Definition at line 29 of file Wire.cs.

6.44.5.5 Y

double ladder_diagram_app.Models.CanvasElements.Instances.Wire.Y [get], [set]

Gets or sets the Y-coordinate of the wire, updating the wire line when set.

Definition at line 51 of file Wire.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/CanvasElements/Instances/Wire.cs

6.45 ladder_diagram_app.Models.CanvasElements.WiresManager Class Reference

Manages a collection of wires in a ladder diagram, providing methods to add, remove, insert, and clear wires.

Public Member Functions

- WiresManager ()
  Initializes a new instance of the WiresManager class with an empty wire list.
- void AddWire (Wire? wire=null)
  Adds a wire to the collection, creating a new wire if none is provided.
- void RemoveWire (Wire wire)
  Removes a specified wire from the collection.
- void InsertWire (Wire wire, int index)
  Inserts a wire at the specified index in the collection.
- void ClearWires ()
  Clears all wires from the collection.

Properties

- List< Wire > Wires [get]
  Gets the list of wires managed by this instance.

6.45.1 Detailed Description

Manages a collection of wires in a ladder diagram, providing methods to add, remove, insert, and clear wires.

Definition at line 8 of file WiresManager.cs.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 WiresManager()

ladder_diagram_app.Models.CanvasElements.WiresManager.WiresManager () [inline]

Initializes a new instance of the WiresManager class with an empty wire list.

Definition at line 18 of file WiresManager.cs.

6.45.3 Member Function Documentation

6.45.3.1 AddWire()

void ladder_diagram_app.Models.CanvasElements.WiresManager.AddWire (
Wire? wire = null) [inline]

Adds a wire to the collection, creating a new wire if none is provided.

Parameters

wire	The wire to add, or null to create a new wire.

Definition at line 27 of file WiresManager.cs.

6.45.3.2 ClearWires()

void ladder_diagram_app.Models.CanvasElements.WiresManager.ClearWires () [inline]

Clears all wires from the collection.

Definition at line 54 of file WiresManager.cs.

6.45.3.3 InsertWire()

void ladder_diagram_app.Models.CanvasElements.WiresManager.InsertWire (
Wire wire,
int index) [inline]

Inserts a wire at the specified index in the collection.

Parameters

wire	The wire to insert.
index	The zero-based index at which to insert the wire.

Definition at line 46 of file WiresManager.cs.

6.45.3.4 RemoveWire()

void ladder_diagram_app.Models.CanvasElements.WiresManager.RemoveWire (
Wire wire) [inline]

Removes a specified wire from the collection.

Parameters

wire	The wire to remove.

Definition at line 36 of file WiresManager.cs.

6.45.4 Property Documentation

6.45.4.1 Wires

List<Wire> ladder_diagram_app.Models.CanvasElements.WiresManager.Wires [get]

Gets the list of wires managed by this instance.

Definition at line 13 of file WiresManager.cs.

The documentation for this class was generated from the following file:

- ladder_diagram_app/Models/CanvasElements/WiresManager.cs

Chapter 7

File Documentation

7.1 ladder_diagram_app/App.xaml File Reference

7.2 App.xaml

Go to the documentation of this file.

```
00001 <Application x:Class="ladder_diagram_app.App"
00002              xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
00003              xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
00004              xmlns:local="clr-namespace:ladder_diagram_app"
00005              StartupUri="MainWindow.xaml">
00006     <Application.Resources>
00007
00008     </Application.Resources>
00009 </Application>
```

7.3 ladder_diagram_app/App.xaml.cs File Reference

Classes

- class ladder_diagram_app.App

  Interaction logic for App.xaml.

Namespaces

- namespace ladder_diagram_app

7.4 App.xaml.cs

Go to the documentation of this file.

```
00001 using System.Configuration;
00002 using System.Data;
00003 using System.Windows;
00004
00005 namespace ladder_diagram_app
00006 {
00010     public partial class App : Application
00011     {
00012     }
00013
00014 }
```

7.5 ladder_diagram_app/AssemblyInfo.cs File Reference

7.6 AssemblyInfo.cs

Go to the documentation of this file.

```
00001 using System.Windows;
00002
00003 [assembly: ThemeInfo(
00004     ResourceDictionaryLocation.None,           //where theme specific resource dictionaries are located
00005                                                //(used if a resource is not found in the page,
00006                                                // or application resource dictionaries)
00007     ResourceDictionaryLocation.SourceAssembly   //where the generic resource dictionary is located
00008                                                //(used if a resource is not found in the page,
00009                                                // app, or any theme specific resource dictionaries)
00010 )]
```

7.7 ladder_diagram_app/MainWindow.xaml File Reference

7.8 MainWindow.xaml

Go to the documentation of this file.

```
00001 <Window x:Class="ladder_diagram_app.MainWindow"
00002         x:Name="MainWindowControl"
00003         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
00004         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
00005         xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
00006         xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
00007         xmlns:local="clr-namespace:ladder_diagram_app"
00008         xmlns:views="clr-namespace:ladder_diagram_app.Views"
00009         xmlns:user_controls="clr-namespace:ladder_diagram_app.UserControls"
00010         mc:Ignorable="d"
00011         Title="Ladder Diagram Configurator" Height="500" Width="900" MinHeight="500" MinWidth="900">
00012     <Window.Resources>
00013         <ResourceDictionary>
00014             <Style x:Key="ModernButtonStyle" TargetType="Button">
00015                 <Setter Property="Background" Value="#E0E0E0"/>
00016                 <Setter Property="BorderBrush" Value="#B0B0B0"/>
00017                 <Setter Property="BorderThickness" Value="1"/>
00018                 <Setter Property="Padding" Value="5"/>
00019                 <Setter Property="FontSize" Value="12"/>
00020                 <Setter Property="Cursor" Value="Hand"/>
00021                 <Setter Property="Template">
00022                     <Setter.Value>
00023                         <ControlTemplate TargetType="Button">
00024                             <Border Background="{TemplateBinding Background}"
00025                                     BorderBrush="{TemplateBinding BorderBrush}"
00026                                     BorderThickness="{TemplateBinding BorderThickness}"
00027                                     CornerRadius="3">
00028                                 <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
00029                             </Border>
00030                             <ControlTemplate.Triggers>
00031                                 <Trigger Property="IsMouseOver" Value="True">
00032                                     <Setter Property="Background" Value="#D0D0D0"/>
00033                                 </Trigger>
00034                                 <Trigger Property="IsPressed" Value="True">
00035                                     <Setter Property="Background" Value="#C0C0C0"/>
00036                                 </Trigger>
00037                             </ControlTemplate.Triggers>
00038                         </ControlTemplate>
00039                     </Setter.Value>
00040                 </Setter>
00041             </Style>
00042
00043             <Style x:Key="SymbolButtonStyle" TargetType="Button">
00044                 <Setter Property="Background" Value="Transparent"/>
00045                 <Setter Property="BorderBrush" Value="#B0B0B0"/>
00046                 <Setter Property="BorderThickness" Value="1"/>
00047                 <Setter Property="Padding" Value="5"/>
00048                 <Setter Property="FontSize" Value="12"/>
00049                 <Setter Property="Cursor" Value="Hand"/>
00050                 <Setter Property="Template">
00051                     <Setter.Value>
```

```
00052                   <ControlTemplate TargetType="Button">
00053                      <Border x:Name="ButtonBorder"
00054                              Background="{TemplateBinding Background}"
00055                              BorderBrush="{TemplateBinding BorderBrush}"
00056                              BorderThickness="{TemplateBinding BorderThickness}"
00057                              CornerRadius="3">
00058                         <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
00059                      </Border>
00060                      <ControlTemplate.Triggers>
00061                         <Trigger Property="IsMouseOver" Value="True">
00062                            <Setter TargetName="ButtonBorder" Property="Background" Value="#E0E0E0"/>
00063                         </Trigger>
00064                         <Trigger Property="IsPressed" Value="True">
00065                            <Setter TargetName="ButtonBorder" Property="Background" Value="#D0D0D0"/>
00066                         </Trigger>
00067                      </ControlTemplate.Triggers>
00068                   </ControlTemplate>
00069                </Setter.Value>
00070             </Setter>
00071          </Style>
00072
00073          <Style x:Key="SymbolButtonStyle2" TargetType="Button">
00074             <Setter Property="Background" Value="Transparent"/>
00075             <Setter Property="BorderBrush" Value="#B0B0B0"/>
00076             <Setter Property="BorderThickness" Value="1"/>
00077             <Setter Property="Padding" Value="0,-4,0,0"/>
00078             <Setter Property="FontSize" Value="18"/>
00079             <Setter Property="Cursor" Value="Hand"/>
00080             <Setter Property="Template">
00081                <Setter.Value>
00082                   <ControlTemplate TargetType="Button">
00083                      <Border x:Name="ButtonBorder"
00084                              Background="{TemplateBinding Background}"
00085                              BorderBrush="{TemplateBinding BorderBrush}"
00086                              BorderThickness="{TemplateBinding BorderThickness}"
00087                              CornerRadius="3">
00088                         <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"
      Margin="{TemplateBinding Padding}"/>
00089                      </Border>
00090                      <ControlTemplate.Triggers>
00091                         <Trigger Property="IsMouseOver" Value="True">
00092                            <Setter TargetName="ButtonBorder" Property="Background" Value="#E0E0E0"/>
00093                         </Trigger>
00094                         <Trigger Property="IsPressed" Value="True">
00095                            <Setter TargetName="ButtonBorder" Property="Background" Value="Blue"/>
00096                         </Trigger>
00097                      </ControlTemplate.Triggers>
00098                   </ControlTemplate>
00099                </Setter.Value>
00100             </Setter>
00101          </Style>
00102
00103          <Style x:Key="VariableBooleanButtonStyle" TargetType="Button">
00104             <Setter Property="Background" Value="Transparent"/>
00105             <Setter Property="BorderBrush" Value="Transparent"/>
00106             <Setter Property="VerticalAlignment" Value="Center"/>
00107
00108             <EventSetter Event="Click" Handler="ButtonChangeBoolean_Click" />
00109          </Style>
00110
00111          <Style x:Key="VariableTextBoxStyle" TargetType="TextBox">
00112             <Setter Property="Background" Value="Transparent"/>
00113             <Setter Property="BorderBrush" Value="Transparent"/>
00114             <Setter Property="TextAlignment" Value="Center"/>
00115
00116             <EventSetter Event="TextBox.KeyDown" Handler="TextBoxVariable_TextChanged" />
00117             <EventSetter Event="TextBox.LostFocus" Handler="TextBoxVariable_TextChanged" />
00118             <EventSetter Event="TextBox.PreviewTextInput" Handler="TextBoxVariable_PreviewTextInput" />
00119          </Style>
00120
00121          <Style x:Key="VariableComboBoxStyle" TargetType="ComboBox">
00122             <EventSetter Event="SelectionChanged" Handler="ComboBoxVariable_SelectionChanged" />
00123          </Style>
00124
00125          <Style x:Key="VariableDeleteButtonStyle" TargetType="Button">
00126             <EventSetter Event="Click" Handler="ButtonDeleteVariable_Click" />
00127          </Style>
00128
00129          <Style TargetType="Expander">
00130             <Setter Property="Background" Value="#F0F0F0"/>
00131             <Setter Property="BorderBrush" Value="Gray"/>
00132             <Setter Property="BorderThickness" Value="1"/>
00133             <Setter Property="Padding" Value="10,2,0,2"/>
00134             <Setter Property="HeaderTemplate">
00135                <Setter.Value>
00136                   <DataTemplate>
00137                      <DockPanel LastChildFill="False" Height="30">
```

```
00138                         <TextBlock Text="{Binding}" FontWeight="Bold" VerticalAlignment="Center"
    DockPanel.Dock="Left"/>
00139                     </DockPanel>
00140                 </DataTemplate>
00141             </Setter.Value>
00142         </Setter>
00143     </Style>
00144
00145     <Style x:Key="GridSplitterStyle" TargetType="GridSplitter">
00146         <Setter Property="Visibility" Value="Visible"/>
00147         <Setter Property="IsEnabled" Value="True"/>
00148         <Style.Triggers>
00149             <DataTrigger Binding="{Binding Visibility, ElementName=MonitorGrid}" Value="Collapsed">
00150                 <Setter Property="Visibility" Value="Collapsed"/>
00151                 <Setter Property="IsEnabled" Value="False"/>
00152             </DataTrigger>
00153             <DataTrigger Binding="{Binding IsExpanded, ElementName=MonitorExpander}" Value="False">
00154                 <Setter Property="Visibility" Value="Collapsed"/>
00155                 <Setter Property="IsEnabled" Value="False"/>
00156             </DataTrigger>
00157         </Style.Triggers>
00158     </Style>
00159
00160     <Style x:Key="GridExpanderStyle" TargetType="RowDefinition">
00161         <Setter Property="MinHeight" Value="30"/>
00162         <Setter Property="Height" Value="100"/>
00163         <!-- Podrazumevana visina kada je vidljiv -->
00164         <Style.Triggers>
00165             <DataTrigger Binding="{Binding Visibility, ElementName=MonitorGrid}" Value="Collapsed">
00166                 <Setter Property="MinHeight" Value="0"/>
00167                 <Setter Property="Height" Value="Auto"/>
00168             </DataTrigger>
00169             <DataTrigger Binding="{Binding IsExpanded, ElementName=MonitorExpander}" Value="False">
00170                 <Setter Property="Height" Value="Auto"/>
00171                 <!-- Resetuj na Auto kada je kolapsovan -->
00172             </DataTrigger>
00173         </Style.Triggers>
00174     </Style>
00175
00176 </ResourceDictionary>
00177 </Window.Resources>
00178
00179 <Grid>
00180     <Grid.RowDefinitions>
00181         <RowDefinition Height="Auto"/>  <!-- Toolbar -->
00182         <RowDefinition Height="*"/>     <!-- Main Content -->
00183         <RowDefinition Height="5"/>     <!-- GridSplitter -->
00184         <RowDefinition Height="Auto" Style="{StaticResource GridExpanderStyle}"/>  <!-- MonitorGrid -->
00185     </Grid.RowDefinitions>
00186
00187     <!-- Toolbar -->
00188     <Grid Grid.Row="0" Background="#F5F5F5">
00189         <WrapPanel Orientation="Horizontal" Margin="10,5,10,10">
00190             <!-- Section 1: File -->
00191             <GroupBox Header="File" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3" BorderThickness="1">
00192                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00193                     <Button x:Name="ButtonImport" Content="Import" Width="70" Height="30" Margin="3"
    Style="{StaticResource ModernButtonStyle}"
00194                             Click="ButtonImport_Click"/>
00195                     <Button x:Name="ButtonExport" Content="Export" Width="70" Height="30" Margin="3"
    Style="{StaticResource ModernButtonStyle}"
00196                             Click="ButtonExport_Click"/>
00197                 </StackPanel>
00198             </GroupBox>
00199
00200             <!-- Section 1: Hardware -->
00201             <GroupBox Header="Hardware" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3"
    BorderThickness="1">
00202                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00203                     <Button  x:Name="ButtonDeviceInfo" Content="Info" Width="70" Height="30" Margin="3"
    Style="{StaticResource ModernButtonStyle}"
00204                             Click="ButtonDeviceInfo_Click"/>
00205                 </StackPanel>
00206             </GroupBox>
00207
00208             <!-- Section 2: Load Device -->
00209             <GroupBox Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3" BorderThickness="1">
00210                 <GroupBox.Header>
00211                     <TextBlock>
00212                         <Run Text="Device - " Foreground="Black"/>
00213                         <Run x:Name="StatusRun" Text="Not Connected" Foreground="Red"/>
00214                     </TextBlock>
00215                 </GroupBox.Header>
00216                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00217                     <StackPanel Margin="0,4,8,0">
00218                         <RadioButton x:Name="ConnectionMQTT" Content="MQTT" IsChecked="True"
    GroupName="Options"/>
```

```
00219                         <RadioButton x:Name="ConnectionBLE" Content="BLE" GroupName="Options"/>
00220                     </StackPanel>
00221                     <Button x:Name="ButtonConnect" Content="Connect" Width="70" Height="30" Margin="3"
    Style="{StaticResource ModernButtonStyle}"
00222                             Click="ButtonConnect_Click"/>
00223                     <Button x:Name="ButtonDisconnect" Content="Disconnect" Width="70" Height="30" Margin="3"
    Style="{StaticResource ModernButtonStyle}"
00224                             Click="ButtonDisconnect_Click"/>
00225                     <Button x:Name="ButtonSendToDevice" Content="Send to Device" Width="100" Height="30"
    Margin="3" Style="{StaticResource ModernButtonStyle}"
00226                             Click="ButtonSendToDevice_Click"/>
00227                 </StackPanel>
00228             </GroupBox>
00229
00230             <!-- Section 2: Device Parents -->
00231             <GroupBox Header="Parent(s)" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3"
    BorderThickness="1">
00232                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00233                     <Button x:Name="ButtonParentDevice" Content=" Add Device(s)" Width="100" Height="30"
    Margin="3" Style="{StaticResource ModernButtonStyle}"
00234                             Click="ButtonParentDevice_Click"/>
00235                 </StackPanel>
00236             </GroupBox>
00237
00238             <!-- Section 4: Layout -->
00239             <GroupBox Header="Layout" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3"
    BorderThickness="1">
00240                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00241                     <Button x:Name="ButtonAddWire" Content="Add Wire" Width="80" Height="30" Margin="3"
    Style="{StaticResource ModernButtonStyle}"
00242                             Click="ButtonAddWire_Click"/>
00243                     <Button x:Name="ButtonDelete" Content="Delete" Width="80" Height="30" Margin="3"
    Style="{StaticResource ModernButtonStyle}"
00244                             Click="ButtonDelete_Click"/>
00245                     <Button x:Name="ButtonBranch" Margin="3" ToolTip="Branch" Cursor="Hand"
    Background="Transparent"
00246                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="Branch"
    Width="50" Height="30"
00247                             Style="{StaticResource SymbolButtonStyle}">
00248                         <Image Source="/Resources/Branch/branch_icon.png" Width="40" Height="20"/>
00249                     </Button>
00250                 </StackPanel>
00251             </GroupBox>
00252
00253             <!-- Section 5: Contacts -->
00254             <GroupBox Header="Contacts" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3"
    BorderThickness="1">
00255                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00256                     <Button x:Name="ButtonNOContact" Margin="3" ToolTip="NO Contact" Cursor="Hand"
    Background="Transparent"
00257                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="NOContact"
    Width="50" Height="30"
00258                             Style="{StaticResource SymbolButtonStyle}">
00259                         <Image Source="/Resources/Contacts/no_contact.png" Width="40" Height="20"/>
00260                     </Button>
00261                     <Button x:Name="ButtonNCContact" Margin="3" ToolTip="NC Contact" Cursor="Hand"
    Background="Transparent"
00262                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="NCContact"
    Width="50" Height="30"
00263                             Style="{StaticResource SymbolButtonStyle}">
00264                         <Image Source="/Resources/Contacts/nc_contact.png" Width="40" Height="20"/>
00265                     </Button>
00266                 </StackPanel>
00267             </GroupBox>
00268
00269             <!-- Section 6: Coils -->
00270             <GroupBox Header="Coils" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3" BorderThickness="1">
00271                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00272                     <Button x:Name="ButtonCoil" Margin="3" ToolTip="Coil" Cursor="Hand" Background="Transparent"
00273                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="Coil"
    Width="50" Height="30"
00274                             Style="{StaticResource SymbolButtonStyle}">
00275                         <Image Source="/Resources/Coils/coil.png" Width="40" Height="20"/>
00276                     </Button>
00277                     <Button x:Name="ButtonOneShotPositiveCoil" Margin="3" ToolTip="One Shot Positive Coil"
    Cursor="Hand" Background="Transparent"
00278                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="OSPCoil"
    Width="50" Height="30"
00279                             Style="{StaticResource SymbolButtonStyle}">
00280                         <Image Source="/Resources/Coils/one_shot_positive_coil.png" Width="40" Height="20"/>
00281                     </Button>
00282                     <Button x:Name="ButtonSetCoil" Margin="3" ToolTip="Set Coil" Cursor="Hand"
    Background="Transparent"
00283                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="SetCoil"
    Width="50" Height="30"
00284                             Style="{StaticResource SymbolButtonStyle}">
00285                         <Image Source="/Resources/Coils/set_coil.png" Width="40" Height="20"/>
```

```
00286                     </Button>
00287                     <Button x:Name="ButtonResetCoil" Margin="3" ToolTip="Reset Coil" Cursor="Hand"
    Background="Transparent"
00288                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="ResetCoil"
    Width="50" Height="30"
00289                             Style="{StaticResource SymbolButtonStyle}">
00290                         <Image Source="/Resources/Coils/reset_coil.png" Width="40" Height="20"/>
00291                     </Button>
00292                 </StackPanel>
00293             </GroupBox>
00294 
00295             <!-- Section 7: Math -->
00296             <GroupBox Header="Math" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3"
    BorderThickness="1">
00297                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00298                     <Button x:Name="ButtonAdd" Margin="3" ToolTip="Add" Cursor="Hand" Background="Transparent"
00299                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="AddMath"
    Width="50" Height="30"
00300                             Content="+" Style="{StaticResource SymbolButtonStyle2}"/>
00301                     <Button x:Name="ButtonSubtract" Margin="3" ToolTip="Subtract" Cursor="Hand"
    Background="Transparent"
00302                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="SubtractMath"
    Width="50" Height="30"
00303                             Content="-" Style="{StaticResource SymbolButtonStyle2}"/>
00304                     <Button x:Name="ButtonMultiply" Margin="3" ToolTip="Multiply" Cursor="Hand"
    Background="Transparent"
00305                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="MultiplyMath"
    Width="50" Height="30"
00306                             Content="×" Style="{StaticResource SymbolButtonStyle2}"/>
00307                     <Button x:Name="ButtonDivide" Margin="3" ToolTip="Divide" Cursor="Hand"
    Background="Transparent"
00308                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="DivideMath"
    Width="50" Height="30"
00309                             Content="÷" Style="{StaticResource SymbolButtonStyle2}"/>
00310                     <Button x:Name="ButtonMove" Margin="3" ToolTip="Move" Cursor="Hand"
    Background="Transparent"
00311                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="MoveMath"
    Width="50" Height="30"
00312                             Content="MOVE" Style="{StaticResource SymbolButtonStyle}"/>
00313                 </StackPanel>
00314             </GroupBox>
00315 
00316             <!-- Section 8: Compare -->
00317             <GroupBox Header="Compare" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3"
    BorderThickness="1">
00318                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00319                     <Button x:Name="ButtonGreater" Margin="3" ToolTip="Greater" Cursor="Hand"
    Background="Transparent"
00320                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="GreaterCompare"
    Width="50" Height="30"
00321                             Content=">" Style="{StaticResource SymbolButtonStyle2}"/>
00322                     <Button x:Name="ButtonLess" Margin="3" ToolTip="Less" Cursor="Hand" Background="Transparent"
00323                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="LessCompare"
    Width="50" Height="30"
00324                             Content="&amp;lt;"  Style="{StaticResource SymbolButtonStyle2}"/>
00325                     <Button x:Name="ButtonGreaterOrEqual" Margin="3" ToolTip="Greater or Equal" Cursor="Hand"
    Background="Transparent"
00326                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown"
    Tag="GreaterOrEqualCompare" Width="50" Height="30"
00327                             Content=" " Style="{StaticResource SymbolButtonStyle2}"/>
00328                     <Button x:Name="ButtonLessOrEqual" Margin="3" ToolTip="Less or Equal" Cursor="Hand"
    Background="Transparent"
00329                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown"
    Tag="LessOrEqualCompare" Width="50" Height="30"
00330                             Content=" " Style="{StaticResource SymbolButtonStyle2}"/>
00331                     <Button x:Name="ButtonEqual" Margin="3" ToolTip="Equal" Cursor="Hand"
    Background="Transparent"
00332                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="EqualCompare"
    Width="50" Height="30"
00333                             Content="=" Style="{StaticResource SymbolButtonStyle2}"/>
00334                     <Button x:Name="ButtonNotEqual" Margin="3" ToolTip="Not Equal" Cursor="Hand"
    Background="Transparent"
00335                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown"
    Tag="NotEqualCompare" Width="50" Height="30"
00336                             Content=" " Style="{StaticResource SymbolButtonStyle2}"/>
00337                 </StackPanel>
00338             </GroupBox>
00339 
00340             <!-- Section 9: Time/Count -->
00341             <GroupBox Header="Time/Count" Margin="2,2,2,0" Padding="5" BorderBrush="#D3D3D3"
    BorderThickness="1">
00342                 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00343                     <Button x:Name="ButtonOnDelayTimer" Margin="3" ToolTip="On Delay Timer" Cursor="Hand"
    Background="Transparent"
00344                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="OnDelayTimer"
    Width="50" Height="30"
00345                             Content="TON" Style="{StaticResource SymbolButtonStyle}"/>
```

```
00346                     <Button x:Name="ButtonOfDelayTimer" Margin="3" ToolTip="Off Delay Timer" Cursor="Hand"
    Background="Transparent"
00347                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="OffDelayTimer"
    Width="50" Height="30"
00348                             Content="TOFF" Style="{StaticResource SymbolButtonStyle}"/>
00349                     <Button x:Name="ButtonCountUp" Margin="3" ToolTip="Count Up" Cursor="Hand"
    Background="Transparent"
00350                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="CountUp"
    Width="50" Height="30"
00351                             Content="CTU" Style="{StaticResource SymbolButtonStyle}"/>
00352                     <Button x:Name="ButtonCountDown" Margin="3" ToolTip="Count Down" Cursor="Hand"
    Background="Transparent"
00353                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="CountDown"
    Width="50" Height="30"
00354                             Content="CTD" Style="{StaticResource SymbolButtonStyle}"/>
00355                     <Button x:Name="ButtonReset" Margin="3" ToolTip="Reset" Cursor="Hand"
    Background="Transparent"
00356                             PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown" Tag="Reset"
    Width="50" Height="30"
00357                             Content="RESET" Style="{StaticResource SymbolButtonStyle}"/>
00358                 </StackPanel>
00359             </GroupBox>
00360         </WrapPanel>
00361     </Grid>
00362
00363     <!-- Main Content -->
00364     <Grid Grid.Row="1">
00365         <Grid.ColumnDefinitions>
00366             <ColumnDefinition Width="410"/>
00367             <ColumnDefinition Width="*"/>
00368         </Grid.ColumnDefinitions>
00369
00370         <Grid x:Name="GridVariables" Grid.Column="0" Background="LightGray">
00371
00372             <Grid.RowDefinitions>
00373                 <RowDefinition Height="Auto"/>
00374                 <RowDefinition Height="*"/>
00375             </Grid.RowDefinitions>
00376
00377             <StackPanel Orientation="Horizontal" Margin="10,10,10,0" Grid.Row="0">
00378                 <TextBox x:Name="NameTextBox" Width="170" Margin="0,0,10,0" MaxLength="61"/>
00379                 <ComboBox x:Name="TypeComboBox" Width="120" Margin="0,0,10,0" SelectedIndex="0">
00380                     <ComboBoxItem Content="Digital Input"/>
00381                     <ComboBoxItem Content="Digital Output"/>
00382                     <ComboBoxItem Content="Analog Input"/>
00383                     <ComboBoxItem Content="Analog Output"/>
00384                     <ComboBoxItem Content="One Wire Input"/>
00385                     <ComboBoxItem Content="ADC Sensor"/>
00386                     <ComboBoxItem Content="Boolean"/>
00387                     <ComboBoxItem Content="Number"/>
00388                     <ComboBoxItem Content="Counter"/>
00389                     <ComboBoxItem Content="Timer"/>
00390                     <ComboBoxItem Content="Current Time"/>
00391                     <ComboBoxItem Content="Time"/>
00392                 </ComboBox>
00393                 <Button Content="Add" Width="80" Click="ButtonAddVariable_Click"/>
00394             </StackPanel>
00395
00396             <ListView x:Name="ElementListView" Grid.Row="1" Margin="10" ItemsSource="{Binding
    VariablesManager.VariablesList}" VerticalAlignment="Stretch">
00397                 <ListView.View>
00398                     <GridView>
00399                         <!-- Name Column -->
00400                         <GridViewColumn Header="Name" Width="125">
00401                             <GridViewColumn.CellTemplate>
00402                                 <DataTemplate>
00403                                     <TextBlock Text="{Binding Name}">
00404                                         <TextBlock.Style>
00405                                             <Style TargetType="TextBlock">
00406                                                 <Style.Triggers>
00407                                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
    Value="    PD_SCK">
00408                                                         <Setter Property="ToolTip" Value="Clock Pin" />
00409                                                     </DataTrigger>
00410                                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
    Value="    DOUT">
00411                                                         <Setter Property="ToolTip" Value="Data Output Pin" />
00412                                                     </DataTrigger>
00413                                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
    Value="    PV">
00414                                                         <Setter Property="ToolTip" Value="Preset Value" />
00415                                                     </DataTrigger>
00416                                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
    Value="    CV">
00417                                                         <Setter Property="ToolTip" Value="Current Value" />
00418                                                     </DataTrigger>
00419                                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
```

```
Value="CU">
00420                                         <Setter Property="ToolTip" Value="Count Up" />
00421                                     </DataTrigger>
00422                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
Value="CD">
00423                                         <Setter Property="ToolTip" Value="Count Down" />
00424                                     </DataTrigger>
00425                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
Value="QU">
00426                                         <Setter Property="ToolTip" Value="Count Up Output" />
00427                                     </DataTrigger>
00428                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
Value="QD">
00429                                         <Setter Property="ToolTip" Value="Count Down Output" />
00430                                     </DataTrigger>
00431                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
Value="PT">
00432                                         <Setter Property="ToolTip" Value="Preset Time" />
00433                                     </DataTrigger>
00434                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
Value="ET">
00435                                         <Setter Property="ToolTip" Value="Elapsed Time" />
00436                                     </DataTrigger>
00437                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
Value="IN">
00438                                         <Setter Property="ToolTip" Value="Timer Input" />
00439                                     </DataTrigger>
00440                                     <DataTrigger Binding="{Binding Text, RelativeSource={RelativeSource Self}}"
Value="Q">
00441                                         <Setter Property="ToolTip" Value="Timer Output" />
00442                                     </DataTrigger>
00443                                 </Style.Triggers>
00444                             </Style>
00445                         </TextBlock.Style>
00446                     </TextBlock>
00447                 </DataTemplate>
00448             </GridViewColumn.CellTemplate>
00449         </GridViewColumn>
00450
00451         <!-- Type Columnd -->
00452         <GridViewColumn Header="Type" Width="100">
00453             <GridViewColumn.CellTemplate>
00454                 <DataTemplate>
00455                     <TextBlock Text="{Binding Type}"/>
00456                 </DataTemplate>
00457             </GridViewColumn.CellTemplate>
00458         </GridViewColumn>
00459
00460         <!-- Value Column -->
00461         <GridViewColumn Header="Value" Width="100">
00462             <GridViewColumn.CellTemplate>
00463                 <DataTemplate>
00464                     <ContentControl>
00465                         <ContentControl.Style>
00466                             <Style TargetType="ContentControl">
00467                                 <Style.Triggers>
00468                                     <!-- Digital input -->
00469                                     <DataTrigger Binding="{Binding Type}" Value="Digital Input">
00470                                         <Setter Property="Content">
00471                                             <Setter.Value>
00472                                                 <ComboBox Width="80" ItemsSource="{Binding
DataContext.DevicePinManager.DigitalInputOptions, ElementName=MainWindowControl}"
00473                                                           SelectedItem="{Binding PinName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
00474                                                           Style="{StaticResource VariableComboBoxStyle}"/>
00475                                             </Setter.Value>
00476                                         </Setter>
00477                                     </DataTrigger>
00478                                     <!-- Digital output -->
00479                                     <DataTrigger Binding="{Binding Type}" Value="Digital Output">
00480                                         <Setter Property="Content">
00481                                             <Setter.Value>
00482                                                 <ComboBox Width="80" ItemsSource="{Binding
DataContext.DevicePinManager.DigitalOutputOptions, ElementName=MainWindowControl}"
00483                                                           SelectedItem="{Binding PinName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
00484                                                           Style="{StaticResource VariableComboBoxStyle}"/>
00485                                             </Setter.Value>
00486                                         </Setter>
00487                                     </DataTrigger>
00488                                     <!-- Analog input -->
00489                                     <DataTrigger Binding="{Binding Type}" Value="Analog Input">
00490                                         <Setter Property="Content">
00491                                             <Setter.Value>
00492                                                 <ComboBox Width="80" ItemsSource="{Binding
DataContext.DevicePinManager.AnalogInputOptions, ElementName=MainWindowControl}"
00493                                                           SelectedItem="{Binding PinName, Mode=TwoWay,
```

```
    UpdateSourceTrigger=PropertyChanged}"
00494                                                   Style="{StaticResource VariableComboBoxStyle}"/>
00495                                           </Setter.Value>
00496                                       </Setter>
00497                                   </DataTrigger>
00498                                   <!-- Analog output -->
00499                                   <DataTrigger Binding="{Binding Type}" Value="Analog Output">
00500                                       <Setter Property="Content">
00501                                           <Setter.Value>
00502                                               <ComboBox Width="80" ItemsSource="{Binding
    DataContext.DevicePinManager.AnalogOutputOptions, ElementName=MainWindowControl}"
00503                                                   SelectedItem="{Binding PinName, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
00504                                                   Style="{StaticResource VariableComboBoxStyle}"/>
00505                                           </Setter.Value>
00506                                       </Setter>
00507                                   </DataTrigger>
00508                                   <!-- One Wire Input -->
00509                                   <DataTrigger Binding="{Binding Type}" Value="One Wire Input">
00510                                       <Setter Property="Content">
00511                                           <Setter.Value>
00512                                               <ComboBox Width="80" ItemsSource="{Binding
    DataContext.DevicePinManager.OneWireInputOptions, ElementName=MainWindowControl}"
00513                                                   SelectedItem="{Binding PinName, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
00514                                                   Style="{StaticResource VariableComboBoxStyle}"/>
00515                                           </Setter.Value>
00516                                       </Setter>
00517                                   </DataTrigger>
00518                                   <!-- ADC Sensor -->
00519                                   <DataTrigger Binding="{Binding Type}" Value="ADC Sensor">
00520                                       <Setter Property="Content">
00521                                           <Setter.Value>
00522                                               <Button Content="{Binding Value}" Width="80"
    Style="{StaticResource VariableBooleanButtonStyle}"/>
00523                                           </Setter.Value>
00524                                       </Setter>
00525                                   </DataTrigger>
00526                                   <!-- ADC Sensor Type -->
00527                                   <DataTrigger Binding="{Binding Type}" Value="ADC Sensor Type">
00528                                       <Setter Property="Content">
00529                                           <Setter.Value>
00530                                               <ComboBox Width="80" ItemsSource="{Binding
    DataContext.AdcSensorTypes, ElementName=MainWindowControl}"
00531                                                   SelectedItem="{Binding PinName, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
00532                                                   Style="{StaticResource VariableComboBoxStyle}"/>
00533                                           </Setter.Value>
00534                                       </Setter>
00535                                   </DataTrigger>
00536                                   <!-- ADC Sensor Digital Input -->
00537                                   <DataTrigger Binding="{Binding Type}" Value="ADC Sensor Digital Input">
00538                                       <Setter Property="Content">
00539                                           <Setter.Value>
00540                                               <ComboBox Width="80" ItemsSource="{Binding
    DataContext.DevicePinManager.DigitalInputOptions, ElementName=MainWindowControl}"
00541                                                   SelectedItem="{Binding PinName, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
00542                                                   Style="{StaticResource VariableComboBoxStyle}"/>
00543                                           </Setter.Value>
00544                                       </Setter>
00545                                   </DataTrigger>
00546                                   <!-- ADC Sensor Digital output -->
00547                                   <DataTrigger Binding="{Binding Type}" Value="ADC Sensor Digital Output">
00548                                       <Setter Property="Content">
00549                                           <Setter.Value>
00550                                               <ComboBox Width="80" ItemsSource="{Binding
    DataContext.DevicePinManager.DigitalOutputOptions, ElementName=MainWindowControl}"
00551                                                   SelectedItem="{Binding PinName, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
00552                                                   Style="{StaticResource VariableComboBoxStyle}"/>
00553                                           </Setter.Value>
00554                                       </Setter>
00555                                   </DataTrigger>
00556                                   <!-- ADC Sampling Rate -->
00557                                   <DataTrigger Binding="{Binding Type}" Value="ADC Sensor Sampling Rate">
00558                                       <Setter Property="Content">
00559                                           <Setter.Value>
00560                                               <ComboBox Width="80" ItemsSource="{Binding
    DataContext.AdcSensorSamplingRates, ElementName=MainWindowControl}"
00561                                                   SelectedItem="{Binding PinName, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
00562                                                   Style="{StaticResource VariableComboBoxStyle}"/>
00563                                           </Setter.Value>
00564                                       </Setter>
00565                                   </DataTrigger>
00566                                   <!-- Boolean -->
```

```
00567                                       <DataTrigger Binding="{Binding Type}" Value="Boolean">
00568                                          <Setter Property="Content">
00569                                             <Setter.Value>
00570                                                <Button Content="{Binding Value}" Width="80"
      Style="{StaticResource VariableBooleanButtonStyle}"/>
00571                                             </Setter.Value>
00572                                          </Setter>
00573                                       </DataTrigger>
00574                                       <!-- Number -->
00575                                       <DataTrigger Binding="{Binding Type}" Value="Number">
00576                                          <Setter Property="Content">
00577                                             <Setter.Value>
00578                                              <TextBox Text="{Binding Value}" Width="80" Style="{StaticResource
      VariableTextBoxStyle}"/>
00579                                             </Setter.Value>
00580                                          </Setter>
00581                                       </DataTrigger>
00582                                       <!-- Counter -->
00583                                       <DataTrigger Binding="{Binding Type}" Value="Counter">
00584                                          <Setter Property="Content">
00585                                             <Setter.Value>
00586                                                <Button Content="{Binding Value}" Width="80"
      Style="{StaticResource VariableBooleanButtonStyle}"/>
00587                                             </Setter.Value>
00588                                          </Setter>
00589                                       </DataTrigger>
00590                                       <!-- Timer -->
00591                                       <DataTrigger Binding="{Binding Type}" Value="Timer">
00592                                          <Setter Property="Content">
00593                                             <Setter.Value>
00594                                                <Button Content="{Binding Value}" Width="80"
      Style="{StaticResource VariableBooleanButtonStyle}"/>
00595                                             </Setter.Value>
00596                                          </Setter>
00597                                       </DataTrigger>
00598                                       <!-- Current Time -->
00599                                       <DataTrigger Binding="{Binding Type}" Value="Current Time">
00600                                          <Setter Property="Content">
00601                                             <Setter.Value>
00602                                                <TextBlock Text=""/>
00603                                             </Setter.Value>
00604                                          </Setter>
00605                                       </DataTrigger>
00606                                       <!-- Time -->
00607                                       <DataTrigger Binding="{Binding Type}" Value="Time">
00608                                          <Setter Property="Content">
00609                                             <Setter.Value>
00610                                                <user_controls:TimePicker SelectedTime="{Binding Value,
      UpdateSourceTrigger=PropertyChanged}" Width="80"/>
00611                                             </Setter.Value>
00612                                          </Setter>
00613                                       </DataTrigger>
00614                                    </Style.Triggers>
00615                                 </Style>
00616                              </ContentControl.Style>
00617                           </ContentControl>
00618                        </DataTemplate>
00619                     </GridViewColumn.CellTemplate>
00620                  </GridViewColumn>
00621
00622                  <!-- Delete Column -->
00623                  <GridViewColumn Header="" Width="35">
00624                     <GridViewColumn.CellTemplate>
00625                        <DataTemplate>
00626                           <ContentControl>
00627                              <ContentControl.Style>
00628                                 <Style TargetType="ContentControl">
00629                                    <Style.Triggers>
00630                                       <!-- If Element is Deleatable -->
00631                                       <DataTrigger Binding="{Binding IsDeletable}" Value="True">
00632                                          <Setter Property="Content">
00633                                             <Setter.Value>
00634                                                <Button Content="X" Width="20" Style="{StaticResource
      VariableDeleteButtonStyle}"/>
00635                                             </Setter.Value>
00636                                          </Setter>
00637                                       </DataTrigger>
00638                                    </Style.Triggers>
00639                                 </Style>
00640                              </ContentControl.Style>
00641                           </ContentControl>
00642                        </DataTemplate>
00643                     </GridViewColumn.CellTemplate>
00644                  </GridViewColumn>
00645               </GridView>
00646            </ListView.View>
00647         </ListView>
```

```
00648           </Grid>
00649
00650           <Grid Grid.Column="1" x:Name="GridCanvas">
00651               <ScrollViewer x:Name="ScrollViewerCanvas" HorizontalScrollBarVisibility="Auto"
    VerticalScrollBarVisibility="Auto">
00652                   <Canvas x:Name="MainCanvas" Background="White"
00653                           AllowDrop="True" Drop="Canvas_Drop" DragOver="Canvas_DragOver"
00654                           MouseLeftButtonDown="MainCanvas_MouseLeftButtonDown"
00655                           MouseMove="MainCanvas_MouseMove"
00656                           MouseLeftButtonUp="MainCanvas_MouseLeftButtonUp"
00657                           RenderTransformOrigin="0, 0">
00658                       <Canvas.RenderTransform>
00659                           <ScaleTransform x:Name="CanvasScaleTransform" ScaleX="1" ScaleY="1"/>
00660                       </Canvas.RenderTransform>
00661                   </Canvas>
00662               </ScrollViewer>
00663           </Grid>
00664       </Grid>
00665
00666       <!-- GridSplitter -->
00667       <GridSplitter Grid.Row="2" Height="5" Background="Transparent" HorizontalAlignment="Stretch"
    VerticalAlignment="Stretch" Style="{StaticResource GridSplitterStyle}"/>
00668
00669       <!-- MonitorGrid -->
00670       <Grid Grid.Row="3" x:Name="MonitorGrid" Visibility="Collapsed">
00671           <Grid.RowDefinitions>
00672               <RowDefinition Height="*"/> <!-- Expander -->
00673           </Grid.RowDefinitions>
00674           <Expander Grid.Row="0" x:Name="MonitorExpander" Header="Live Monitoring" IsExpanded="False"
    ExpandDirection="Down"
00675                     Collapsed="MonitorExpander_Collapsed">
00676
00677               <Grid>
00678                   <Grid.ColumnDefinitions>
00679                       <ColumnDefinition Width="1*" MinWidth="200"/>   <!-- Monitor Content -->
00680                       <ColumnDefinition Width="5"/>                   <!-- Resize handle -->
00681                       <ColumnDefinition Width="1*" MinWidth="200"/>   <!-- OneWire Content -->
00682                   </Grid.ColumnDefinitions>
00683
00684                   <ScrollViewer Grid.Column="0" VerticalScrollBarVisibility="Auto"
    HorizontalScrollBarVisibility="Disabled" Height="{Binding ActualHeight, RelativeSource={RelativeSource
    AncestorType=Grid}}">
00685                       <TextBlock x:Name="MonitorTextBlock" TextWrapping="Wrap"/>
00686                   </ScrollViewer>
00687
00688                   <GridSplitter Grid.Column="1" Width="5" Background="Transparent" HorizontalAlignment="Stretch"
    VerticalAlignment="Stretch" />
00689
00690                   <ScrollViewer Grid.Column="2" x:Name="OneWireContent" VerticalScrollBarVisibility="Auto">
00691                       <ListView x:Name="OneWireItemsControl">
00692                           <ListView.View>
00693                               <GridView>
00694                                   <GridViewColumn Header="Pin" Width="Auto">
00695                                       <GridViewColumn.CellTemplate>
00696                                           <DataTemplate>
00697                                               <StackPanel Orientation="Horizontal">
00698                                                   <TextBlock Text="{Binding Pin}" Margin="0,0,10,0"/>
00699                                               </StackPanel>
00700                                           </DataTemplate>
00701                                       </GridViewColumn.CellTemplate>
00702                                   </GridViewColumn>
00703                                   <GridViewColumn Header="Address" Width="Auto">
00704                                       <GridViewColumn.CellTemplate>
00705                                           <DataTemplate>
00706                                               <TextBlock Text="{Binding Address}" FontWeight="Bold" Margin="0,0,10,0"/>
00707                                           </DataTemplate>
00708                                       </GridViewColumn.CellTemplate>
00709                                   </GridViewColumn>
00710                                   <GridViewColumn Header="Type" Width="Auto">
00711                                       <GridViewColumn.CellTemplate>
00712                                           <DataTemplate>
00713                                               <TextBlock Text="{Binding Type}" Foreground="Gray" Margin="0,0,10,0"/>
00714                                           </DataTemplate>
00715                                       </GridViewColumn.CellTemplate>
00716                                   </GridViewColumn>
00717                                   <GridViewColumn Header="Status" Width="170">
00718                                       <GridViewColumn.CellTemplate>
00719                                           <DataTemplate>
00720                                               <TextBlock>
00721                                                   <TextBlock.Style>
00722                                                       <Style TargetType="TextBlock">
00723                                                           <Setter Property="Text" Value=""/>
00724                                                           <Style.Triggers>
00725                                                           <DataTrigger Binding="{Binding IsInDeviceAndFromMqtt}" Value="True">
00726                                                                   <Setter Property="Text" Value="Sensor already configured"/>
00727                                                               </DataTrigger>
00728                                                               <DataTrigger Binding="{Binding IsInDeviceAndNotFromMqtt}"
```

```
    Value="True">
00729                                                 <Setter Property="Text" Value="Sensor not found on device"/>
00730                                             </DataTrigger>
00731                                             <DataTrigger Binding="{Binding IsNotInDeviceAndFromMqtt}"
    Value="True">
00732                                                 <Setter Property="Text" Value="Sensor is not added to device"/>
00733                                             </DataTrigger>
00734                                         </Style.Triggers>
00735                                     </Style>
00736                                 </TextBlock.Style>
00737                             </TextBlock>
00738                         </DataTemplate>
00739                     </GridViewColumn.CellTemplate>
00740                 </GridViewColumn>
00741                 <GridViewColumn Header="Name" Width="150">
00742                     <GridViewColumn.CellTemplate>
00743                         <DataTemplate>
00744                             <Grid>
00745                                 <TextBlock Text="{Binding SensorName}" Margin="0,0,10,0">
00746                                     <TextBlock.Style>
00747                                         <Style TargetType="TextBlock">
00748                                             <Setter Property="Visibility" Value="Collapsed"/>
00749                                             <Style.Triggers>
00750                                                 <DataTrigger Binding="{Binding IsInDevice}" Value="True">
00751                                                     <Setter Property="Visibility" Value="Visible"/>
00752                                                 </DataTrigger>
00753                                             </Style.Triggers>
00754                                         </Style>
00755                                     </TextBlock.Style>
00756                                 </TextBlock>
00757                                 <TextBox Text="{Binding SensorName, UpdateSourceTrigger=PropertyChanged}"
    Width="120" Margin="0,0,10,0">
00758                                     <TextBox.Style>
00759                                         <Style TargetType="TextBox">
00760                                             <Setter Property="Visibility" Value="Collapsed"/>
00761                                             <Style.Triggers>
00762                                                 <DataTrigger Binding="{Binding IsInDevice}" Value="False">
00763                                                     <Setter Property="Visibility" Value="Visible"/>
00764                                                 </DataTrigger>
00765                                             </Style.Triggers>
00766                                         </Style>
00767                                     </TextBox.Style>
00768                                 </TextBox>
00769                             </Grid>
00770                         </DataTemplate>
00771                     </GridViewColumn.CellTemplate>
00772                 </GridViewColumn>
00773                 <GridViewColumn Header="Action" Width="80">
00774                     <GridViewColumn.CellTemplate>
00775                         <DataTemplate>
00776                             <StackPanel Orientation="Horizontal">
00777                                 <!-- Action Button -->
00778                                 <Button x:Name="ActionButton" Width="60" Margin="2"
    Click="ActionButton_Click">
00779                                     <Button.Style>
00780                                         <Style TargetType="Button">
00781                                             <Setter Property="Content" Value=""/>
00782                                             <Setter Property="Visibility" Value="Collapsed"/>
00783                                             <Setter Property="Tag" Value="{Binding}"/>
00784                                             <Style.Triggers>
00785                                                 <!-- Add -->
00786                                                 <DataTrigger Binding="{Binding IsNotInDeviceAndFromMqtt}"
    Value="True">
00787                                                     <Setter Property="Content" Value="Add"/>
00788                                                     <Setter Property="Visibility" Value="Visible"/>
00789                                                     <Setter Property="Background" Value="LightGreen"/>
00790                                                 </DataTrigger>
00791                                                 <!-- Remove -->
00792                                                 <DataTrigger Binding="{Binding IsInDevice}" Value="True">
00793                                                     <Setter Property="Content" Value="Remove"/>
00794                                                     <Setter Property="Visibility" Value="Visible"/>
00795                                                     <Setter Property="Background" Value="LightCoral"/>
00796                                                 </DataTrigger>
00797                                             </Style.Triggers>
00798                                         </Style>
00799                                     </Button.Style>
00800                                 </Button>
00801                             </StackPanel>
00802                         </DataTemplate>
00803                     </GridViewColumn.CellTemplate>
00804                 </GridViewColumn>
00805             </GridView>
00806         </ListView.View>
00807     </ListView>
00808 </ScrollViewer>
00809 </Grid>
00810 </Expander>
```

```
00811         </Grid>
00812     </Grid>
00813 </Window>
```

7.9 ladder_diagram_app/MainWindow.xaml.cs File Reference

Classes

- class ladder_diagram_app.MainWindow

  Main application window for managing ladder diagrams, device communication, and variables.

Namespaces

- namespace ladder_diagram_app

7.10 MainWindow.xaml.cs

Go to the documentation of this file.

```
00001 using System.Diagnostics;
00002 using System.Windows;
00003 using System.Windows.Controls;
00004 using System.Windows.Input;
00005 using System.Windows.Media;
00006 using Microsoft.Win32;
00007 using System.IO;
00008 using System.Globalization;
00009 using System.Windows.Threading;
00010
00011 using ladder_diagram_app.Models.CanvasElements;
00012 using ladder_diagram_app.Models.DeviceElement;
00013 using ladder_diagram_app.Models.Variables.Instances;
00014 using ladder_diagram_app.Models.Variables;
00015 using ladder_diagram_app.Views;
00016 using ladder_diagram_app.Services.MonitorServices;
00017 using ladder_diagram_app.Services.ImportExportServices;
00018 using ladder_diagram_app.Services.CommunicationServices;
00019 using ladder_diagram_app.Services.CanvasServices;
00020
00021 namespace ladder_diagram_app
00022 {
00026     public partial class MainWindow : Window
00027     {
00031         public List<string> AdcSensorTypes { get; } = ["TM7711", "HX710B"];
00035         public List<string> AdcSensorSamplingRates { get; } = ["10Hz", "40Hz", "Temperature"];
00036
00037         // Device Element
00038         private readonly Device _device;
00042         public readonly DevicePinManager _devicePinManager;
00046         public DevicePinManager DevicePinManager => _devicePinManager;
00047
00048         // Variables
00049         private readonly VariablesManager _variablesManager;
00053         public VariablesManager VariablesManager => _variablesManager;
00054
00055         // Wires
00056         private readonly WiresManager _wiresManager;
00057
00058         // Canvas Services
00059         private readonly CanvasManager _canvasManager;
00060         private readonly CanvasElementFinder _canvasElementFinder;
00061         private readonly CanvasInteractionManager _canvasInteractionManager;
00062
00063         // Communication Services
00064         private readonly DeviceCommunicationManager _deviceCommunicationManager;
00065
00066         // Monitor Services
00067         private readonly MonitorDataService _monitorDataService;
00068         private readonly OneWireDataService _oneWireDataService;
00069
00070         // Import/Export Services
```

```
00071         private readonly ImportExportService _importExportService;
00072
00076         public MainWindow()
00077         {
00078             InitializeComponent();
00079             DataContext = this;
00080
00081             _device = new Device();
00082             _devicePinManager = new DevicePinManager();
00083
00084             _variablesManager = new VariablesManager(_device);
00085
00086             _wiresManager = new WiresManager();
00087
00088             _canvasManager = new CanvasManager(
00089                 canvas: MainCanvas,
00090                 gridCanvas: GridCanvas,
00091                 wiresManager: _wiresManager
00092             );
00093
00094             _canvasElementFinder = new CanvasElementFinder(
00095                 getWiresManager: () => _wiresManager
00096             );
00097
00098             _canvasInteractionManager = new CanvasInteractionManager(
00099                 canvas: MainCanvas,
00100                 wiresManager: _wiresManager,
00101                 elementFinder: _canvasElementFinder,
00102                 canvasManager: _canvasManager,
00103                 variablesManager: _variablesManager
00104             );
00105
00106             _monitorDataService = new MonitorDataService(this);
00107             _oneWireDataService = new OneWireDataService(this, _device);
00108
00109             _deviceCommunicationManager = new DeviceCommunicationManager(
00110                 onConfigurationReceived: jsonConfig => Dispatcher.Invoke(() => OnConfigurationReceived(jsonConfig)),
00111                 onMonitorDataReceived: jsonData => Dispatcher.Invoke(() =>
    _monitorDataService.OnMonitorDataReceived(jsonData)),
00112                 onOneWireDataReceived: jsonData => Dispatcher.Invoke(() =>
    _oneWireDataService.OnOneWireDataReceived(jsonData)),
00113                 onConnectionStatusChanged: isConnected => Dispatcher.Invoke(() =>
    OnConnectionStatusChanged(isConnected))
00114             );
00115
00116             _importExportService = new ImportExportService(
00117                 variablesManager: _variablesManager,
00118                 device: _device,
00119                 wiresManager: _wiresManager,
00120                 canvasManager: _canvasManager,
00121                 devicePinManager: _devicePinManager
00122             );
00123
00124             // Synchronize canvas sizes
00125             MainCanvas.Width = GridCanvas.ActualWidth;
00126             MainCanvas.Height = GridCanvas.ActualHeight;
00127
00128             // Initialize canvas with a wire
00129             _wiresManager.AddWire();
00130             _canvasManager.UpdateCanvas();
00131
00132             // Update canvas on grid size change
00133             GridCanvas.SizeChanged += (s, e) =>
00134             {
00135                 _canvasManager.UpdateCanvas();
00136             };
00137
00138             // Unselect elements when canvas loses focus, unless focus is on delete button
00139             GridCanvas.LostFocus += (s, e) =>
00140             {
00141                 // Check that the focus has not moved to ButtonDelete
00142                 if (Keyboard.FocusedElement != ButtonDelete)
00143                 {
00144                     _canvasInteractionManager.UnselectEverything();
00145                 }
00146             };
00147
00148             this.PreviewKeyDown += MainWindow_PreviewKeyDown;
00149
00150             this.Closing += Window_Closing;
00151         }
00152
00158         private void MainWindow_PreviewKeyDown(object sender, KeyEventArgs e)
00159         {
00160             if (e.Key == Key.Delete)
00161             {
00162                 // Delete selected variable from ListView
```

```
00163             if (ElementListView.IsKeyboardFocusWithin && ElementListView.SelectedItem is Variable selectedVariable)
00164             {
00165                 _variablesManager.DeleteVariable(selectedVariable, this);
00166                 e.Handled = true;
00167             }
00168             // Delete selected canvas element
00169             else if (_canvasInteractionManager.IsElementSelected())
00170             {
00171                 _canvasInteractionManager.DeleteSelected(this);
00172                 e.Handled = true;
00173             }
00174             else
00175             {
00176                 new NotificationWindow("Select an element or variable to delete", this).Show();
00177                 e.Handled = true;
00178             }
00179         }
00180         else if (e.Key == Key.Enter && NameTextBox.IsKeyboardFocused)
00181         {
00182             ButtonAddVariable_Click(null, null);
00183             e.Handled = true;
00184         }
00185     }
00186
00187     // ========================== IMPORT/EKSPORT
    ============================================================
00193     private void ButtonImport_Click(object sender, RoutedEventArgs e)
00194     {
00195         try
00196         {
00197             if (_device.IsDeviceLoaded())
00198             {
00199                 var dialog = new NotificationWindow("Are you sure you want to import another project? The current
    content will be deleted.", this, NotificationButtons.YesNo);
00200                 dialog.ShowDialog();
00201                 if (dialog.Result == false) return;
00202             }
00203
00204             OpenFileDialog openFileDialog = new OpenFileDialog
00205             {
00206                 Filter = "JSON files (*.json)|*.json|All files (*.*)|*.*",
00207                 Title = "Import Ladder Diagram"
00208             };
00209
00210             if (openFileDialog.ShowDialog() != true) return;
00211
00212             string jsonString = File.ReadAllText(openFileDialog.FileName);
00213
00214             _importExportService.ImportFromJson(jsonString, this);
00215         }
00216         catch (Exception ex)
00217         {
00218             new NotificationWindow("Error loading configuration", this).Show();
00219             Debug.WriteLine($"Error loading config: {ex.Message}");
00220         }
00221     }
00222
00228     private void ButtonExport_Click(object sender, RoutedEventArgs e)
00229     {
00230         var jsonString = _importExportService.ExportToJson(this);
00231         if (jsonString == null)
00232             return;
00233
00234         try
00235         {
00236             SaveFileDialog saveFileDialog = new SaveFileDialog
00237             {
00238                 Filter = "JSON files (*.json)|*.json|All files (*.*)|*.*",
00239                 Title = "Export Ladder Diagram",
00240                 FileName = "ladder_diagram.json"
00241             };
00242
00243             if (saveFileDialog.ShowDialog() == true)
00244             {
00245                 File.WriteAllText(saveFileDialog.FileName, jsonString);
00246                 FileInfo fileInfo = new FileInfo(saveFileDialog.FileName);
00247                 fileInfo.IsReadOnly = true;
00248                 new NotificationWindow("Export successful!", this).Show();
00249             }
00250         }
00251         catch (Exception ex)
00252         {
00253             new NotificationWindow($"Export failed: {ex.Message}", this).Show();
00254         }
00255     }
00256
00257     // ========================== VARIABLES
```

```
    ============================================================
00263       private void ButtonAddVariable_Click(object? sender, RoutedEventArgs? e)
00264       {
00265           string name = NameTextBox.Text.Trim();
00266           string type = TypeComboBox.Text;
00267
00268           _variablesManager.AddVariable(name, type, this);
00269
00270           NameTextBox.Clear();
00271       }
00272
00278       private void ButtonDeleteVariable_Click(object sender, RoutedEventArgs e)
00279       {
00280           if (sender is Button button && button.DataContext is Variable variable)
00281           {
00282               _variablesManager.DeleteVariable(variable, this);
00283           }
00284       }
00285
00291       private void ButtonChangeBoolean_Click(object sender, RoutedEventArgs e)
00292       {
00293           if (sender is Button button && button.DataContext is Variable variable)
00294           {
00295               _variablesManager.VariableBooleanClick(variable);
00296           }
00297       }
00298
00304       private void TextBoxVariable_TextChanged(object sender, RoutedEventArgs e)
00305       {
00306           if (sender is TextBox tb && tb.DataContext is Variable variable)
00307           {
00308               bool isEnterPressed = e is KeyEventArgs keyArgs && keyArgs.Key == Key.Enter;
00309               bool isLostFocus = e is RoutedEventArgs && e.RoutedEvent == UIElement.LostFocusEvent;
00310
00311               string inputText = tb.Text.Trim();
00312
00313               if (isEnterPressed)
00314                   ElementListView.Focus();
00315
00316               if (isEnterPressed || isLostFocus)
00317               {
00318                   _variablesManager.VariableTextBoxChange(variable, inputText);
00319               }
00320           }
00321       }
00322
00328       private void ComboBoxVariable_SelectionChanged(object sender, RoutedEventArgs e)
00329       {
00330           if (sender is ComboBox cb && cb.DataContext is Variable variable)
00331           {
00332               string? selectedValue = cb.SelectedItem != null ? cb.SelectedItem.ToString() : null;
00333               if (selectedValue != null) _variablesManager.VariableComboBoxChange(variable, selectedValue);
00334           }
00335       }
00336
00342       private void TextBoxVariable_PreviewTextInput(object sender, TextCompositionEventArgs e)
00343       {
00344           if (sender is TextBox textBox)
00345           {
00346               // Get the current text and cursor position
00347               string currentText = textBox.Text;
00348               int caretIndex = textBox.CaretIndex;
00349               string newText;
00350
00351               // If the entire text is selected, replace it with the new entry
00352               if (textBox.SelectedText == currentText && !string.IsNullOrEmpty(currentText))
00353               {
00354                   newText = e.Text;
00355               }
00356               else
00357               {
00358                   newText = currentText.Insert(caretIndex, e.Text);
00359               }
00360
00361               // Allow input if result is empty, "-", or valid double
00362               e.Handled = !(string.IsNullOrEmpty(newText) ||
00363                             newText == "-" ||
00364                             double.TryParse(newText, NumberStyles.Any, CultureInfo.InvariantCulture, out _));
00365           }
00366       }
00367
00368       // ========================== DEVICE INFO
    ============================================================
00374       private void ButtonDeviceInfo_Click(object sender, RoutedEventArgs e)
00375       {
00376           new NotificationWindow(_device.DeviceInfo(), this, NotificationButtons.Ok).Show();
00377       }
```

```
00378
00379       // ========================== PARENT DEVICE(S)
    ==========================================================
00385       private void ButtonParentDevice_Click(object sender, RoutedEventArgs e)
00386       {
00387           _device.AddParentDevices(this);
00388       }
00389
00390       // ========================== ADDING WIRE
    ============================================================
00396       private void ButtonAddWire_Click(object sender, RoutedEventArgs e)
00397       {
00398           _wiresManager.AddWire();
00399           _canvasManager.UpdateCanvas();
00400       }
00401
00402       // ========================== CANVAS
    ================================================================
00408       private void Button_PreviewMouseLeftButtonDown(object sender, MouseEventArgs e)
00409       {
00410           if (e.LeftButton == MouseButtonState.Pressed && sender is Button button && button.Tag is string tag)
00411           {
00412               DragDrop.DoDragDrop(button, tag, DragDropEffects.Copy);
00413           }
00414       }
00415
00421       private void Canvas_DragOver(object sender, DragEventArgs e)
00422       {
00423           _canvasInteractionManager.HandleDragOver(e);
00424       }
00425
00431       private void Canvas_Drop(object sender, DragEventArgs e)
00432       {
00433           _canvasInteractionManager.HandleDrop(e, this);
00434       }
00435
00441       private void MainCanvas_MouseMove(object sender, MouseEventArgs e)
00442       {
00443           _canvasInteractionManager.HandleMouseMove(e);
00444       }
00445
00451       private void MainCanvas_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
00452       {
00453           _canvasInteractionManager.HandleMouseLeftButtonUp(e, this);
00454       }
00455
00461       private void MainCanvas_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
00462       {
00463           _canvasInteractionManager.HandleMouseLeftButtonDown(e, ElementListView);
00464       }
00465
00471       private void ButtonDelete_Click(object? sender, RoutedEventArgs? e)
00472       {
00473           _canvasInteractionManager.DeleteSelected(this);
00474       }
00475
00476       // ========================== DEVICE COMMUNICATION
    ================================================
00482       private async void ButtonConnect_Click(object sender, RoutedEventArgs e)
00483       {
00484           var connectionType = ConnectionMQTT.IsChecked == true ? "MQTT" : "BLE";
00485           await _deviceCommunicationManager.ConnectAsync(this, connectionType);
00486       }
00487
00493       private async void ButtonDisconnect_Click(object sender, RoutedEventArgs e)
00494       {
00495           await _deviceCommunicationManager.DisconnectAsync(this);
00496       }
00497
00503       private async void ButtonSendToDevice_Click(object sender, RoutedEventArgs e)
00504       {
00505           var jsonString = _importExportService.ExportToJson(this);
00506           if (jsonString == null) return;
00507
00508           await _deviceCommunicationManager.SendConfigurationAsync(jsonString, this);
00509       }
00510
00515       private void OnConfigurationReceived(string jsonConfig)
00516       {
00517           _importExportService.ImportFromJson(jsonConfig, this);
00518
00519           _oneWireDataService.DeleteLastOneWireMessage();
00520       }
00521
00526       private void OnConnectionStatusChanged(bool isConnected)
00527       {
00528           Dispatcher.InvokeAsync(() =>
```

```
00529         {
00530             // Update connection status display
00531             StatusRun.Text = isConnected ? $"Connected
      ({_deviceCommunicationManager._communicationService?.ConnectionType})" : "Not Connected";
00532             StatusRun.Foreground = isConnected ? Brushes.Green : Brushes.Red;
00533 
00534             // Show/hide monitor grid based on connection status
00535             MonitorGrid.Visibility = isConnected ? Visibility.Visible : Visibility.Collapsed;
00536             MonitorExpander.IsExpanded = isConnected;   // Open Expander only when connected
00537 
00538             // Adjust row height in parent grid
00539             var parentGrid = MonitorGrid.Parent as Grid;
00540             if (parentGrid != null)
00541             {
00542                 var row = Grid.GetRow(MonitorGrid);
00543                 parentGrid.RowDefinitions[row].Height = isConnected ? new GridLength(100) : GridLength.Auto;
00544             }
00545 
00546             // Show connection status notification
00547             new NotificationWindow(isConnected ? "Device Connected" : "Device Disconnected", this).Show();
00548         });
00549     }
00550 
00551     // ========================== MANAGING THE MONITOR / ONE WIRE SECTION
      ==============================
00557     private void MonitorExpander_Collapsed(object sender, RoutedEventArgs e)
00558     {
00559         // Reset the row height to Auto so that the Expander sticks to the bottom
00560         var parentGrid = MonitorGrid.Parent as Grid;
00561         if (parentGrid != null)
00562         {
00563             var row = Grid.GetRow(MonitorGrid);
00564             parentGrid.RowDefinitions[row].Height = GridLength.Auto;
00565         }
00566     }
00567 
00573     private void ActionButton_Click(object sender, RoutedEventArgs e)
00574     {
00575         _oneWireDataService.ActionButton_Click(sender, e);
00576     }
00577 
00578     // ========================== CLOSING THE APP
      ============================================================
00584     private async void Window_Closing(object? sender, System.ComponentModel.CancelEventArgs e)
00585     {
00586         if (_deviceCommunicationManager != null)
00587         {
00588             await _deviceCommunicationManager.DisconnectAsync(this);
00589             _deviceCommunicationManager.Dispose();
00590         }
00591     }
00592 }
00593 }
```

7.11 ladder_diagram_app/Models/CanvasElements/Instances/↩ Branch.cs File Reference

Classes

- class ladder_diagram_app.Models.CanvasElements.Instances.Branch

  Represents a branch node in a ladder diagram, containing two lists of child nodes and visual lines for rendering.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.CanvasElements
- namespace ladder_diagram_app.Models.CanvasElements.Instances

7.12 Branch.cs

Go to the documentation of this file.

```
00001 using System.Windows.Controls;
00002 using System.Windows.Media;
00003 using System.Windows.Media.Imaging;
00004 using System.Windows.Media.Effects;
00005 using System.Windows.Shapes;
00006
00007 namespace ladder_diagram_app.Models.CanvasElements.Instances
00008 {
00012     public class Branch : Node
00013     {
00017         public List<Node> Nodes1 { get; set; }
00018
00022         public List<Node> Nodes2 { get; set; }
00023
00027         private double _y;
00028
00032         private double _x;
00033
00037         public double Y2 { get; set; }
00038
00042         public Line UpperLine { get; set; }
00043         public Line LowerLine { get; set; }
00044         public Line LeftLine { get; set; }
00045         public Line RightLine { get; set; }
00046
00050         public Branch()
00051         {
00052             // Initialize the branch icon with a drop shadow effect
00053             Image = new Image
00054             {
00055                 Width = 48,
00056                 Height = 24,
00057                 Source = new BitmapImage(new Uri("pack://application:,,,/Resources/Branch/branch_icon.png")),
00058                 Effect = new DropShadowEffect
00059                 {
00060                     Color = Colors.Red,
00061                     Direction = 0,
00062                     ShadowDepth = 0,
00063                     BlurRadius = 10,
00064                     Opacity = 0.8
00065                 }
00066             };
00067
00068             // Initialize node lists
00069             Nodes1 = [];
00070             Nodes2 = [];
00071
00072             // Initialize lines with default properties and tag them with this branch instance
00073             UpperLine = new Line() { X1 = 0, Y1 = 0, X2 = 0, Y2 = 0, Stroke = Brushes.Black, StrokeThickness = 2, Tag =
    this };
00074             LowerLine = new Line() { X1 = 0, Y1 = 0, X2 = 0, Y2 = 0, Stroke = Brushes.Black, StrokeThickness = 2, Tag =
    this };
00075             LeftLine = new Line() { X1 = 0, Y1 = 0, X2 = 0, Y2 = 0, Stroke = Brushes.Black, StrokeThickness = 2, Tag =
    this };
00076             RightLine = new Line() { X1 = 0, Y1 = 0, X2 = 0, Y2 = 0, Stroke = Brushes.Black, StrokeThickness = 2, Tag =
    this };
00077         }
00078
00082         public override double Y
00083         {
00084             get => _y;
00085             set
00086             {
00087                 _y = value;
00088                 Y2 = _y + CalculateY2(Nodes1);  // Update Y2 based on child nodes
00089                 UpdateLines(); // Refresh line positions
00090             }
00091         }
00092
00096         public override double X
00097         {
00098             get => _x;
00099             set
00100             {
00101                 _x = value;
00102                 UpdateLines(); // Refresh line positions
00103             }
00104         }
00105
00109         private void UpdateLines()
00110         {
00111             // Update upper horizontal line
```

```
00112           UpperLine.X1 = X - Width / 2 + 10;
00113           UpperLine.Y1 = Y;
00114           UpperLine.X2 = X + Width / 2 - 10;
00115           UpperLine.Y2 = Y;
00116
00117           // Update lower horizontal line
00118           LowerLine.X1 = X - Width / 2 + 10;
00119           LowerLine.Y1 = Y2;
00120           LowerLine.X2 = X + Width / 2 - 10;
00121           LowerLine.Y2 = Y2;
00122
00123           // Update left vertical line
00124           LeftLine.X1 = X - Width / 2 + 10;
00125           LeftLine.Y1 = Y;
00126           LeftLine.X2 = X - Width / 2 + 10;
00127           LeftLine.Y2 = Y2;
00128
00129           // Update right vertical line
00130           RightLine.X1 = X + Width / 2 - 10;
00131           RightLine.Y1 = Y;
00132           RightLine.X2 = X + Width / 2 - 10;
00133           RightLine.Y2 = Y2;
00134       }
00135
00141       private static double CalculateY2(List<Node> nodes)
00142       {
00143           double maxY2 = 125; // Default minimum height for empty or non-branch nodes
00144
00145           // Iterate through nodes to calculate height
00146           foreach (Node node in nodes)
00147           {
00148               if (node is Branch branch)
00149               {
00150                   // Recursively calculate heights for sub-branch Nodes1 and Nodes2
00151                   double nodes1Height = CalculateY2(branch.Nodes1);
00152                   double nodes2Height = CalculateY2(branch.Nodes2);
00153
00154                   // Sum heights as branches extend downward
00155                   double branchY2 = nodes1Height + nodes2Height;
00156
00157                   // Update maxY2 if this branch is deeper
00158                   maxY2 = Math.Max(maxY2, branchY2);
00159               }
00160           }
00161           return maxY2;
00162       }
00163
00167       public override double Width
00168       {
00169           get
00170           {
00171               double nodes1Width = CalculateTotalWidth(Nodes1);
00172               double nodes2Width = CalculateTotalWidth(Nodes2);
00173               // Return the maximum width plus padding, or default width of 130 if no nodes
00174               return Math.Max(nodes1Width, nodes2Width) != 0 ? Math.Max(nodes1Width, nodes2Width) + 20 : 130;
00175           }
00176       }
00177
00183       private static double CalculateTotalWidth(List<Node> nodes)
00184       {
00185           double totalWidth = 0;
00186           foreach (var node in nodes)
00187           {
00188               totalWidth += node.Width;
00189           }
00190           return totalWidth;
00191       }
00192
00197       public void HighlightBranch(bool isUpperLine)
00198       {
00199           if (isUpperLine)
00200           {
00201               // Highlight upper line
00202               UpperLine.Stroke = Brushes.Blue;
00203               UpperLine.StrokeThickness = 4;
00204               UpperLine.StrokeDashArray = [2, 1];
00205           }
00206           else
00207           {
00208               // Highlight lower line
00209               LowerLine.Stroke = Brushes.Blue;
00210               LowerLine.StrokeThickness = 4;
00211               LowerLine.StrokeDashArray = [2, 1];
00212           }
00213       }
00214
00218       public void UnhighlightBranch()
```

```
00219         {
00220             // Reset upper line
00221             UpperLine.Stroke = Brushes.Black;
00222             UpperLine.StrokeThickness = 2;
00223             UpperLine.StrokeDashArray = null;
00224 
00225             // Reset lower line
00226             LowerLine.Stroke = Brushes.Black;
00227             LowerLine.StrokeThickness = 2;
00228             LowerLine.StrokeDashArray = null;
00229         }
00230 
00234         public void HighlightBranchRecursive()
00235         {
00236             // Highlight all lines of the current branch
00237             UpperLine.Stroke = Brushes.Red; UpperLine.StrokeThickness = 4;
00238             LowerLine.Stroke = Brushes.Red; LowerLine.StrokeThickness = 4;
00239             LeftLine.Stroke = Brushes.Red; LeftLine.StrokeThickness = 4;
00240             RightLine.Stroke = Brushes.Red; RightLine.StrokeThickness = 4;
00241 
00242             // Recursively highlight sub-branches in Nodes1 and Nodes2
00243             foreach (var node in Nodes1.Concat(Nodes2))
00244             {
00245                 if (node is Branch subBranch)
00246                 {
00247                     subBranch.HighlightBranchRecursive();
00248                 }
00249             }
00250         }
00251 
00255         public void UnhighlightBranchRecursive()
00256         {
00257             // Reset all lines of the current branch
00258             UpperLine.Stroke = Brushes.Black; UpperLine.StrokeThickness = 2;
00259             LowerLine.Stroke = Brushes.Black; LowerLine.StrokeThickness = 2;
00260             LeftLine.Stroke = Brushes.Black; LeftLine.StrokeThickness = 2;
00261             RightLine.Stroke = Brushes.Black; RightLine.StrokeThickness = 2;
00262 
00263             // Recursively reset sub-branches in Nodes1 and Nodes2
00264             foreach (var node in Nodes1.Concat(Nodes2))
00265             {
00266                 if (node is Branch subBranch)
00267                 {
00268                     subBranch.UnhighlightBranchRecursive();
00269                 }
00270             }
00271         }
00272 
00278         public bool IsBranchNested(Branch targetBranch)
00279         {
00280             if (this == targetBranch) return true; // Same branch
00281 
00282             // Recursively check Nodes1 for nested branches
00283             foreach (var node in Nodes1)
00284             {
00285                 if (node is Branch subBranch && subBranch.IsBranchNested(targetBranch))
00286                 {
00287                     return true;
00288                 }
00289             }
00290 
00291             // Recursively check Nodes2 for nested branches
00292             foreach (var node in Nodes2)
00293             {
00294                 if (node is Branch subBranch && subBranch.IsBranchNested(targetBranch))
00295                 {
00296                     return true;
00297                 }
00298             }
00299 
00300             return false; // Not nested
00301         }
00302     }
00303 }
```

7.13 ladder_diagram_app/Models/CanvasElements/Instances/LadderElement.cs File Reference

Classes

- class ladder_diagram_app.Models.CanvasElements.Instances.LadderElement

Represents a ladder diagram element (e.g., contact, coil, timer) with an associated image and variable selection ComboBoxes.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.CanvasElements
- namespace ladder_diagram_app.Models.CanvasElements.Instances

7.14 LadderElement.cs

Go to the documentation of this file.

```
00001 using System.Collections.ObjectModel;
00002 using System.Windows.Controls;
00003 using System.Windows.Media.Imaging;
00004
00005 namespace ladder_diagram_app.Models.CanvasElements.Instances
00006 {
00010     public class LadderElement : Node
00011     {
00015         private readonly List<ComboBox> _variableComboBoxes = [];
00016
00020         public string Type { get; set; }
00021
00025         public override double X { get; set; }
00026
00030         public override double Y { get; set; }
00031
00035         public override double Width => Math.Max(Image?.Width ?? 0, _variableComboBoxes.Any() ?
      _variableComboBoxes.Max(cb => cb?.Width ?? 0) : 0) + 10; // 10 -> 5px margin sa svake strane
00036
00037
00049         public LadderElement(string type,
00050             ObservableCollection<string> variablesListContacts,
00051             ObservableCollection<string> variablesListCoils,
00052             ObservableCollection<string> variablesListMath,
00053             ObservableCollection<string> variablesListCompare,
00054             ObservableCollection<string> variablesListCounter,
00055             ObservableCollection<string> variablesListTimer,
00056             ObservableCollection<string> variablesListReset)
00057         {
00058             Type = type;
00059
00060             // Initialize the image for the ladder element
00061             Image = new Image();
00062             if (!string.IsNullOrEmpty(Type))
00063             {
00064                 // Set image dimensions based on element type
00065                 (Image.Width, Image.Height) = Type switch
00066                 {
00067                     "NOContact" => (48, 24),
00068                     "NCContact" => (48, 24),
00069                     "Coil" => (48, 24),
00070                     "OSPCoil" => (48, 24),
00071                     "SetCoil" => (48, 24),
00072                     "ResetCoil" => (48, 24),
00073                     "AddMath" => (114, 119),
00074                     "SubtractMath" => (114, 119),
00075                     "MultiplyMath" => (114, 119),
00076                     "DivideMath" => (114, 119),
00077                     "MoveMath" => (114, 104),
00078                     "GreaterCompare" => (114, 84),
00079                     "LessCompare" => (114, 84),
00080                     "GreaterOrEqualCompare" => (114, 84),
00081                     "LessOrEqualCompare" => (114, 84),
00082                     "EqualCompare" => (114, 84),
00083                     "NotEqualCompare" => (114, 84),
00084                     "OnDelayTimer" => (114, 54),
00085                     "OffDelayTimer" => (114, 54),
00086                     "CountUp" => (114, 54),
00087                     "CountDown" => (114, 54),
00088                     "Reset" => (114, 54),
00089                     _ => (0, 0) // Default case for unknown types
00090                 };
```

```
00091
00092             // Set image source based on element type
00093             string imagePath = Type switch
00094             {
00095                 "NOContact" => "pack://application:,,,/Resources/Contacts/no_contact.png",
00096                 "NCContact" => "pack://application:,,,/Resources/Contacts/nc_contact.png",
00097                 "Coil" => "pack://application:,,,/Resources/Coils/coil.png",
00098                 "OSPCoil" => "pack://application:,,,/Resources/Coils/one_shot_positive_coil.png",
00099                 "SetCoil" => "pack://application:,,,/Resources/Coils/set_coil.png",
00100                 "ResetCoil" => "pack://application:,,,/Resources/Coils/reset_coil.png",
00101                 "AddMath" => "pack://application:,,,/Resources/Math/add.png",
00102                 "SubtractMath" => "pack://application:,,,/Resources/Math/subtract.png",
00103                 "MultiplyMath" => "pack://application:,,,/Resources/Math/multiply.png",
00104                 "DivideMath" => "pack://application:,,,/Resources/Math/divide.png",
00105                 "MoveMath" => "pack://application:,,,/Resources/Math/move.png",
00106                 "GreaterCompare" => "pack://application:,,,/Resources/Compare/greater.png",
00107                 "LessCompare" => "pack://application:,,,/Resources/Compare/less.png",
00108                 "GreaterOrEqualCompare" => "pack://application:,,,/Resources/Compare/greater_or_equal.png",
00109                 "LessOrEqualCompare" => "pack://application:,,,/Resources/Compare/less_or_equal.png",
00110                 "EqualCompare" => "pack://application:,,,/Resources/Compare/equal.png",
00111                 "NotEqualCompare" => "pack://application:,,,/Resources/Compare/not_equal.png",
00112                 "OnDelayTimer" => "pack://application:,,,/Resources/Time_Count/on_delay_timer.png",
00113                 "OffDelayTimer" => "pack://application:,,,/Resources/Time_Count/off_delay_timer.png",
00114                 "CountUp" => "pack://application:,,,/Resources/Time_Count/count_up.png",
00115                 "CountDown" => "pack://application:,,,/Resources/Time_Count/count_down.png",
00116                 "Reset" => "pack://application:,,,/Resources/Time_Count/reset.png",
00117                 _ => "" // Default case for unknown types
00118             };
00119
00120             // Set image source only if a valid path is provided
00121             if (!string.IsNullOrEmpty(imagePath))
00122             {
00123                 Image.Source = new BitmapImage(new Uri(imagePath));
00124             }
00125         }
00126
00127         // Determine the number of ComboBoxes based on element type
00128         int comboBoxCount = Type switch
00129         {
00130             "AddMath" or "SubtractMath" or "MultiplyMath" or "DivideMath" => 3,
00131             "MoveMath" or "GreaterCompare" or "LessCompare" or
00132             "GreaterOrEqualCompare" or "LessOrEqualCompare" or
00133             "EqualCompare" or "NotEqualCompare" => 2,
00134             _ => 1 // Default case for NOContact, NCContact, Coil, etc.
00135         };
00136
00137         // Initialize ComboBoxes for variable selection
00138         for (int i = 0; i < comboBoxCount; i++)
00139         {
00140             var comboBox = new ComboBox
00141             {
00142                 Width = 100,
00143                 Height = 25,
00144                 Tag = this, // Associate ComboBox with this LadderElement
00145                 SelectedIndex = -1 // No initial selection
00146             };
00147
00148             // Assign appropriate variable list based on element type
00149             if (Type.Contains("Contact"))
00150                 comboBox.ItemsSource = variablesListContacts;
00151             else if (Type.Contains("Coil"))
00152                 comboBox.ItemsSource = variablesListCoils;
00153             else if (Type.Contains("Math"))
00154                 comboBox.ItemsSource = variablesListMath;
00155             else if (Type.Contains("Compare"))
00156                 comboBox.ItemsSource = variablesListCompare;
00157             else if (Type.Contains("Timer"))
00158                 comboBox.ItemsSource = variablesListTimer;
00159             else if (Type.Contains("Count"))
00160                 comboBox.ItemsSource = variablesListCounter;
00161             else if (Type.Contains("Reset"))
00162                 comboBox.ItemsSource = variablesListReset;
00163
00164             _variableComboBoxes.Add(comboBox);
00165         }
00166     }
00167
00171     public IReadOnlyList<ComboBox> VariableComboBoxes => _variableComboBoxes.AsReadOnly();
00172
00176     public int ComboBoxCount => _variableComboBoxes.Count;
00177 }
00178 }
```

7.15 ladder_diagram_app/Models/CanvasElements/Instances/Node.cs File Reference

Classes

- class ladder_diagram_app.Models.CanvasElements.Instances.Node

  Abstract base class for nodes in a ladder diagram, providing common properties and methods for positioning and highlighting.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.CanvasElements
- namespace ladder_diagram_app.Models.CanvasElements.Instances

7.16 Node.cs

Go to the documentation of this file.

```
00001 using System.Windows.Controls;
00002 using System.Windows.Media;
00003 using System.Windows.Media.Effects;
00004
00005 namespace ladder_diagram_app.Models.CanvasElements.Instances
00006 {
00010     public abstract class Node
00011     {
00015         public abstract double X { get; set; }
00016
00020         public abstract double Y { get; set; }
00021
00025         public object? Parent { get; set; }
00026
00030         public abstract double Width { get; }
00031
00035         public virtual Image? Image { get; set; }
00036
00037
00041         public void HighlightNode()
00042         {
00043             // Apply highlight effect only if the node has an image
00044             if (Image != null)
00045             {
00046                 Image.Effect = new DropShadowEffect
00047                 {
00048                     Color = Colors.Red,
00049                     Direction = 0,
00050                     ShadowDepth = 0,
00051                     BlurRadius = 10,
00052                     Opacity = 0.8
00053                 };
00054             }
00055         }
00056
00060         public void UnhighlightNode()
00061         {
00062             // Clear highlight effect only if the node has an image
00063             if (Image != null)
00064             {
00065                 Image.Effect = null;
00066             }
00067         }
00068     }
00069 }
```

7.17 ladder_diagram_app/Models/CanvasElements/Instances/Wire.cs File Reference

Classes

- class ladder_diagram_app.Models.CanvasElements.Instances.Wire

  Represents a wire in a ladder diagram, connecting nodes with a visual line.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.CanvasElements
- namespace ladder_diagram_app.Models.CanvasElements.Instances

7.18 Wire.cs

Go to the documentation of this file.

```
00001 using System.Windows.Media;
00002 using System.Windows.Shapes;
00003
00004 namespace ladder_diagram_app.Models.CanvasElements.Instances
00005 {
00009     public class Wire
00010     {
00014         private double _y;
00015
00019         public double Width { get; set; }
00020
00024         public List<Node> Nodes { get; set; }
00025
00029         public Line WireLine { get; set; }
00030
00034         public Wire()
00035         {
00036             Nodes = [];
00037             WireLine = new Line() {
00038                 X1 = 0,
00039                 Y1 = 0,
00040                 X2 = 0,
00041                 Y2 = 0,
00042                 Stroke = Brushes.Black,
00043                 StrokeThickness = 2,
00044                 Tag = this
00045             };
00046         }
00047
00051         public double Y
00052         {
00053             get => _y;
00054             set
00055             {
00056                 _y = value;
00057                 UpdateWireLine();   // Refresh wire line position
00058             }
00059         }
00060
00064         private void UpdateWireLine()
00065         {
00066             WireLine.X1 = 0;
00067             WireLine.Y1 = Y;
00068             WireLine.X2 = Width;
00069             WireLine.Y2 = Y;
00070         }
00071
00075         public double Height
00076         {
00077             get
00078             {
00079                 return GetMaxY2FromBranches(Nodes) - Y + 125;   // Add minimum height margin
```

```
00080             }
00081         }
00082
00088         private double GetMaxY2FromBranches(List<Node> nodes)
00089         {
00090             double maxY2 = Y;
00091             foreach (var node in nodes)
00092             {
00093                 if (node is Branch branch)
00094                 {
00095                     // Compare current branch Y2
00096                     maxY2 = Math.Max(maxY2, branch.Y2);
00097                     // Recursively check Nodes2 for deeper branches
00098                     double maxY2FromNodes2 = GetMaxY2FromBranches(branch.Nodes2);
00099                     maxY2 = Math.Max(maxY2, maxY2FromNodes2);
00100                 }
00101             }
00102             return maxY2;
00103         }
00104
00108         public void SelectWire()
00109         {
00110             WireLine.Stroke = Brushes.Red;
00111             WireLine.StrokeThickness = 4;
00112         }
00113
00117         public void UnselectWire()
00118         {
00119             WireLine.Stroke = Brushes.Black;
00120             WireLine.StrokeThickness = 2;
00121         }
00122
00126         public void HighlightWire()
00127         {
00128             WireLine.Stroke = Brushes.Blue;
00129             WireLine.StrokeThickness = 4;
00130             WireLine.StrokeDashArray = [2, 1];
00131         }
00132
00136         public void UnhighlightWire()
00137         {
00138             WireLine.Stroke = Brushes.Black;
00139             WireLine.StrokeThickness = 2;
00140             WireLine.StrokeDashArray = null;
00141         }
00142     }
00143 }
```

7.19 ladder_diagram_app/Models/CanvasElements/WiresManager.cs File Reference

Classes

- class ladder_diagram_app.Models.CanvasElements.WiresManager

  Manages a collection of wires in a ladder diagram, providing methods to add, remove, insert, and clear wires.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.CanvasElements

7.20 WiresManager.cs

Go to the documentation of this file.

```
00001 using ladder_diagram_app.Models.CanvasElements.Instances;
```

```
00002
00003 namespace ladder_diagram_app.Models.CanvasElements
00004 {
00008     public class WiresManager
00009     {
00013         public List<Wire> Wires { get; }
00014
00018         public WiresManager()
00019         {
00020             Wires = [];
00021         }
00022
00027         public void AddWire(Wire? wire = null)
00028         {
00029             Wires.Add(wire ?? new Wire()); // Use provided wire or create a new one
00030         }
00031
00036         public void RemoveWire(Wire wire)
00037         {
00038             Wires.Remove(wire); // Remove the specified wire
00039         }
00040
00046         public void InsertWire(Wire wire, int index)
00047         {
00048             Wires.Insert(index, wire); // Insert wire at the specified index
00049         }
00050
00054         public void ClearWires()
00055         {
00056             Wires.Clear(); // Remove all wires
00057         }
00058     }
00059 }
```

7.21 ladder_diagram_app/Models/DeviceElement/Device.cs File Reference

Classes

- class ladder_diagram_app.Models.DeviceElement.Device

  Represents a device in a ladder diagram application, encapsulating its properties and configuration.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.DeviceElement

7.22 Device.cs

Go to the documentation of this file.

```
00001 using ladder_diagram_app.Views;
00002 using System.Windows;
00003
00004 namespace ladder_diagram_app.Models.DeviceElement
00005 {
00009     public class Device
00010     {
00014         public string device_name { get; set; }
00015
00019         public double logic_voltage { get; set; }
00020
00024         public List<int> digital_inputs { get; set; }
00025
00029         public List<string> digital_inputs_names { get; set; }
00030
00034         public List<int> digital_outputs { get; set; }
```

```
00035
00039         public List<string> digital_outputs_names { get; set; }
00040
00044         public List<int> analog_inputs { get; set; }
00045
00049         public List<string> analog_inputs_names { get; set; }
00050
00054         public List<int> dac_outputs { get; set; }
00055
00059         public List<string> dac_outputs_names { get; set; }
00060
00064         public List<int> one_wire_inputs { get; set; }
00065
00069         public List<List<string» one_wire_inputs_names { get; set; }
00070
00074         public List<List<string» one_wire_inputs_devices_types { get; set; }
00075
00079         public List<List<string» one_wire_inputs_devices_addresses { get; set; }
00080
00084         public int pwm_channels { get; set; }
00085
00089         public int max_hardware_timers { get; set; }
00090
00094         public bool has_rtos { get; set; }
00095
00099         public List<int> UART { get; set; }
00100
00104         public List<int> I2C { get; set; }
00105
00109         public List<int> SPI { get; set; }
00110
00114         public bool USB { get; set; }
00115
00119         public List<string> parent_devices { get; set; }
00120
00124         public Device()
00125         {
00126             device_name = string.Empty;
00127             logic_voltage = 0;
00128             digital_inputs = [];
00129             digital_inputs_names = [];
00130             digital_outputs = [];
00131             digital_outputs_names = [];
00132             analog_inputs = [];
00133             analog_inputs_names = [];
00134             dac_outputs = [];
00135             dac_outputs_names = [];
00136             one_wire_inputs = [];
00137             one_wire_inputs_names = [];
00138             one_wire_inputs_devices_types = [];
00139             one_wire_inputs_devices_addresses = [];
00140             pwm_channels = 0;
00141             max_hardware_timers = 0;
00142             has_rtos = false;
00143             UART = [];
00144             I2C = [];
00145             SPI = [];
00146             USB = false;
00147             parent_devices = [];
00148         }
00149
00154         public bool IsDeviceLoaded()
00155         {
00156             return !string.IsNullOrEmpty(device_name);
00157         }
00158
00163         public void AddParentDevices(Window owner)
00164         {
00165             var addParentsWindow = new AddParentsWindow(parent_devices, owner);
00166             addParentsWindow.ShowDialog();
00167         }
00168
00173         public void UpdateFrom(Device device)
00174         {
00175             if (device == null)
00176                 return; // Exit if source device is null
00177
00178             device_name = device.device_name ?? string.Empty;
00179             logic_voltage = device.logic_voltage;
00180             digital_inputs = device.digital_inputs != null ? new List<int>(device.digital_inputs) : new List<int>();
00181             digital_inputs_names = device.digital_inputs_names != null ? new List<string>(device.digital_inputs_names) :
    new List<string>();
00182             digital_outputs = device.digital_outputs != null ? new List<int>(device.digital_outputs) : new List<int>();
00183             digital_outputs_names = device.digital_outputs_names != null ? new
    List<string>(device.digital_outputs_names) : new List<string>();
00184             analog_inputs = device.analog_inputs != null ? new List<int>(device.analog_inputs) : new List<int>();
00185             analog_inputs_names = device.analog_inputs_names != null ? new List<string>(device.analog_inputs_names) :
```

```
    new List<string>();
00186           dac_outputs = device.dac_outputs != null ? new List<int>(device.dac_outputs) : new List<int>();
00187           dac_outputs_names = device.dac_outputs_names != null ? new List<string>(device.dac_outputs_names) : new
    List<string>();
00188           one_wire_inputs = device.one_wire_inputs != null ? new List<int>(device.one_wire_inputs) : new List<int>();
00189           one_wire_inputs_names = device.one_wire_inputs_names != null
00190               ? device.one_wire_inputs_names.Select(inner => inner != null ? new List<string>(inner) : new
    List<string>()).ToList()
00191               : new List<List<string»();
00192           one_wire_inputs_devices_types = device.one_wire_inputs_devices_types != null
00193               ? device.one_wire_inputs_devices_types.Select(inner => inner != null ? new List<string>(inner) : new
    List<string>()).ToList()
00194               : new List<List<string»();
00195           one_wire_inputs_devices_addresses = device.one_wire_inputs_devices_addresses != null
00196               ? device.one_wire_inputs_devices_addresses.Select(inner => inner != null ? new List<string>(inner) : new
    List<string>()).ToList()
00197               : new List<List<string»();
00198           pwm_channels = device.pwm_channels;
00199           max_hardware_timers = device.max_hardware_timers;
00200           has_rtos = device.has_rtos;
00201           UART = device.UART != null ? new List<int>(device.UART) : new List<int>();
00202           I2C = device.I2C != null ? new List<int>(device.I2C) : new List<int>();
00203           SPI = device.SPI != null ? new List<int>(device.SPI) : new List<int>();
00204           USB = device.USB;
00205           parent_devices = device.parent_devices != null ? new List<string>(device.parent_devices) : new List<string>();
00206       }
00207
00212       public bool Validate()
00213       {
00214           // Check if device name is set
00215           if (string.IsNullOrEmpty(device_name)) return false;
00216           return true; // Additional validations can be added as needed
00217       }
00218
00223       public string DeviceInfo()
00224       {
00225           // Return placeholder if no device is loaded
00226           if (string.IsNullOrEmpty(device_name))
00227               return "No device loaded";
00228
00229           // Build detailed device information string
00230           return $"Device Name: {device_name}\n" +
00231                  $"Logic Voltage: {logic_voltage}\n" +
00232                  $"Digital Inputs: [{string.Join(", ", digital_inputs)}]\n" +
00233                  $"Digital Inputs Names: [{string.Join(", ", digital_inputs_names)}]\n" +
00234                  $"Digital Outputs: [{string.Join(", ", digital_outputs)}]\n" +
00235                  $"Digital Outputs Names: [{string.Join(", ", digital_outputs_names)}]\n" +
00236                  $"Analog Inputs: [{string.Join(", ", analog_inputs)}]\n" +
00237                  $"Analog Inputs Names: [{string.Join(", ", analog_inputs_names)}]\n" +
00238                  $"DAC Outputs: [{string.Join(", ", dac_outputs)}]\n" +
00239                  $"DAC Outputs Names: [{string.Join(", ", dac_outputs_names)}]\n" +
00240                  $"One Wire Inputs: [{string.Join(", ", one_wire_inputs)}]\n" +
00241                  $"One Wire Inputs Names: [{FormatListOfLists(one_wire_inputs_names)}]\n" +
00242                  $"One Wire Inputs Devices Types: [{FormatListOfLists(one_wire_inputs_devices_types)}]\n" +
00243                  $"One Wire Inputs Devices Addresses: [{FormatListOfLists(one_wire_inputs_devices_addresses)}]\n" +
00244                  $"PWM Channels: {pwm_channels}\n" +
00245                  $"Max Hardware Timers: {max_hardware_timers}\n" +
00246                  $"Has RTOS: {has_rtos}\n" +
00247                  $"UART: [{string.Join(", ", UART)}]\n" +
00248                  $"I2C: [{string.Join(", ", I2C)}]\n" +
00249                  $"SPI: [{string.Join(", ", SPI)}]\n" +
00250                  $"USB: {USB}\n" +
00251                  $"Parent Devices: [{string.Join(", ", parent_devices)}]";
00252       }
00253
00259       private static string FormatListOfLists(List<List<string» listOfLists)
00260       {
00261           if (listOfLists == null)
00262               return "null";
00263
00264           // Join inner lists with commas and outer lists with semicolons
00265           return string.Join("; ",
00266               listOfLists.Select(innerList =>
00267                   $"[{string.Join(", ", innerList)}]"));
00268       }
00269   }
00270 }
```

7.23 ladder_diagram_app/Models/DeviceElement/DevicePinManager.cs File Reference

Classes

- class ladder_diagram_app.Models.DeviceElement.DevicePinManager

  Manages pin mapping options for a device, providing collections of available digital, analog, and one-wire pin names.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.DeviceElement

7.24 DevicePinManager.cs

Go to the documentation of this file.

```
00001 using System.Collections.ObjectModel;
00002
00003 namespace ladder_diagram_app.Models.DeviceElement
00004 {
00008     public class DevicePinManager
00009     {
00013         public ObservableCollection<string> DigitalInputOptions { get; }
00014
00018         public ObservableCollection<string> DigitalOutputOptions { get; }
00019
00023         public ObservableCollection<string> AnalogInputOptions { get; }
00024
00028         public ObservableCollection<string> AnalogOutputOptions { get; }
00029
00033         public ObservableCollection<string> OneWireInputOptions { get; }
00034
00038         public DevicePinManager()
00039         {
00040             DigitalInputOptions = [];
00041             DigitalOutputOptions = [];
00042             AnalogInputOptions = [];
00043             AnalogOutputOptions = [];
00044             OneWireInputOptions = [];
00045         }
00046
00051         public void UpdateDevicePinOptions(Device device)
00052         {
00053             // Clear existing pin options
00054             DigitalInputOptions.Clear();
00055             DigitalOutputOptions.Clear();
00056             AnalogInputOptions.Clear();
00057             AnalogOutputOptions.Clear();
00058             OneWireInputOptions.Clear();
00059
00060             // Populate digital input options
00061             foreach (var pin in device.digital_inputs_names)
00062                 DigitalInputOptions.Add(pin.ToString());
00063
00064             // Populate digital output options
00065             foreach (var pin in device.digital_outputs_names)
00066                 DigitalOutputOptions.Add(pin.ToString());
00067
00068             // Populate analog input options
00069             foreach (var pin in device.analog_inputs_names)
00070                 AnalogInputOptions.Add(pin.ToString());
00071
00072             // Populate analog output (DAC) options
00073             foreach (var pin in device.dac_outputs_names)
00074                 AnalogOutputOptions.Add(pin.ToString());
00075
00076             // Populate one-wire input options from nested lists
00077             foreach (var x in device.one_wire_inputs_names)
```

```
00078             {
00079                 foreach (var y in x)
00080                     OneWireInputOptions.Add(y.ToString());
00081             }
00082         }
00083     }
00084 }
```

7.25 ladder_diagram_app/Models/Variables/Instances/ADCSensorVariable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.ADCSensorVariable

  Represents an ADC sensor variable in a ladder diagram, encapsulating sensor-specific properties and validation.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.26 ADCSensorVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class ADCSensorVariable : Variable
00007     {
00011         private string? _sensorType;
00012
00016         private string? _pdSck;
00017
00021         private string? _dout;
00022
00026         private double _mapLow;
00027
00031         private double _mapHigh;
00032
00036         private double _gain;
00037
00041         private string? _samplingRate;
00042
00046         private string _value;
00047
00051         public string? SensorType
00052         {
00053             get => _sensorType;
00054             set => SetField(ref _sensorType, value);
00055         }
00056
00060         public string? PD_SCK
00061         {
00062             get => _pdSck;
00063             set => SetField(ref _pdSck, value);
00064         }
00065
00069         public string? DOUT
00070         {
00071             get => _dout;
00072             set => SetField(ref _dout, value);
00073         }
00074
00078         public double MapLow
```

```
00079         {
00080             get => _mapLow;
00081             set => SetField(ref _mapLow, value);
00082         }
00083
00087         public double MapHigh
00088         {
00089             get => _mapHigh;
00090             set => SetField(ref _mapHigh, value);
00091         }
00092
00096         public double Gain
00097         {
00098             get => _gain;
00099             set => SetField(ref _gain, value);
00100         }
00101
00105         public string? SamplingRate
00106         {
00107             get => _samplingRate;
00108             set => SetField(ref _samplingRate, value);
00109         }
00110
00114         public string Value
00115         {
00116             get => _value;
00117             set => SetField(ref _value, value);
00118         }
00119
00123         public ADCSensorVariable()
00124         {
00125             Type = "ADC Sensor";
00126             _value = " ";
00127             MapLow = 0.0;
00128             MapHigh = 100.0;
00129             Gain = 1.0;
00130         }
00131
00135         public bool IsValid => !string.IsNullOrEmpty(SensorType) &&
00136                                !string.IsNullOrEmpty(PD_SCK) &&
00137                                !string.IsNullOrEmpty(DOUT) &&
00138                                Gain >= 0 &&
00139                                !string.IsNullOrEmpty(SamplingRate);
00140
00145         public override Dictionary<string, object> ToExportDictionary()
00146         {
00147             return new Dictionary<string, object>
00148             {
00149                 ["Type"] = Type,
00150                 ["Name"] = Name,
00151                 ["Sensor Type"] = SensorType ?? string.Empty,
00152                 ["PD_SCK"] = PD_SCK ?? string.Empty,
00153                 ["DOUT"] = DOUT ?? string.Empty,
00154                 ["Map Low"] = MapLow,
00155                 ["Map High"] = MapHigh,
00156                 ["Gain"] = Gain,
00157                 ["Sampling Rate"] = SamplingRate ?? string.Empty
00158             };
00159         }
00160     }
00161 }
```

7.27 ladder_diagram_app/Models/Variables/Instances/BooleanVariable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.BooleanVariable

  Represents a boolean variable in a ladder diagram, encapsulating a true/false value.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.28 BooleanVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class BooleanVariable : Variable
00007     {
00011         private bool _value;
00012 
00016         public bool Value
00017         {
00018             get => _value;
00019             set => SetField(ref _value, value);
00020         }
00021 
00025         public BooleanVariable()
00026         {
00027             Type = "Boolean";
00028             Value = false; // Default value
00029         }
00030 
00035         public override Dictionary<string, object> ToExportDictionary()
00036         {
00037             return new Dictionary<string, object>
00038             {
00039                 ["Type"] = Type,
00040                 ["Name"] = Name,
00041                 ["Value"] = Value
00042             };
00043         }
00044     }
00045 }
```

7.29 ladder_diagram_app/Models/Variables/Instances/CounterVariable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.CounterVariable

  Represents a counter variable in a ladder diagram, encapsulating preset and current values, count direction, and output states.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.30 CounterVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class CounterVariable : Variable
00007     {
00011         private double _pv;
00012 
00016         private double _cv;
00017 
00021         private bool _cu;
00022 
00026         private bool _cd;
00027 
```

```
00031         private bool _qu;
00032
00036         private bool _qd;
00037
00041         private string _value;
00042
00046         public double PV
00047         {
00048             get => _pv;
00049             set => SetField(ref _pv, value);
00050         }
00051
00055         public double CV
00056         {
00057             get => _cv;
00058             set => SetField(ref _cv, value);
00059         }
00060
00064         public bool CU
00065         {
00066             get => _cu;
00067             set
00068             {
00069                 if (SetField(ref _cu, value) && value)
00070                     CD = false; // Ensure CD is false when CU is true
00071             }
00072         }
00073
00077         public bool CD
00078         {
00079             get => _cd;
00080             set
00081             {
00082                 if (SetField(ref _cd, value) && value)
00083                     CU = false; // Ensure CU is false when CD is true
00084             }
00085         }
00086
00090         public bool QU
00091         {
00092             get => _qu;
00093             set => SetField(ref _qu, value);
00094         }
00095
00099         public bool QD
00100         {
00101             get => _qd;
00102             set => SetField(ref _qd, value);
00103         }
00104
00108         public string Value
00109         {
00110             get => _value;
00111             set => SetField(ref _value, value);
00112         }
00113
00117         public CounterVariable()
00118         {
00119             Type = "Counter";
00120             _value = " "; // Default placeholder value
00121             CU = true; // Default to count up
00122         }
00123
00128         public override Dictionary<string, object> ToExportDictionary()
00129         {
00130             var dict = new Dictionary<string, object>
00131             {
00132                 ["Type"] = Type,
00133                 ["Name"] = Name,
00134                 ["PV"] = PV,
00135                 ["CU"] = CU,
00136                 ["CD"] = CD,
00137                 ["QU"] = QU,
00138                 ["QD"] = QD
00139             };
00140
00141             // Set CV based on count direction
00142             if (CU)
00143                 dict["CV"] = 0; // Start at 0 for count up
00144             else if (CD)
00145                 dict["CV"] = PV; // Start at preset value for count down
00146
00147             return dict;
00148         }
00149     }
00150 }
```

7.31 ladder_diagram_app/Models/Variables/Instances/DigitalAnalogInputOutputVariable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.DigitalAnalogInputOutputVariable
  Represents a digital or analog input/output variable in a ladder diagram, associated with a specific pin.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.32 DigitalAnalogInputOutputVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class DigitalAnalogInputOutputVariable : Variable
00007     {
00011         private string _pinName;
00012
00016         public DigitalAnalogInputOutputVariable()
00017         {
00018             _pinName = string.Empty;
00019         }
00020
00024         public string PinName
00025         {
00026             get => _pinName;
00027             set => SetField(ref _pinName, value);
00028         }
00029
00033         public bool IsValid => !string.IsNullOrEmpty(PinName);
00034
00039         public override Dictionary<string, object> ToExportDictionary()
00040         {
00041             return new Dictionary<string, object>
00042             {
00043                 ["Type"] = Type,
00044                 ["Name"] = Name,
00045                 ["Pin"] = PinName
00046             };
00047         }
00048     }
00049 }
```

7.33 ladder_diagram_app/Models/Variables/Instances/NumericVariable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.NumericVariable
  Represents a numeric variable in a ladder diagram, encapsulating a double-precision value.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.34 NumericVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class NumericVariable : Variable
00007     {
00011         private double _value;
00012 
00016         public double Value
00017         {
00018             get => _value;
00019             set => SetField(ref _value, value);
00020         }
00021 
00025         public NumericVariable()
00026         {
00027             Type = "Number";
00028             Value = 0.0; // Default value
00029         }
00030 
00035         public override Dictionary<string, object> ToExportDictionary()
00036         {
00037             return new Dictionary<string, object>
00038             {
00039                 ["Type"] = Type,
00040                 ["Name"] = Name,
00041                 ["Value"] = Value
00042             };
00043         }
00044     }
00045 }
```

7.35 ladder_diagram_app/Models/Variables/Instances/OneWireInput↩ Variable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.OneWireInputVariable

  Represents a one-wire input variable in a ladder diagram, associated with a specific pin.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.36 OneWireInputVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class OneWireInputVariable : Variable
00007     {
00011         private string _pinName;
00012
00016         public OneWireInputVariable()
00017         {
00018             _pinName = string.Empty;
00019         }
00020
00024         public string PinName
00025         {
00026             get => _pinName;
00027             set => SetField(ref _pinName, value);
00028         }
00029
00033         public bool IsValid => !string.IsNullOrEmpty(PinName);
00034
00039         public override Dictionary<string, object> ToExportDictionary()
00040         {
00041             return new Dictionary<string, object>
00042             {
00043                 ["Type"] = Type,
00044                 ["Name"] = Name,
00045                 ["Pin"] = PinName
00046             };
00047         }
00048     }
00049 }
```

7.37 ladder_diagram_app/Models/Variables/Instances/TimerVariable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.TimerVariable

  Represents a timer variable in a ladder diagram, encapsulating preset time, elapsed time, input, and output states.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.38 TimerVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class TimerVariable : Variable
00007     {
00011         private double _pt;
00012
00016         private double _et;
00017
00021         private bool _in;
00022
00026         private bool _q;
```

```
00027
00031         private string _value;
00032
00036         public double PT
00037         {
00038             get => _pt;
00039             set => SetField(ref _pt, value);
00040         }
00041
00045         public double ET
00046         {
00047             get => _et;
00048             set => SetField(ref _et, value);
00049         }
00050
00054         public bool IN
00055         {
00056             get => _in;
00057             set => SetField(ref _in, value);
00058         }
00059
00063         public bool Q
00064         {
00065             get => _q;
00066             set => SetField(ref _q, value);
00067         }
00068
00072         public string Value
00073         {
00074             get => _value;
00075             set => SetField(ref _value, value);
00076         }
00077
00081         public TimerVariable()
00082         {
00083             Type = "Timer";
00084             _value = " "; // Default placeholder value
00085         }
00086
00090         public bool IsValid => PT >= 0;
00091
00096         public override Dictionary<string, object> ToExportDictionary()
00097         {
00098             return new Dictionary<string, object>
00099             {
00100                 ["Type"] = Type,
00101                 ["Name"] = Name,
00102                 ["PT"] = PT,
00103                 ["ET"] = ET,
00104                 ["IN"] = IN,
00105                 ["Q"] = Q
00106             };
00107         }
00108     }
00109 }
```

7.39 ladder_diagram_app/Models/Variables/Instances/TimeVariable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.TimeVariable

  Represents a time variable in a ladder diagram, encapsulating a double-precision time value.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.40 TimeVariable.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Models.Variables.Instances
00002 {
00006     public class TimeVariable : Variable
00007     {
00011         private double _value;
00012
00016         public double Value
00017         {
00018             get => _value;
00019             set => SetField(ref _value, value);
00020         }
00021
00025         public TimeVariable()
00026         {
00027             Type = "Time";
00028             Value = 0.0; // Default value
00029         }
00030
00035         public override Dictionary<string, object> ToExportDictionary()
00036         {
00037             return new Dictionary<string, object>
00038             {
00039                 ["Type"] = Type,
00040                 ["Name"] = Name,
00041                 ["Value"] = Value
00042             };
00043         }
00044     }
00045 }
```

7.41 ladder_diagram_app/Models/Variables/Instances/Variable.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.Instances.Variable

  Abstract base class for variables in a ladder diagram, providing common properties and change notification.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables
- namespace ladder_diagram_app.Models.Variables.Instances

7.42 Variable.cs

Go to the documentation of this file.

```
00001 using System.ComponentModel;
00002 using System.Runtime.CompilerServices;
00003
00004 namespace ladder_diagram_app.Models.Variables.Instances
00005 {
00009     public abstract class Variable : INotifyPropertyChanged
00010     {
00014         private string _name;
00015
00019         private string _type;
00020
00024         protected Variable()
```

```
00025         {
00026             _name = string.Empty;
00027             _type = string.Empty;
00028         }
00029
00033         public string Name
00034         {
00035             get => _name;
00036             set => SetField(ref _name, value);
00037         }
00038
00042         public string Type
00043         {
00044             get => _type;
00045             set => SetField(ref _type, value);
00046         }
00047
00051         public bool IsDeletable { get; set; } = true;
00052
00056         public event PropertyChangedEventHandler? PropertyChanged;
00057
00062         protected virtual void OnPropertyChanged([CallerMemberName] string? propertyName = null)
00063         {
00064             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
00065         }
00066
00075         protected bool SetField<T>(ref T field, T value, [CallerMemberName] string? propertyName = null)
00076         {
00077             if (EqualityComparer<T>.Default.Equals(field, value)) return false;
00078             field = value;
00079             OnPropertyChanged(propertyName);
00080             return true;
00081         }
00082
00087         public abstract Dictionary<string, object> ToExportDictionary();
00088     }
00089 }
```

7.43 ladder_diagram_app/Models/Variables/VariablesManager.cs File Reference

Classes

- class ladder_diagram_app.Models.Variables.VariablesManager

  Manages variables in a ladder diagram application, associating them with a device and maintaining lists for UI components.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Models
- namespace ladder_diagram_app.Models.Variables

7.44 VariablesManager.cs

Go to the documentation of this file.

```
00001 using System.Windows;
00002 using System.Collections.ObjectModel;
00003 using System.Collections.Specialized;
00004 using ladder_diagram_app.Models.Variables.Instances;
00005 using ladder_diagram_app.Models.DeviceElement;
00006 using System.Globalization;
00007 using ladder_diagram_app.Views;
00008 using System.Diagnostics;
00009
00010 namespace ladder_diagram_app.Models.Variables
00011 {
```

```
00015    public class VariablesManager
00016    {
00020        public Device Device { get; set; }
00021
00025        public ObservableCollection<Variable> VariablesList { get; } = [];
00026
00030        public ObservableCollection<string> VariablesListContacts { get; } = [];
00031
00035        public ObservableCollection<string> VariablesListCoils { get; } = [];
00036
00040        public ObservableCollection<string> VariablesListMath { get; } = [];
00041
00045        public ObservableCollection<string> VariablesListCompare { get; } = [];
00046
00050        public ObservableCollection<string> VariablesListCounter { get; } = [];
00051
00055        public ObservableCollection<string> VariablesListTimer { get; } = [];
00056
00060        public ObservableCollection<string> VariablesListReset { get; } = [];
00061
00066        public VariablesManager(Device device)
00067        {
00068            Device = device;
00069            VariablesList.CollectionChanged += VariablesList_CollectionChanged;
00070        }
00071
00072        // ==================== VARIABLE LIST MANAGEMENT ======================
00076        public void ClearVariablesList()
00077        {
00078            VariablesList.Clear();
00079            VariablesListContacts.Clear();
00080            VariablesListCoils.Clear();
00081            VariablesListMath.Clear();
00082            VariablesListCompare.Clear();
00083            VariablesListCounter.Clear();
00084            VariablesListTimer.Clear();
00085            VariablesListReset.Clear();
00086        }
00087
00111        public void AddVariable(string name, string type, Window owner,
00112                                string pinName = "",
00113                                string sensorType = "", string pdSck = "", string dout = "", string samplingRate = "", double
     mapLow = 0.0, double mapHigh = 100.0, double gain = 1.0,
00114                                bool boolValue = false,
00115                                double numValue = 0.0,
00116                                double pv = 0.0, double cv = 0.0, bool cu = true, bool cd = false,
00117                                double pt = 0.0, double et = 0.0,
00118                                double timeValue = 0.0)
00119        {
00120            if (string.IsNullOrWhiteSpace(name))
00121            {
00122                new NotificationWindow("A Variable must have a name", owner).Show();
00123                return;
00124            }
00125            if (VariablesList.Any(v => v.Name.Equals(name, StringComparison.OrdinalIgnoreCase)))
00126            {
00127                new NotificationWindow("A Variable with the same name already exists", owner).Show();
00128                return;
00129            }
00130
00131            Variable? newVariable = null;
00132
00133            switch (type)
00134            {
00135                case "Digital Input":
00136                    if (VariablesList.Count(v => v.Type == "Digital Input") == Device.digital_inputs_names.Count)
00137                    {
00138                        if (Device.digital_inputs_names.Count() == 0)
00139                        {
00140                            new NotificationWindow("The device has no digital inputs", owner).Show();
00141                            return;
00142                        }
00143                        else
00144                        {
00145                            new NotificationWindow("The device has no more digital inputs", owner).Show();
00146                            return;
00147                        }
00148                    }
00149                    newVariable = new DigitalAnalogInputOutputVariable { Name = name, Type = type, PinName = pinName
     };
00150                    break;
00151                case "Digital Output":
00152                    if (VariablesList.Count(v => v.Type == "Digital Output") == Device.digital_outputs_names.Count)
00153                    {
00154                        if (Device.digital_outputs_names.Count() == 0)
00155                        {
00156                            new NotificationWindow("The device has no digital outputs", owner).Show();
```

```
00157                     return;
00158                 }
00159                 else
00160                 {
00161                     new NotificationWindow("The device has no more digital outputs", owner).Show();
00162                     return;
00163                 }
00164             }
00165             newVariable = new DigitalAnalogInputOutputVariable { Name = name, Type = type, PinName = pinName
};
00166             break;
00167         case "Analog Input":
00168             if (VariablesList.Count(v => v.Type == "Analog Input") == Device.analog_inputs_names.Count)
00169             {
00170                 if (Device.analog_inputs_names.Count() == 0)
00171                 {
00172                     new NotificationWindow("The device has no analog inputs", owner).Show();
00173                     return;
00174                 }
00175                 else
00176                 {
00177                     new NotificationWindow("The device has no more analog inputs", owner).Show();
00178                     return;
00179                 }
00180             }
00181             newVariable = new DigitalAnalogInputOutputVariable { Name = name, Type = type, PinName = pinName
};
00182             break;
00183         case "Analog Output":
00184             if (VariablesList.Count(v => v.Type == "Analog Output") == Device.dac_outputs_names.Count)
00185             {
00186                 if (Device.dac_outputs_names.Count == 0)
00187                 {
00188                     new NotificationWindow("The device has no analog outputs", owner).Show();
00189                     return;
00190                 }
00191                 else
00192                 {
00193                     new NotificationWindow("The device has no more analog outputs", owner).Show();
00194                     return;
00195                 }
00196             }
00197             newVariable = new DigitalAnalogInputOutputVariable { Name = name, Type = type, PinName = pinName
};
00198             break;
00199         case "One Wire Input":
00200             if (VariablesList.Count(v => v.Type == "One Wire Input") >=
Device.one_wire_inputs_names.Sum(innerList => innerList?.Count ?? 0))
00201             {
00202                 if (Device.one_wire_inputs_names.Count == 0)
00203                 {
00204                     new NotificationWindow("The device has no one wire inputs", owner).Show();
00205                     return;
00206                 }
00207                 else
00208                 {
00209                     new NotificationWindow("The device has no more one wire inputs", owner).Show();
00210                     return;
00211                 }
00212             }
00213             newVariable = new OneWireInputVariable { Name = name, Type = type, PinName = pinName };
00214             break;
00215         case "ADC Sensor":
00216             if (Device.digital_outputs_names.Count == 0 || Device.digital_inputs_names.Count == 0)
00217             {
00218                 new NotificationWindow("The device has no digital inputs or outputs to use for ADC sensor",
owner).Show();
00219                 return;
00220             }
00221             newVariable = new ADCSensorVariable { Name = name, Type = type, SensorType = sensorType, PD_SCK
= pdSck, DOUT = dout, MapLow = mapLow, MapHigh = mapHigh, Gain = gain, SamplingRate = samplingRate };
00222             break;
00223         case "Boolean":
00224             newVariable = new BooleanVariable { Name = name, Type = type, Value = boolValue };
00225             break;
00226         case "Number":
00227             newVariable = new NumericVariable { Name = name, Type = type, Value = numValue };
00228             break;
00229         case "Counter":
00230             newVariable = new CounterVariable { Name = name, Type = type, PV = pv, CV = cv, CU = cu, CD = cd };
00231             break;
00232         case "Timer":
00233             newVariable = new TimerVariable { Name = name, Type = type, PT = pt, ET = et };
00234             break;
00235         case "Current Time":
00236             if (VariablesList.Count(v => v.Type == "Current Time") == 1)
00237             {
```

```
00238                   new NotificationWindow("There is already a variable that represents current time", owner).Show();
00239                   return;
00240               }
00241               newVariable = new TimeVariable { Name = name, Type = type };
00242               break;
00243           case "Time":
00244               newVariable = new TimeVariable { Name = name, Type = type, Value = timeValue };
00245               break;
00246       }
00247
00248       if (newVariable != null)
00249       {
00250           VariablesList.Add(newVariable);
00251           return;
00252       }
00253
00254       new NotificationWindow($"Failed to add variable '{name}' of type '{type}'", owner).Show();
00255       return;
00256   }
00257
00263   public void DeleteVariable(Variable variable, Window owner)
00264   {
00265       if (!variable.IsDeletable)
00266       {
00267           new NotificationWindow("This item cannot be deleted", owner).Show();
00268           return;
00269       }
00270
00271       // Handle expanded ADC Sensor properties
00272       if (variable is ADCSensorVariable adcs && adcs.Value == " ")
00273       {
00274           int index = VariablesList.IndexOf(variable);
00275           for (int i = 0; i < 7; i++)
00276               VariablesList.RemoveAt(index + 1);
00277       }
00278       // Handle expanded Counter properties
00279       else if (variable is CounterVariable c && c.Value == " ")
00280       {
00281           int index = VariablesList.IndexOf(variable);
00282           for (int i = 0; i < 6; i++)
00283               VariablesList.RemoveAt(index + 1);
00284       }
00285       // Handle expanded Timer properties
00286       else if (variable is TimerVariable t && t.Value == " ")
00287       {
00288           int index = VariablesList.IndexOf(variable);
00289           for (int i = 0; i < 4; i++)
00290               VariablesList.RemoveAt(index + 1);
00291       }
00292
00293       VariablesList.Remove(variable);
00294   }
00295
00300   public void VariableBooleanClick(Variable variable)
00301   {
00302       int index = VariablesList.IndexOf(variable);
00303
00304       if (variable is BooleanVariable b)
00305       {
00306           b.Value = !b.Value;
00307           // Update Counter properties if CU or CD is toggled
00308           if (variable.Name == "      CU" || variable.Name == "      CD")
00309           {
00310               for (int i = index - 1; i >= 0; i--)
00311               {
00312                   if (VariablesList[i].Type == "Counter" && VariablesList[i] is CounterVariable cnt)
00313                   {
00314                       if (variable.Name == "      CU")
00315                       {
00316                           cnt.CU = b.Value;
00317                           cnt.CD = !b.Value;
00318                           if (VariablesList[index + 1] is BooleanVariable cdBool)
00319                               cdBool.Value = !b.Value;
00320                       }
00321                       else if (variable.Name == "      CD")
00322                       {
00323                           cnt.CD = b.Value;
00324                           cnt.CU = !b.Value;
00325                           if (VariablesList[index - 1] is BooleanVariable cdBool)
00326                               cdBool.Value = !b.Value;
00327                       }
00328                       break;
00329                   }
00330               }
00331           }
00332       }
00333
```

```
00334           // Expand or collapse ADC Sensor properties
00335           if (variable is ADCSensorVariable adcs)
00336           {
00337               if (adcs.Value == " ")
00338               {
00339                   VariablesList.Insert(index + 1, new DigitalAnalogInputOutputVariable { Name = "     Sensor Type", Type
= "ADC Sensor Type", PinName = adcs.SensorType ?? string.Empty, IsDeletable = false });
00340                   VariablesList.Insert(index + 2, new DigitalAnalogInputOutputVariable { Name = "     PD_SCK", Type =
"ADC Sensor Digital Output", PinName = adcs.PD_SCK ?? string.Empty, IsDeletable = false });
00341                   VariablesList.Insert(index + 3, new DigitalAnalogInputOutputVariable { Name = "     DOUT", Type =
"ADC Sensor Digital Input", PinName = adcs.DOUT ?? string.Empty, IsDeletable = false });
00342                   VariablesList.Insert(index + 4, new NumericVariable { Name = "     Map Low", Type = "Number", Value =
adcs.MapLow, IsDeletable = false });
00343                   VariablesList.Insert(index + 5, new NumericVariable { Name = "     Map High", Type = "Number", Value
= adcs.MapHigh, IsDeletable = false });
00344                   VariablesList.Insert(index + 6, new NumericVariable { Name = "     Gain", Type = "Number", Value =
adcs.Gain, IsDeletable = false });
00345                   VariablesList.Insert(index + 7, new DigitalAnalogInputOutputVariable { Name = "     Sampling Rate",
Type = "ADC Sensor Sampling Rate", PinName = adcs.SamplingRate ?? string.Empty, IsDeletable = false });
00346                   adcs.Value = " ";
00347               }
00348               else
00349               {
00350                   for (int i = index + 1; i <= index + 7; i++)
00351                       VariablesList.RemoveAt(index + 1);
00352                   adcs.Value = " ";
00353               }
00354           }
00355
00356           // Expand or collapse Counter properties
00357           if (variable is CounterVariable c)
00358           {
00359               if (c.Value == " ")
00360               {
00361                   VariablesList.Insert(index + 1, new NumericVariable { Name = "     PV", Type = "Number", Value = c.PV,
IsDeletable = false });
00362                   VariablesList.Insert(index + 2, new NumericVariable { Name = "     CV", Type = "Number ", Value =
c.CV, IsDeletable = false });
00363                   VariablesList.Insert(index + 3, new BooleanVariable { Name = "     CU", Type = "Boolean", Value = c.CU,
IsDeletable = false });
00364                   VariablesList.Insert(index + 4, new BooleanVariable { Name = "     CD", Type = "Boolean", Value = c.CD,
IsDeletable = false });
00365                   VariablesList.Insert(index + 5, new BooleanVariable { Name = "     QU", Type = "Boolean ", IsDeletable =
false });
00366                   VariablesList.Insert(index + 6, new BooleanVariable { Name = "     QD", Type = "Boolean ", IsDeletable =
false });
00367                   c.Value = " ";
00368               }
00369               else
00370               {
00371                   for (int i = index + 1; i <= index + 6; i++)
00372                       VariablesList.RemoveAt(index + 1);
00373                   c.Value = " ";
00374               }
00375           }
00376
00377           // Expand or collapse Timer properties
00378           if (variable is TimerVariable t)
00379           {
00380               if (t.Value == " ")
00381               {
00382                   VariablesList.Insert(index + 1, new TimeVariable { Name = "     PT", Type = "Number", Value = t.PT,
IsDeletable = false });
00383                   VariablesList.Insert(index + 2, new TimeVariable { Name = "     ET", Type = "Number ", Value = t.ET,
IsDeletable = false });
00384                   VariablesList.Insert(index + 3, new BooleanVariable { Name = "     IN", Type = "Boolean ", IsDeletable =
false });
00385                   VariablesList.Insert(index + 4, new BooleanVariable { Name = "     Q", Type = "Boolean ", IsDeletable =
false });
00386                   t.Value = " ";
00387               }
00388               else
00389               {
00390                   for (int i = index + 1; i <= index + 4; i++)
00391                       VariablesList.RemoveAt(index + 1);
00392                   t.Value = " ";
00393               }
00394           }
00395       }
00396
00402       public void VariableTextBoxChange(Variable variable, string inputText)
00403       {
00404           if (!double.TryParse(inputText, NumberStyles.AllowLeadingSign | NumberStyles.AllowDecimalPoint |
NumberStyles.AllowThousands, CultureInfo.InvariantCulture, out double parsedValue))
00405               return;
00406
00407           int index = VariablesList.IndexOf(variable);
```

```
00408
00409           if (variable is NumericVariable n)
00410               n.Value = parsedValue;
00411           else if (variable is TimeVariable t)
00412               t.Value = parsedValue;
00413
00414           // Update ADC Sensor parameters
00415           if (variable.Name == "     Map Low")
00416           {
00417               for (int i = index - 1; i >= 0; i--)
00418               {
00419                   if (VariablesList[i].Type == "ADC Sensor" && VariablesList[i] is ADCSensorVariable adcs)
00420                   {
00421                       adcs.MapLow = parsedValue;
00422                       Debug.WriteLine($"Updating Map Low to : {adcs.MapLow}");
00423                       break;
00424                   }
00425               }
00426           }
00427           else if (variable.Name == "     Map High")
00428           {
00429               for (int i = index - 1; i >= 0; i--)
00430               {
00431                   if (VariablesList[i].Type == "ADC Sensor" && VariablesList[i] is ADCSensorVariable adcs)
00432                   {
00433                       adcs.MapHigh = parsedValue;
00434                       break;
00435                   }
00436               }
00437           }
00438           else if (variable.Name == "     Gain")
00439           {
00440               for (int i = index - 1; i >= 0; i--)
00441               {
00442                   if (VariablesList[i].Type == "ADC Sensor" && VariablesList[i] is ADCSensorVariable adcs)
00443                   {
00444                       adcs.Gain = parsedValue;
00445                       break;
00446                   }
00447               }
00448           }
00449
00450           // Update Counter parameters
00451           else if (variable.Name == "     PV")
00452           {
00453               for (int i = index - 1; i >= 0; i--)
00454               {
00455                   if (VariablesList[i].Type == "Counter" && VariablesList[i] is CounterVariable cnt)
00456                   {
00457                       cnt.PV = parsedValue;
00458                       break;
00459                   }
00460               }
00461           }
00462
00463           // Update Timer parameters
00464           else if (variable.Name == "     PT")
00465           {
00466               for (int i = index - 1; i >= 0; i--)
00467               {
00468                   if (VariablesList[i].Type == "Timer" && VariablesList[i] is TimerVariable tmr)
00469                   {
00470                       tmr.PT = parsedValue;
00471                       break;
00472                   }
00473               }
00474           }
00475       }
00476
00482       public void VariableComboBoxChange(Variable variable, string selectedValue)
00483       {
00484           int index = VariablesList.IndexOf(variable);
00485
00486           // Update ADC Sensor parameters
00487           if (variable.Name == "     Sensor Type")
00488           {
00489               for (int i = index - 1; i >= 0; i--)
00490               {
00491                   if (VariablesList[i].Type == "ADC Sensor" && VariablesList[i] is ADCSensorVariable adcs)
00492                   {
00493                       adcs.SensorType = selectedValue;
00494                       break;
00495                   }
00496               }
00497           }
00498           else if (variable.Name == "     PD_SCK")
00499           {
```

```
00500             for (int i = index - 1; i >= 0; i--)
00501             {
00502                 if (VariablesList[i].Type == "ADC Sensor" && VariablesList[i] is ADCSensorVariable adcs)
00503                 {
00504                     adcs.PD_SCK = selectedValue;
00505                     break;
00506                 }
00507             }
00508         }
00509         else if (variable.Name == "     DOUT")
00510         {
00511             for (int i = index - 1; i >= 0; i--)
00512             {
00513                 if (VariablesList[i].Type == "ADC Sensor" && VariablesList[i] is ADCSensorVariable adcs)
00514                 {
00515                     adcs.DOUT = selectedValue;
00516                     break;
00517                 }
00518             }
00519         }
00520         else if (variable.Name == "     Sampling Rate")
00521         {
00522             for (int i = index - 1; i >= 0; i--)
00523             {
00524                 if (VariablesList[i].Type == "ADC Sensor" && VariablesList[i] is ADCSensorVariable adcs)
00525                 {
00526                     adcs.SamplingRate = selectedValue;
00527                     break;
00528                 }
00529             }
00530         }
00531 
00532         // Update PinName for DigitalAnalogInputOutput variables
00533         if (variable is DigitalAnalogInputOutputVariable daio)
00534         {
00535             daio.PinName = selectedValue;
00536         }
00537     }
00538 
00539     // ==================== COMBOBOX VALUE TRACKING ======================
00545     private void VariablesList_CollectionChanged(object? sender, NotifyCollectionChangedEventArgs e)
00546     {
00547         if (e.Action == NotifyCollectionChangedAction.Add)
00548         {
00549             if (e.NewItems != null)
00550             {
00551                 foreach (Variable variable in e.NewItems)
00552                 {
00553                     AddVariableToCollections(variable);
00554                 }
00555             }
00556         }
00557         else if (e.Action == NotifyCollectionChangedAction.Remove)
00558         {
00559             if (e.OldItems != null)
00560             {
00561                 foreach (Variable variable in e.OldItems)
00562                 {
00563                     RemoveVariableFromCollections(variable);
00564                 }
00565             }
00566         }
00567     }
00568 
00573     private void AddVariableToCollections(Variable variable)
00574     {
00575         if (variable.Type == "Digital Input")
00576         {
00577             VariablesListContacts.Add(variable.Name);
00578         }
00579         else if (variable.Type == "Digital Output")
00580         {
00581             VariablesListContacts.Add(variable.Name);
00582             VariablesListCoils.Add(variable.Name);
00583         }
00584         else if (variable.Type == "Analog Input" || variable.Type == "Analog Output")
00585         {
00586             VariablesListMath.Add(variable.Name);
00587             VariablesListCompare.Add(variable.Name);
00588         }
00589         else if (variable.Type == "One Wire Input")
00590         {
00591             VariablesListMath.Add(variable.Name);
00592             VariablesListCompare.Add(variable.Name);
00593         }
00594         else if (variable.Type == "ADC Sensor")
00595         {
```

```
00596                 VariablesListMath.Add(variable.Name);
00597                 VariablesListCompare.Add(variable.Name);
00598             }
00599             else if (variable.Type == "Boolean" && !variable.Name.Contains("     "))
00600             {
00601                 VariablesListContacts.Add(variable.Name);
00602                 VariablesListCoils.Add(variable.Name);
00603             }
00604             else if (variable.Type == "Number" && !variable.Name.Contains("     "))
00605             {
00606                 VariablesListMath.Add(variable.Name);
00607                 VariablesListCompare.Add(variable.Name);
00608             }
00609             else if (variable.Type == "Counter" && !variable.Name.Contains("     "))
00610             {
00611                 VariablesListContacts.Add($"{variable.Name}.QU");
00612                 VariablesListContacts.Add($"{variable.Name}.QD");
00613                 VariablesListMath.Add($"{variable.Name}.PV");
00614                 VariablesListMath.Add($"{variable.Name}.CV");
00615                 VariablesListCompare.Add($"{variable.Name}.PV");
00616                 VariablesListCompare.Add($"{variable.Name}.CV");
00617                 VariablesListCounter.Add(variable.Name);
00618                 VariablesListReset.Add(variable.Name);
00619             }
00620             else if (variable.Type == "Timer")
00621             {
00622                 VariablesListContacts.Add($"{variable.Name}.IN");
00623                 VariablesListContacts.Add($"{variable.Name}.Q");
00624                 VariablesListMath.Add($"{variable.Name}.PT");
00625                 VariablesListMath.Add($"{variable.Name}.ET");
00626                 VariablesListCompare.Add($"{variable.Name}.PT");
00627                 VariablesListCompare.Add($"{variable.Name}.ET");
00628                 VariablesListTimer.Add(variable.Name);
00629                 VariablesListReset.Add(variable.Name);
00630             }
00631             else if (variable.Type == "Current Time" && !variable.Name.Contains("     "))
00632             {
00633                 VariablesListCompare.Add(variable.Name);
00634             }
00635             else if (variable.Type == "Time" && !variable.Name.Contains("     "))
00636             {
00637                 VariablesListCompare.Add(variable.Name);
00638             }
00639         }
00640 
00645         private void RemoveVariableFromCollections(Variable variable)
00646         {
00647             if (variable.Type == "Digital Input")
00648             {
00649                 VariablesListContacts.Remove(variable.Name);
00650             }
00651             else if (variable.Type == "Digital Output")
00652             {
00653                 VariablesListContacts.Remove(variable.Name);
00654                 VariablesListCoils.Remove(variable.Name);
00655             }
00656             else if (variable.Type == "Analog Input" || variable.Type == "Analog Output")
00657             {
00658                 VariablesListMath.Remove(variable.Name);
00659                 VariablesListCompare.Remove(variable.Name);
00660             }
00661             else if (variable.Type == "One Wire Input")
00662             {
00663                 VariablesListMath.Remove(variable.Name);
00664                 VariablesListCompare.Remove(variable.Name);
00665             }
00666             else if (variable.Type == "ADC Sensor")
00667             {
00668                 VariablesListMath.Remove(variable.Name);
00669                 VariablesListCompare.Remove(variable.Name);
00670             }
00671             else if (variable.Type == "Boolean" && !variable.Name.Contains("     "))
00672             {
00673                 VariablesListContacts.Remove(variable.Name);
00674                 VariablesListCoils.Remove(variable.Name);
00675             }
00676             else if (variable.Type == "Number" && !variable.Name.Contains("     "))
00677             {
00678                 VariablesListMath.Remove(variable.Name);
00679                 VariablesListCompare.Remove(variable.Name);
00680             }
00681             else if (variable.Type == "Counter" && !variable.Name.Contains("     "))
00682             {
00683                 VariablesListContacts.Remove($"{variable.Name}.QU");
00684                 VariablesListContacts.Remove($"{variable.Name}.QD");
00685                 VariablesListMath.Remove($"{variable.Name}.PV");
00686                 VariablesListMath.Remove($"{variable.Name}.CV");
```

```
00687                 VariablesListCompare.Remove($"{variable.Name}.PV");
00688                 VariablesListCompare.Remove($"{variable.Name}.CV");
00689                 VariablesListCounter.Remove(variable.Name);
00690                 VariablesListReset.Remove(variable.Name);
00691             }
00692             else if (variable.Type == "Timer")
00693             {
00694                 VariablesListContacts.Remove($"{variable.Name}.IN");
00695                 VariablesListContacts.Remove($"{variable.Name}.Q");
00696                 VariablesListMath.Remove($"{variable.Name}.PT");
00697                 VariablesListMath.Remove($"{variable.Name}.ET");
00698                 VariablesListCompare.Remove($"{variable.Name}.PT");
00699                 VariablesListCompare.Remove($"{variable.Name}.ET");
00700                 VariablesListTimer.Remove(variable.Name);
00701                 VariablesListReset.Remove(variable.Name);
00702             }
00703             else if (variable.Type == "Current Time" && !variable.Name.Contains("     "))
00704             {
00705                 VariablesListCompare.Remove(variable.Name);
00706             }
00707             else if (variable.Type == "Time" && !variable.Name.Contains("     "))
00708             {
00709                 VariablesListCompare.Remove(variable.Name);
00710             }
00711         }
00712 
00713         // ==================== VARIABLE VALIDATION FOR EXPORT ====================
00719         public bool ValidateVariables(Window owner)
00720         {
00721             foreach (var variable in VariablesList)
00722             {
00723                 if (variable.Type == "Digital Input" ||
00724                     variable.Type == "Digital Output" ||
00725                     variable.Type == "Analog Input" ||
00726                     variable.Type == "Analog Output")
00727                 {
00728                     if (variable is DigitalAnalogInputOutputVariable daio && !daio.IsValid)
00729                     {
00730                         new NotificationWindow("All Analog/Digital Input/Output variables must have a mapped value
    selected", owner).Show();
00731                         return false;
00732                     }
00733                 }
00734                 else if (variable.Type == "One Wire Input")
00735                 {
00736                     if (variable is OneWireInputVariable owi && !owi.IsValid)
00737                     {
00738                         new NotificationWindow("All One Wire Input variables must have a mapped value selected",
    owner).Show();
00739                         return false;
00740                     }
00741                 }
00742                 else if (variable.Type == "ADC Sensor")
00743                 {
00744                     if (variable is ADCSensorVariable adcs && !adcs.IsValid)
00745                     {
00746                       new NotificationWindow("All ADC Sensor parameters must have a mapped value selected", owner).Show();
00747                         return false;
00748                     }
00749                 }
00750                 else if (variable.Type == "Timer")
00751                 {
00752                     if (variable is TimerVariable tmr && !tmr.IsValid)
00753                     {
00754                         new NotificationWindow("Timers Preset Time must be >= 0", owner).Show();
00755                         return false;
00756                     }
00757                 }
00758             }
00759 
00760             return true;
00761         }
00762     }
00763 }
```

7.45 ladder_diagram_app/Services/CanvasServices/CanvasElement↩ Finder.cs File Reference

Classes

- class ladder_diagram_app.Services.CanvasServices.CanvasElementFinder

Provides methods to find canvas elements (wires, ladder elements, and branches) based on cursor position in a ladder diagram application.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CanvasServices

7.46 CanvasElementFinder.cs

Go to the documentation of this file.

```
00001 using System.Windows;
00002 using System.Windows.Controls;
00003 using System.Windows.Shapes;
00004 using ladder_diagram_app.Models.CanvasElements;
00005 using ladder_diagram_app.Models.CanvasElements.Instances;
00006
00007 namespace ladder_diagram_app.Services.CanvasServices
00008 {
00012     public class CanvasElementFinder
00013     {
00014         private readonly Func<WiresManager> _getWiresManager;
00015
00021         public CanvasElementFinder(Func<WiresManager> getWiresManager)
00022         {
00023             _getWiresManager = getWiresManager ?? throw new ArgumentNullException(nameof(getWiresManager));
00024         }
00025
00031         public Wire? FindClosestWire(Point cursorPosition)
00032         {
00033             var wiresManager = _getWiresManager();
00034             var closestWire = wiresManager.Wires.MinBy(w => Math.Abs(w.Y - cursorPosition.Y));
00035             return closestWire != null && Math.Abs(closestWire.Y - cursorPosition.Y) <= 20 ? closestWire : null;
00036         }
00037
00043         public LadderElement? FindClosestElement(Point cursorPosition)
00044         {
00045             var wiresManager = _getWiresManager();
00046
00047             LadderElement? FindElementInBounds(List<Node> nodes)
00048             {
00049                 foreach (var node in nodes)
00050                 {
00051                     if (node is LadderElement element)
00052                     {
00053                         if (element.Image == null) continue;
00054
00055                         double left = Canvas.GetLeft(element.Image);
00056                         double top = Canvas.GetTop(element.Image);
00057                         double right = left + element.Image.Width;
00058                         double bottom = top + element.Image.Height;
00059
00060                         if (cursorPosition.X >= left && cursorPosition.X <= right &&
00061                             cursorPosition.Y >= top && cursorPosition.Y <= bottom)
00062                         {
00063                             return element;
00064                         }
00065                     }
00066                     else if (node is Branch branch)
00067                     {
00068                         var foundInNodes1 = FindElementInBounds(branch.Nodes1);
00069                         if (foundInNodes1 != null)
00070                         {
00071                             return foundInNodes1;
00072                         }
00073                         var foundInNodes2 = FindElementInBounds(branch.Nodes2);
00074                         if (foundInNodes2 != null)
00075                         {
00076                             return foundInNodes2;
00077                         }
00078                     }
00079                 }
00080                 return null;
00081             }
00082
```

```
00083           foreach (var wire in wiresManager.Wires)
00084           {
00085               var foundElement = FindElementInBounds(wire.Nodes);
00086               if (foundElement != null)
00087               {
00088                   return foundElement;
00089               }
00090           }
00091           return null;
00092       }
00093
00100       public (Branch? ClosestBranch, bool IsUpperLine) FindClosestBranch(Point cursorPosition, Branch? selectedBranch
      = null)
00101       {
00102           var wiresManager = _getWiresManager();
00103           var allBranches = new List<(Branch Branch, int Depth)>();
00104
00105           // Recursively collect all branches with their nesting depth
00106           void CollectBranches(List<Node> nodes, int depth = 0)
00107           {
00108               foreach (var node in nodes)
00109               {
00110                   if (node is Branch branch)
00111                   {
00112                       allBranches.Add((branch, depth));
00113                       CollectBranches(branch.Nodes1, depth + 1);
00114                       CollectBranches(branch.Nodes2, depth + 1);
00115                   }
00116               }
00117           }
00118
00119           foreach (var wire in wiresManager.Wires)
00120           {
00121               CollectBranches(wire.Nodes);
00122           }
00123
00124           if (!allBranches.Any())
00125           {
00126               return (null, false);
00127           }
00128
00129           Branch? closestBranch = null;
00130           bool isUpperLine = false;
00131           double minDistance = double.MaxValue;
00132           int maxDepth = -1;
00133
00134           foreach (var (branch, depth) in allBranches)
00135           {
00136               // Skip if the branch is nested within the selected branch
00137               if (selectedBranch != null && selectedBranch.IsBranchNested(branch))
00138               {
00139                   continue;
00140               }
00141
00142               Line? upperLine = branch.UpperLine;
00143               Line? lowerLine = branch.LowerLine;
00144
00145               if (upperLine == null || lowerLine == null)
00146               {
00147                   continue;
00148               }
00149
00150               double upperX1 = upperLine.X1;
00151               double upperX2 = upperLine.X2;
00152               double upperY = upperLine.Y1;
00153
00154               double lowerX1 = lowerLine.X1;
00155               double lowerX2 = lowerLine.X2;
00156               double lowerY = lowerLine.Y1;
00157
00158               // Check proximity to upper line
00159               if (cursorPosition.X >= upperX1 && cursorPosition.X <= upperX2 &&
00160                   cursorPosition.Y >= upperY - 20 && cursorPosition.Y <= upperY + 20)
00161               {
00162                   double distanceToY1 = Math.Abs(upperY - cursorPosition.Y);
00163                   if (distanceToY1 < minDistance || (distanceToY1 == minDistance && depth > maxDepth))
00164                   {
00165                       minDistance = distanceToY1;
00166                       closestBranch = branch;
00167                       isUpperLine = true;
00168                       maxDepth = depth;
00169                   }
00170               }
00171
00172               // Check proximity to lower line
00173               if (cursorPosition.X >= lowerX1 && cursorPosition.X <= lowerX2 &&
00174                   cursorPosition.Y >= lowerY - 20 && cursorPosition.Y <= lowerY + 20)
```

```
00175                 {
00176                     double distanceToY2 = Math.Abs(lowerY - cursorPosition.Y);
00177                     if (distanceToY2 < minDistance || (distanceToY2 == minDistance && depth > maxDepth))
00178                     {
00179                         minDistance = distanceToY2;
00180                         closestBranch = branch;
00181                         isUpperLine = false;
00182                         maxDepth = depth;
00183                     }
00184                 }
00185             }
00186 
00187             if (minDistance <= 20 && closestBranch != null)
00188             {
00189                 return (closestBranch, isUpperLine);
00190             }
00191             return (null, false);
00192         }
00193     }
00194 }
```

7.47 ladder_diagram_app/Services/CanvasServices/CanvasInteraction↩ Manager.cs File Reference

Classes

- class ladder_diagram_app.Services.CanvasServices.CanvasInteractionManager

  Manages user interactions with the canvas in a ladder diagram application, including dragging, dropping, selecting, and deleting elements.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CanvasServices

7.48 CanvasInteractionManager.cs

Go to the documentation of this file.

```
00001 using System.Windows;
00002 using System.Windows.Controls;
00003 using System.Windows.Input;
00004 using System.Windows.Media;
00005 using System.Windows.Shapes;
00006 using ladder_diagram_app.Models.CanvasElements;
00007 using ladder_diagram_app.Models.CanvasElements.Instances;
00008 using ladder_diagram_app.Models.Variables;
00009 using ladder_diagram_app.Views;
00010 
00011 namespace ladder_diagram_app.Services.CanvasServices
00012 {
00016     public class CanvasInteractionManager
00017     {
00018         private readonly Canvas _canvas;
00019         private readonly WiresManager _wiresManager;
00020         private readonly CanvasElementFinder _elementFinder;
00021         private readonly CanvasManager _canvasManager;
00022         private readonly VariablesManager _variablesManager;
00023 
00024         // Fields for dragging elements
00025         private Point _dragStartPosition;
00026         private bool _isDragging;
00027         private bool _isDraggingWire;
00028 
00029         // Fields for selecting and deleting elements, wires, and branches
00030         private Node? _selectedNode;
00031         private Wire? _selectedWire;
```

```
00032       private Line? _wireLine;
00033
00034       // Tracks the currently highlighted object to manage highlight state
00035       private object? _highlightedObject;
00036
00045       public CanvasInteractionManager(
00046           Canvas canvas,
00047           WiresManager wiresManager,
00048           CanvasElementFinder elementFinder,
00049           CanvasManager canvasManager,
00050           VariablesManager variablesManager)
00051       {
00052           _canvas = canvas;
00053           _wiresManager = wiresManager;
00054           _elementFinder = elementFinder;
00055           _canvasManager = canvasManager;
00056           _variablesManager = variablesManager;
00057       }
00058
00063       public bool IsElementSelected()
00064       {
00065           return _selectedNode != null || _selectedWire != null;
00066       }
00067
00072       public void HighlightPosition(Point currentPosition)
00073       {
00074           Wire? closestWire = _elementFinder.FindClosestWire(currentPosition);
00075           var (closestBranch, isUpperLine) = _elementFinder.FindClosestBranch(currentPosition);
00076
00077           // Clear previous highlight
00078           if (_highlightedObject != null)
00079           {
00080               if (_highlightedObject is Wire wire) wire.UnhighlightWire();
00081               else if (_highlightedObject is Branch branch) branch.UnhighlightBranch();
00082           }
00083
00084           // Apply new highlight
00085           if (closestBranch != null)
00086           {
00087               closestBranch.HighlightBranch(isUpperLine);
00088               _highlightedObject = closestBranch;
00089           }
00090           else if (closestWire != null)
00091           {
00092               closestWire.HighlightWire();
00093               _highlightedObject = closestWire;
00094           }
00095       }
00096
00101       public void HandleDragOver(DragEventArgs e)
00102       {
00103           HighlightPosition(e.GetPosition(_canvas));
00104       }
00105
00111       public void HandleDrop(DragEventArgs e, Window owner)
00112       {
00113           Point dropPosition = e.GetPosition(_canvas);
00114           string? droppedElement = e.Data.GetData(typeof(string)) as string;
00115
00116           if (string.IsNullOrEmpty(droppedElement)) return;
00117
00118           Wire? closestWire = _elementFinder.FindClosestWire(dropPosition);
00119           var (closestBranch, isUpperLine) = _elementFinder.FindClosestBranch(dropPosition);
00120
00121           // Create new element or branch
00122           Node element = droppedElement != "Branch"
00123               ? new LadderElement(droppedElement,
00124                   _variablesManager.VariablesListContacts,
00125                   _variablesManager.VariablesListCoils,
00126                   _variablesManager.VariablesListMath,
00127                   _variablesManager.VariablesListCompare,
00128                   _variablesManager.VariablesListCounter,
00129                   _variablesManager.VariablesListTimer,
00130                   _variablesManager.VariablesListReset)
00131               : new Branch();
00132
00133           if (closestBranch != null)
00134           {
00135               if (droppedElement.Contains("Coil"))
00136               {
00137                   new NotificationWindow("A Coil-type element cannot be placed in a branch", owner).Show();
00138                   closestBranch.UnhighlightBranch();
00139                   UnselectEverything();
00140                   _highlightedObject = null;
00141                   return;
00142               }
00143
```

```
00144             List<Node> targetNodes = isUpperLine ? closestBranch.Nodes1 : closestBranch.Nodes2;
00145
00146             // Inserting elements between others
00147             bool inserted = false;
00148             for (int i = 0; i < targetNodes.Count; i++)
00149             {
00150                 // If the new element is before the first
00151                 if (i == 0 && dropPosition.X < targetNodes[i].X)
00152                 {
00153                     targetNodes.Insert(0, element);
00154                     inserted = true;
00155                     break;
00156                 }
00157                 // If the new element is between two existing ones
00158                 if (i < targetNodes.Count - 1 && dropPosition.X > targetNodes[i].X && dropPosition.X < targetNodes[i +
    1].X)
00159                 {
00160                     targetNodes.Insert(i + 1, element);
00161                     inserted = true;
00162                     break;
00163                 }
00164             }
00165             // If not inserted, add to the end
00166             if (!inserted)
00167             {
00168                 targetNodes.Add(element);
00169             }
00170
00171             element.Parent = closestBranch;
00172
00173             _canvasManager.UpdateCanvas();
00174
00175             closestBranch.UnhighlightBranch();
00176             UnselectEverything();
00177             _highlightedObject = null;
00178         }
00179         else if (closestWire != null)
00180         {
00181             // Checking for Coil types and existence on the wire
00182             if (droppedElement.Contains("Coil"))
00183             {
00184                 LadderElement? foundCoilElement = closestWire.Nodes.OfType<LadderElement>().FirstOrDefault(el =>
    el.Type.Contains("Coil"));
00185
00186                 if (foundCoilElement != null)
00187                 {
00188                     var dialog = new NotificationWindow("The wire already contains a Coil. Do you want to replace it?",
    owner, NotificationButtons.YesNo);
00189                     dialog.ShowDialog();
00190                     if (dialog.Result == true)
00191                     {
00192                         closestWire.Nodes.Remove(foundCoilElement);
00193                         closestWire.Nodes.Add(element);
00194                         element.Parent = closestWire;
00195                     }
00196                 }
00197                 else
00198                 {
00199                     closestWire.Nodes.Add(element);
00200                     element.Parent = closestWire;
00201                 }
00202             }
00203             else
00204             {
00205                 // Inserting elements between others
00206                 bool inserted = false;
00207                 for (int i = 0; i < closestWire.Nodes.Count; i++)
00208                 {
00209                     // If the new element is before the first
00210                     if (i == 0 && dropPosition.X < closestWire.Nodes[i].X)
00211                     {
00212                         closestWire.Nodes.Insert(0, element);
00213                         inserted = true;
00214                         break;
00215                     }
00216                     // If the new element is between two existing ones
00217                     if (i < closestWire.Nodes.Count - 1)
00218                     {
00219                         if (dropPosition.X > closestWire.Nodes[i].X && dropPosition.X < closestWire.Nodes[i + 1].X)
00220                         {
00221                             closestWire.Nodes.Insert(i + 1, element);
00222                             inserted = true;
00223                             break;
00224                         }
00225                     }
00226                 }
00227                 // If not inserted, add to the end
```

```
00228                 if (!inserted)
00229                 {
00230                     closestWire.Nodes.Add(element);
00231                 }
00232                 element.Parent = closestWire;
00233             }
00234 
00235             _canvasManager.UpdateCanvas();
00236 
00237             closestWire.UnhighlightWire();
00238             UnselectEverything();
00239             _highlightedObject = null;
00240         }
00241     }
00242 
00247     public void HandleMouseMove(MouseEventArgs e)
00248     {
00249         Point currentPosition = e.GetPosition(_canvas);
00250 
00251         if ((_selectedNode != null || _selectedWire != null) && e.LeftButton == MouseButtonState.Pressed &&
    !_isDragging && !_isDraggingWire)
00252         {
00253             Vector offset = currentPosition - _dragStartPosition;
00254 
00255             // Start dragging if moved beyond 5-pixel threshold
00256             if (Math.Abs(offset.X) > 5 || Math.Abs(offset.Y) > 5)   // 5 pixel threshold
00257             {
00258                 if (_selectedNode != null && _selectedNode.Image != null)
00259                 {
00260                     _isDragging = true;
00261                     _selectedNode.Image.CaptureMouse();
00262 
00263                     // Set the starting position of the branch image at the point where the branch drag started
00264                     if (!_canvas.Children.Contains(_selectedNode.Image))
00265                     {
00266                         _canvas.Children.Add(_selectedNode.Image);
00267                         Canvas.SetLeft(_selectedNode.Image, _dragStartPosition.X - _selectedNode.Image.Width / 2);
00268                         Canvas.SetTop(_selectedNode.Image, _dragStartPosition.Y - _selectedNode.Image.Height / 2);
00269                     }
00270                 }
00271                 else
00272                 {
00273                     _isDraggingWire = true;
00274                     _selectedWire?.SelectWire();
00275 
00276                     _wireLine = new Line
00277                     {
00278                         X1 = 0,
00279                         Y1 = currentPosition.Y,
00280                         X2 = _canvas.Width,
00281                         Y2 = currentPosition.Y,
00282                         Stroke = Brushes.Blue,
00283                         StrokeThickness = 4
00284                     };
00285                     _canvas.Children.Add(_wireLine);
00286                     _wireLine.CaptureMouse();
00287                 }
00288             }
00289         }
00290 
00291         if (_isDraggingWire && _selectedWire != null && _wireLine != null)
00292         {
00293             _wireLine.Y1 = currentPosition.Y;
00294             _wireLine.Y2 = currentPosition.Y;
00295             e.Handled = true;
00296         }
00297 
00298         if (_isDragging)
00299         {
00300             Vector offset = currentPosition - _dragStartPosition;
00301 
00302             double newX = Canvas.GetLeft(_selectedNode?.Image) + offset.X;
00303             double newY = Canvas.GetTop(_selectedNode?.Image) + offset.Y;
00304 
00305             Canvas.SetLeft(_selectedNode?.Image, newX);
00306             Canvas.SetTop(_selectedNode?.Image, newY);
00307 
00308             // Update ComboBox positions for LadderElement
00309             if (_selectedNode is LadderElement element && element.Image != null)
00310             {
00311                 element.Image.Opacity = 0.8;
00312                 element.VariableComboBoxes[0].Opacity = 0.8;
00313                 if (element.VariableComboBoxes.Count > 1)
00314                     element.VariableComboBoxes[1].Opacity = 0.8;
00315                 if (element.VariableComboBoxes.Count > 2)
00316                     element.VariableComboBoxes[2].Opacity = 0.8;
00317 
```

```
00318                   // Position ComboBoxes based on element type
00319                   if (element.Type.Contains("Contact") || element.Type.Contains("Coil"))
00320                   {
00321                       Canvas.SetLeft(element.VariableComboBoxes[0], newX - element.VariableComboBoxes[0].Width / 2 +
    element.Image.Width / 2);
00322                       Canvas.SetTop(element.VariableComboBoxes[0], newY - 32);
00323                   }
00324                   else if (element.Type == "AddMath" || element.Type == "SubtractMath" || element.Type ==
    "MultiplyMath" || element.Type == "DivideMath")
00325                   {
00326                       Canvas.SetLeft(element.VariableComboBoxes[0], newX - element.VariableComboBoxes[0].Width / 2 +
    element.Image.Width / 2);
00327                       Canvas.SetTop(element.VariableComboBoxes[0], newY + 7);
00328                       Canvas.SetLeft(element.VariableComboBoxes[1], newX - element.VariableComboBoxes[1].Width / 2 +
    element.Image.Width / 2);
00329                       Canvas.SetTop(element.VariableComboBoxes[1], newY + 47);
00330                       Canvas.SetLeft(element.VariableComboBoxes[2], newX - element.VariableComboBoxes[2].Width / 2 +
    element.Image.Width / 2);
00331                       Canvas.SetTop(element.VariableComboBoxes[2], newY + 92);
00332                   }
00333                   else if (element.Type == "MoveMath")
00334                   {
00335                       Canvas.SetLeft(element.VariableComboBoxes[0], newX - element.VariableComboBoxes[0].Width / 2 +
    element.Image.Width / 2);
00336                       Canvas.SetTop(element.VariableComboBoxes[0], newY + 27);
00337                       Canvas.SetLeft(element.VariableComboBoxes[1], newX - element.VariableComboBoxes[1].Width / 2 +
    element.Image.Width / 2);
00338                       Canvas.SetTop(element.VariableComboBoxes[1], newY + 72);
00339                   }
00340                   else if (element.Type.Contains("Compare"))
00341                   {
00342                       Canvas.SetLeft(element.VariableComboBoxes[0], newX - element.VariableComboBoxes[0].Width / 2 +
    element.Image.Width / 2);
00343                       Canvas.SetTop(element.VariableComboBoxes[0], newY + 7);
00344                       Canvas.SetLeft(element.VariableComboBoxes[1], newX - element.VariableComboBoxes[1].Width / 2 +
    element.Image.Width / 2);
00345                       Canvas.SetTop(element.VariableComboBoxes[1], newY + 52);
00346                   }
00347                   else if (element.Type.Contains("Timer") || element.Type.Contains("Count") || element.Type == "Reset")
00348                   {
00349                       Canvas.SetLeft(element.VariableComboBoxes[0], newX - element.VariableComboBoxes[0].Width / 2 +
    element.Image.Width / 2);
00350                       Canvas.SetTop(element.VariableComboBoxes[0], newY + 22);
00351                   }
00352               }
00353
00354               HighlightPosition(currentPosition);
00355
00356               _dragStartPosition = currentPosition;
00357               e.Handled = true;
00358           }
00359       }
00360
00366       public void HandleMouseLeftButtonUp(MouseButtonEventArgs e, Window owner)
00367       {
00368           if (!_isDragging && !_isDraggingWire) return;
00369
00370           Point dropPosition = e.GetPosition(_canvas);
00371           Wire? closestWire = _elementFinder.FindClosestWire(dropPosition);
00372           var (closestBranch, isUpperLine) = _elementFinder.FindClosestBranch(dropPosition, _selectedNode as Branch);
00373
00374           if (_selectedNode != null && _selectedNode.Image != null)
00375           {
00376               _selectedNode.Image.ReleaseMouseCapture();
00377
00378               object? originalParent = _selectedNode.Parent;
00379
00380               // Remove from original parent
00381               if (originalParent is Wire originalWire && originalWire.Nodes.Contains(_selectedNode))
00382               {
00383                   originalWire.Nodes.Remove(_selectedNode);
00384               }
00385               else if (originalParent is Branch originalBranch)
00386               {
00387                   if (!originalBranch.Nodes1.Remove(_selectedNode))
00388                       originalBranch.Nodes2.Remove(_selectedNode);
00389               }
00390
00391               // Restore opacity
00392               _selectedNode.Image.Opacity = 1;
00393               if (_selectedNode is LadderElement el)
00394               {
00395                   el.VariableComboBoxes[0].Opacity = 1;
00396                   if (el.VariableComboBoxes.Count > 1)
00397                       el.VariableComboBoxes[1].Opacity = 1;
00398                   if (el.VariableComboBoxes.Count > 2)
00399                       el.VariableComboBoxes[2].Opacity = 1;
```

```
00400             }
00401
00402             if (closestBranch != null)
00403             {
00404                 if (_selectedNode is LadderElement element && element.Type.Contains("Coil"))
00405                 {
00406                     new NotificationWindow("A Coil-type element cannot be placed in a branch", owner).Show();
00407                     if (originalParent is Wire wire)
00408                         wire.Nodes.Add(_selectedNode);
00409                 }
00410                 else
00411                 {
00412                     List<Node> targetNodes = isUpperLine ? closestBranch.Nodes1 : closestBranch.Nodes2;
00413
00414                     // Insert based on X position
00415                     bool inserted = false;
00416                     for (int i = 0; i < targetNodes.Count; i++)
00417                     {
00418                         if (i == 0 && dropPosition.X < targetNodes[i].X)
00419                         {
00420                             targetNodes.Insert(0, _selectedNode);
00421                             inserted = true;
00422                             break;
00423                         }
00424                         if (i < targetNodes.Count - 1 && dropPosition.X > targetNodes[i].X && dropPosition.X <
    targetNodes[i + 1].X)
00425                         {
00426                             targetNodes.Insert(i + 1, _selectedNode);
00427                             inserted = true;
00428                             break;
00429                         }
00430                     }
00431                     if (!inserted)
00432                     {
00433                         targetNodes.Add(_selectedNode);
00434                     }
00435                     _selectedNode.Parent = closestBranch;
00436                 }
00437             }
00438             else if (closestWire != null)
00439             {
00440                 // Checking for Coil types and existence on the wire
00441                 if (_selectedNode is LadderElement element && element.Type.Contains("Coil"))
00442                 {
00443                     LadderElement? foundElement = closestWire.Nodes.OfType<LadderElement>().FirstOrDefault(el =>
    el.Type.Contains("Coil"));
00444
00445                     if (foundElement != null)
00446                     {
00447                         _canvas.Children.Remove(_selectedNode.Image);
00448                         if (_selectedNode is LadderElement x)
00449                             _canvas.Children.Remove(x.VariableComboBoxes[0]);
00450                         var dialog = new NotificationWindow("The wire already contains a Coil. Do you want to replace it?",
    owner, NotificationButtons.YesNo);
00451                         dialog.ShowDialog();
00452                         if (dialog.Result == true)
00453                         {
00454                             closestWire.Nodes.Remove(foundElement);
00455                             closestWire.Nodes.Add(_selectedNode);
00456                             _selectedNode.Parent = closestWire;
00457                         }
00458                         else
00459                         {
00460                             // Revert to original parent if user rejects replacement
00461                             if (originalParent is Wire wire)
00462                                 wire.Nodes.Add(_selectedNode);
00463                         }
00464                     }
00465                     else
00466                     {
00467                         closestWire.Nodes.Add(_selectedNode);
00468                         _selectedNode.Parent = closestWire;
00469                     }
00470                 }
00471                 else
00472                 {
00473                     // Inserting elements between others
00474                     bool inserted = false;
00475                     for (int i = 0; i < closestWire.Nodes.Count; i++)
00476                     {
00477                         // If the new element is before the first
00478                         if (i == 0 && dropPosition.X < closestWire.Nodes[i].X)
00479                         {
00480                             closestWire.Nodes.Insert(0, _selectedNode);
00481                             inserted = true;
00482                             break;
00483                         }
```

```
00484                       // If the new element is between two existing ones
00485                       if (i < closestWire.Nodes.Count - 1 && dropPosition.X > closestWire.Nodes[i].X && dropPosition.X < closestWire.Nodes[i + 1].X)
00486                       {
00487                           closestWire.Nodes.Insert(i + 1, _selectedNode);
00488                           inserted = true;
00489                           break;
00490                       }
00491                   }
00492                   if (!inserted)
00493                   {
00494                       closestWire.Nodes.Add(_selectedNode);
00495                   }
00496                   _selectedNode.Parent = closestWire;
00497               }
00498           }
00499           else
00500           {
00501               // Revert to original parent if no valid drop location
00502               if (originalParent is Wire wire)
00503                   wire.Nodes.Add(_selectedNode);
00504               else if (originalParent is Branch branch)
00505                   (isUpperLine ? branch.Nodes1 : branch.Nodes2).Add(_selectedNode);
00506           }
00507
00508           if (_highlightedObject is Wire hwire) hwire.UnhighlightWire();
00509           else if (_highlightedObject is Branch hbranch) hbranch.UnhighlightBranch();
00510           _highlightedObject = null;
00511           UnselectEverything();
00512           _canvasManager.UpdateCanvas();
00513           _isDragging = false;
00514           _selectedNode = null;
00515           e.Handled = true;
00516       }
00517
00518       if (_selectedWire != null && _wireLine != null)
00519       {
00520           _wireLine.ReleaseMouseCapture();
00521
00522           double dropY = e.GetPosition(_canvas).Y;
00523
00524           _wiresManager.RemoveWire(_selectedWire);
00525
00526           // Insert wire based on Y position
00527           bool inserted = false;
00528           for (int i = 0; i < _wiresManager.Wires.Count; i++)
00529           {
00530               if (i == 0 && dropPosition.Y < _wiresManager.Wires[i].Y)
00531               {
00532                   _wiresManager.InsertWire(_selectedWire, 0);
00533                   inserted = true;
00534                   break;
00535               }
00536               // If the new element is between two existing ones
00537               if (i < _wiresManager.Wires.Count - 1 && dropPosition.Y > _wiresManager.Wires[i].Y && dropPosition.Y < _wiresManager.Wires[i + 1].Y)
00538               {
00539                   _wiresManager.InsertWire(_selectedWire, i + 1);
00540                   inserted = true;
00541                   break;
00542               }
00543           }
00544           if (!inserted)
00545           {
00546               _wiresManager.AddWire(_selectedWire);
00547           }
00548
00549           _selectedWire.UnselectWire();
00550           _isDraggingWire = false;
00551           _selectedWire = null;
00552
00553           _canvasManager.UpdateCanvas();
00554       }
00555   }
00556
00562   public void HandleMouseLeftButtonDown(MouseButtonEventArgs e, ListView elementListView)
00563   {
00564       _canvas.Focus(); // Ensure canvas gets focus on click
00565       elementListView.SelectedItem = null;    // Clear ListView selection
00566
00567       Point clickPosition = e.GetPosition(_canvas);
00568       LadderElement? closestElement = _elementFinder.FindClosestElement(clickPosition);
00569       Wire? closestWire = _elementFinder.FindClosestWire(clickPosition);
00570
00571       UnselectEverything();
00572
00573       if (closestElement != null)
```

```
00574         {
00575             _selectedNode = closestElement;
00576             _selectedNode.HighlightNode();
00577             _dragStartPosition = e.GetPosition(_canvas);
00578             e.Handled = true;
00579             return;
00580         }
00581 
00582         // Perform hit test for lines
00583         double hitTestOffset = 5.0;
00584         var clickPoint = clickPosition;
00585         var hitArea = new EllipseGeometry(clickPoint, hitTestOffset, hitTestOffset);
00586         var hitTestParams = new GeometryHitTestParameters(hitArea);
00587         List<DependencyObject> hitResults = new List<DependencyObject>();
00588 
00589         VisualTreeHelper.HitTest(
00590             _canvas,
00591             null,
00592             result =>
00593             {
00594                 if (result.VisualHit != null)
00595                 {
00596                     hitResults.Add(result.VisualHit);
00597                 }
00598                 return HitTestResultBehavior.Continue;
00599             },
00600             hitTestParams);
00601 
00602         // Check for matches
00603         foreach (var hit in hitResults)
00604         {
00605             if (hit is Line clickedLine)
00606             {
00607                 // If the line has a Tag that is Branch, select that branch
00608                 if (clickedLine.Tag is Branch branch)
00609                 {
00610                     _selectedNode = branch;
00611                     branch.HighlightBranchRecursive();
00612                     _dragStartPosition = clickPosition;
00613                     return;
00614                 }
00615                 // If the line has a Tag that is Wire, select the wire
00616                 else if (clickedLine.Tag is Wire wire)
00617                 {
00618                     _selectedWire = wire;
00619                     _selectedWire.SelectWire();
00620                     _dragStartPosition = clickPosition;
00621                     return;
00622                 }
00623             }
00624         }
00625     }
00626 
00630     public void UnselectEverything()
00631     {
00632         if (_selectedNode != null)
00633         {
00634             if (_selectedNode is LadderElement && _selectedNode.Image != null)
00635             {
00636                 _selectedNode.UnhighlightNode();
00637             }
00638             else if (_selectedNode is Branch branch)
00639             {
00640                 branch.UnhighlightBranchRecursive();
00641             }
00642             _selectedNode = null;
00643         }
00644 
00645         if (_selectedWire != null)
00646         {
00647             var previousLine = _selectedWire.WireLine;
00648             if (previousLine != null)
00649             {
00650                 previousLine.Stroke = Brushes.Black;
00651                 previousLine.StrokeThickness = 2;
00652             }
00653             _selectedWire = null;
00654         }
00655     }
00656 
00661     public void DeleteSelected(Window owner)
00662     {
00663         if (_selectedNode != null)
00664         {
00665             if (_selectedNode.Parent is Wire wire)
00666             {
00667                 wire.Nodes.Remove(_selectedNode);
```

```
00668                 }
00669                 else if (_selectedNode.Parent is Branch branch)
00670                 {
00671                     if (branch.Nodes1.Contains(_selectedNode))
00672                         branch.Nodes1.Remove(_selectedNode);
00673                     else if (branch.Nodes2.Contains(_selectedNode))
00674                         branch.Nodes2.Remove(_selectedNode);
00675                 }
00676 
00677                 _selectedNode = null;
00678                 _canvasManager.UpdateCanvas();
00679             }
00680             else if (_selectedWire != null)
00681             {
00682                 _wiresManager.RemoveWire(_selectedWire);
00683                 _selectedWire = null;
00684                 _canvasManager.UpdateCanvas();
00685             }
00686             else
00687             {
00688                 new NotificationWindow("Select an element to delete", owner).Show();
00689             }
00690         }
00691     }
00692 }
```

7.49 ladder_diagram_app/Services/CanvasServices/CanvasManager.cs File Reference

Classes

- class ladder_diagram_app.Services.CanvasServices.CanvasManager

  Manages the rendering and layout of canvas elements in a ladder diagram application.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CanvasServices

7.50 CanvasManager.cs

Go to the documentation of this file.

```
00001 using ladder_diagram_app.Models.CanvasElements;
00002 using System.Windows.Controls;
00003 using System.Windows;
00004 using ladder_diagram_app.Models.CanvasElements.Instances;
00005 
00006 namespace ladder_diagram_app.Services.CanvasServices
00007 {
00011     public class CanvasManager
00012     {
00013         private readonly Canvas _canvas;
00014         private readonly Grid _gridCanvas;
00015         private readonly WiresManager _wiresManager;
00016 
00023         public CanvasManager(Canvas canvas, Grid gridCanvas, WiresManager wiresManager)
00024         {
00025             _canvas = canvas;
00026             _gridCanvas = gridCanvas;
00027             _wiresManager = wiresManager;
00028         }
00029 
00033         public void UpdateCanvas()
00034         {
00035             // Calculate base canvas dimensions, accounting for scrollbars
00036             double baseWidth = _gridCanvas.ActualWidth - SystemParameters.VerticalScrollBarWidth;
```

```
00037           double baseHeight = _gridCanvas.ActualHeight - SystemParameters.HorizontalScrollBarHeight;
00038
00039           // Determine required canvas size based on wire and node dimensions
00040           double requiredWidth = _wiresManager.Wires.Any() ? _wiresManager.Wires.Max(w => w.Nodes.Sum(e =>
    e.Width)) : 0;
00041           double requiredHeight = _wiresManager.Wires.Sum(w => w.Height) + 80;
00042
00043           _canvas.Width = Math.Max(baseWidth, requiredWidth);
00044           _canvas.Height = Math.Max(baseHeight, requiredHeight);
00045
00046           // Position wires and their nodes
00047           double startY = 60;
00048           foreach (var wire in _wiresManager.Wires)
00049           {
00050               wire.Width = _canvas.Width;
00051               wire.Y = startY;
00052               UpdateElementsParameters(wire.Nodes, wire.Y, 0, _canvas);
00053
00054               startY += wire.Height;
00055
00056               wire.Nodes = wire.Nodes.OrderBy(n => n.X).ToList();
00057           }
00058
00059           // Re-adjust canvas size after positioning
00060           requiredWidth = _wiresManager.Wires.Any() ? _wiresManager.Wires.Max(w => w.Nodes.Sum(e => e.Width)) :
    0;
00061           requiredHeight = _wiresManager.Wires.Sum(w => w.Height) + 80;
00062
00063           _canvas.Width = Math.Max(baseWidth, requiredWidth);
00064           _canvas.Height = Math.Max(baseHeight, requiredHeight);
00065
00066           // Render canvas elements
00067           _canvas.Children.Clear();
00068           foreach (var wire in _wiresManager.Wires)
00069           {
00070               _canvas.Children.Add(wire.WireLine);
00071               DrawNodes(wire.Nodes, wire.Y, 0, _canvas);
00072           }
00073       }
00074
00082       private void UpdateElementsParameters(List<Node> nodes, double y1, double startX, Canvas canvas)
00083       {
00084           double currentX = startX;
00085
00086           foreach (var node in nodes)
00087           {
00088               if (node is LadderElement element)
00089               {
00090                   // Position coil elements at the right edge of the canvas
00091                   if (element.Type == "Coil" || element.Type == "OSPCoil" || element.Type == "SetCoil" || element.Type
    == "ResetCoil")
00092                   {
00093                       element.X = canvas.Width - element.Width / 2;
00094                   }
00095                   else
00096                   {
00097                       element.X = currentX + element.Width / 2;
00098                       currentX += element.Width;
00099                   }
00100                   element.Y = y1;
00101               }
00102               else if (node is Branch branch)
00103               {
00104                   branch.X = currentX + branch.Width / 2;
00105                   currentX += branch.Width;
00106                   branch.Y = y1;
00107
00108                   double branchStartX = branch.X - branch.Width / 2 + 10;
00109
00110                   // Recursively update nodes in both branch lines
00111                   UpdateElementsParameters(branch.Nodes1, branch.Y, branchStartX, canvas);
00112                   UpdateElementsParameters(branch.Nodes2, branch.Y2, branchStartX, canvas);
00113
00114                   branch.Nodes1 = branch.Nodes1.OrderBy(n => n.X).ToList();
00115                   branch.Nodes2 = branch.Nodes2.OrderBy(n => n.X).ToList();
00116               }
00117           }
00118       }
00119
00127       private void DrawNodes(List<Node> nodes, double y1, double startX, Canvas canvas)
00128       {
00129           foreach (var node in nodes)
00130           {
00131               if (node is LadderElement element && element.Image != null)
00132               {
00133                   // Position the element image
00134                   Canvas.SetLeft(element.Image, element.X - element.Image.Width / 2);
```

```
00135                 Canvas.SetTop(element.Image, element.Y - element.Image.Height / 2);
00136                 canvas.Children.Add(element.Image);
00137
00138                 // Position ComboBoxes based on element type
00139                 if (element.Type.Contains("Contact") || element.Type.Contains("Coil"))
00140                 {
00141                     Canvas.SetLeft(element.VariableComboBoxes[0], element.X - element.VariableComboBoxes[0].Width / 2);
00142                     Canvas.SetTop(element.VariableComboBoxes[0], element.Y - 45);
00143                     canvas.Children.Add(element.VariableComboBoxes[0]);
00144                 }
00145                 else if (element.Type == "AddMath" || element.Type == "SubtractMath" || element.Type ==
    "MultiplyMath" || element.Type == "DivideMath")
00146                 {
00147                     Canvas.SetLeft(element.VariableComboBoxes[0], element.X - element.VariableComboBoxes[0].Width / 2);
00148                     Canvas.SetTop(element.VariableComboBoxes[0], element.Y - 52.5);
00149                     Canvas.SetLeft(element.VariableComboBoxes[1], element.X - element.VariableComboBoxes[1].Width / 2);
00150                     Canvas.SetTop(element.VariableComboBoxes[1], element.Y - 12.5);
00151                     Canvas.SetLeft(element.VariableComboBoxes[2], element.X - element.VariableComboBoxes[2].Width / 2);
00152                     Canvas.SetTop(element.VariableComboBoxes[2], element.Y + 27.5);
00153                     canvas.Children.Add(element.VariableComboBoxes[0]);
00154                     canvas.Children.Add(element.VariableComboBoxes[1]);
00155                     canvas.Children.Add(element.VariableComboBoxes[2]);
00156                 }
00157                 else if (element.Type == "MoveMath")
00158                 {
00159                     Canvas.SetLeft(element.VariableComboBoxes[0], element.X - element.VariableComboBoxes[0].Width / 2);
00160                     Canvas.SetTop(element.VariableComboBoxes[0], element.Y - 25);
00161                     Canvas.SetLeft(element.VariableComboBoxes[1], element.X - element.VariableComboBoxes[1].Width / 2);
00162                     Canvas.SetTop(element.VariableComboBoxes[1], element.Y + 20);
00163                     canvas.Children.Add(element.VariableComboBoxes[0]);
00164                     canvas.Children.Add(element.VariableComboBoxes[1]);
00165                 }
00166                 else if (element.Type.Contains("Compare"))
00167                 {
00168                     Canvas.SetLeft(element.VariableComboBoxes[0], element.X - element.VariableComboBoxes[0].Width / 2);
00169                     Canvas.SetTop(element.VariableComboBoxes[0], element.Y - 35);
00170                     Canvas.SetLeft(element.VariableComboBoxes[1], element.X - element.VariableComboBoxes[1].Width / 2);
00171                     Canvas.SetTop(element.VariableComboBoxes[1], element.Y + 10);
00172                     canvas.Children.Add(element.VariableComboBoxes[0]);
00173                     canvas.Children.Add(element.VariableComboBoxes[1]);
00174                 }
00175                 else if (element.Type.Contains("Timer") || element.Type.Contains("Count") || element.Type == "Reset")
00176                 {
00177                     Canvas.SetLeft(element.VariableComboBoxes[0], element.X - element.VariableComboBoxes[0].Width / 2);
00178                     Canvas.SetTop(element.VariableComboBoxes[0], element.Y - 5);
00179                     canvas.Children.Add(element.VariableComboBoxes[0]);
00180                 }
00181             }
00182             else if (node is Branch branch)
00183             {
00184                 // Draw branch lines
00185                 canvas.Children.Add(branch.UpperLine);
00186                 canvas.Children.Add(branch.LowerLine);
00187                 canvas.Children.Add(branch.LeftLine);
00188                 canvas.Children.Add(branch.RightLine);
00189
00190                 double branchStartX = branch.X - branch.Width / 2 + 10;
00191                 // Recursively draw nodes in both branch lines
00192                 DrawNodes(branch.Nodes1, branch.Y, branchStartX, canvas);
00193                 DrawNodes(branch.Nodes2, branch.Y2, branchStartX, canvas);
00194             }
00195         }
00196     }
00197 }
00198 }
```

7.51 ladder_diagram_app/Services/CommunicationServices/BLE/BleCommunicationService.cs File Reference

Classes

- class ladder_diagram_app.Services.CommunicationServices.BLE.BleCommunicationService

  Provides Bluetooth Low Energy (BLE) communication services for connecting to and interacting with a BLE device.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CommunicationServices
- namespace ladder_diagram_app.Services.CommunicationServices.BLE

7.52 BleCommunicationService.cs

Go to the documentation of this file.

```
00001 using System.Diagnostics;
00002 using System.Text;
00003 using System.Text.Json;
00004 using ladder_diagram_app.Services.CommunicationServices;
00005 using Windows.Devices.Bluetooth;
00006 using Windows.Devices.Bluetooth.GenericAttributeProfile;
00007 using Windows.Storage.Streams;
00008
00009 namespace ladder_diagram_app.Services.CommunicationServices.BLE
00010 {
00014     public class BleCommunicationService : IDeviceCommunicationService, IDisposable
00015     {
00016         private BluetoothLEDevice? _bleDevice;
00017         private GattCharacteristic? _readConfigurationCharacteristic;
00018         private GattCharacteristic? _writeConfigurationCharacteristic;
00019         private GattCharacteristic? _readMonitorCharacteristic;
00020         private GattCharacteristic? _readOneWireCharacteristic;
00021
00022         private readonly Guid _serviceUuid = Guid.Parse("00001234-0000-1000-8000-00805f9b34fb");
00023         private readonly Guid _readConfigurationCharUuid = Guid.Parse("0000FFF1-0000-1000-8000-00805f9b34fb");
00024         private readonly Guid _writeConfigurationCharUuid = Guid.Parse("0000FFF2-0000-1000-8000-00805f9b34fb");
00025         private readonly Guid _readMonitorCharUuid = Guid.Parse("0000FFF3-0000-1000-8000-00805f9b34fb");
00026         private readonly Guid _readOneWireCharUuid = Guid.Parse("0000FFF4-0000-1000-8000-00805f9b34fb");
00027
00028         public event EventHandler<string>? ConfigurationReceived;
00029         public event EventHandler<string>? MonitorDataReceived;
00030         public event EventHandler<string>? OneWireDataReceived;
00031         public event EventHandler<bool>? ConnectionStatusChanged;
00032
00036         public bool IsConnected { get; private set; }
00037
00041         public string ConnectionType => "BLE";
00042
00043         private StringBuilder _jsonConfigurationBuffer = new StringBuilder();
00044         private StringBuilder _jsonMonitorBuffer = new StringBuilder();
00045         private StringBuilder _jsonOneWireBuffer = new StringBuilder();
00046
00047         private bool _monitorTaskRunning = false;
00048         private bool _oneWireTaskRunning = false;
00049
00050         private readonly int ChunkSize = 250; // Maximum chunk size for data transfer, accounting for ATT overhead
00051
00058         private async Task<bool> ConnectToDeviceWithRetry(string deviceId, int maxRetries = 5)
00059         {
00060             int retryCount = 0;
00061             while (retryCount < maxRetries)
00062             {
00063                 try
00064                 {
00065                     _bleDevice = await BluetoothLEDevice.FromIdAsync(deviceId);
00066                     if (_bleDevice != null) return true;
00067                     retryCount++;
00068                     await Task.Delay(1000 * retryCount);
00069                 }
00070                 catch (Exception ex)
00071                 {
00072                     Debug.WriteLine($"BLE Connection attempt {retryCount + 1} failed: {ex.Message}");
00073                     retryCount++;
00074                     await Task.Delay(1000 * retryCount);
00075                 }
00076             }
00077             return false;
00078         }
00079
00086         private async Task<GattDeviceServicesResult?> GetServicesWithRetry(BluetoothLEDevice device, int maxRetries =
      5)
00087         {
00088             int retryCount = 0;
```

```
00089           while (retryCount < maxRetries)
00090           {
00091               var result = await device.GetGattServicesAsync(BluetoothCacheMode.Uncached);
00092               if (result != null && result.Status == GattCommunicationStatus.Success)
00093                   return result;
00094               retryCount++;
00095               await Task.Delay(1000 * retryCount);
00096           }
00097           return null;
00098       }
00099
00105       public async Task<bool> ConnectAsync(string deviceId)
00106       {
00107           try
00108           {
00109               if (IsConnected) return true;
00110
00111               if (string.IsNullOrWhiteSpace(deviceId))
00112               {
00113                   Debug.WriteLine("BLE Connection failed: Device ID is null or empty");
00114                   return false;
00115               }
00116
00117               bool connected = await ConnectToDeviceWithRetry(deviceId);
00118               if (!connected || _bleDevice == null)
00119               {
00120                   await DisconnectAsync();
00121                   Debug.WriteLine("BLE Failed to connect to device after multiple attempts");
00122                   return false;
00123               }
00124
00125               var servicesResult = await GetServicesWithRetry(_bleDevice);
00126               if (servicesResult == null || servicesResult.Status != GattCommunicationStatus.Success)
00127               {
00128                   await DisconnectAsync();
00129                   Debug.WriteLine("BLE Failed to get services after multiple attempts");
00130                   return false;
00131               }
00132
00133               bool setupCharacteristicsSuccess = await SetupCharacteristics(servicesResult);
00134               if (!setupCharacteristicsSuccess) return false;
00135
00136               await RequestConfigurationAsync();
00137
00138               IsConnected = true;
00139               ConnectionStatusChanged?.Invoke(this, true);
00140
00141               _jsonMonitorBuffer.Clear();
00142               _jsonOneWireBuffer.Clear();
00143
00144               if (!_monitorTaskRunning)
00145               {
00146                   _monitorTaskRunning = true;
00147                   _ = Task.Run(() => ReadMonitorBle());
00148               }
00149               if (!_oneWireTaskRunning)
00150               {
00151                   _oneWireTaskRunning = true;
00152                   _ = Task.Run(() => ReadOneWireBle());
00153               }
00154
00155               return true;
00156           }
00157           catch (Exception ex)
00158           {
00159               Debug.WriteLine($"BLE Connection failed: {ex.Message}");
00160               return false;
00161           }
00162       }
00163
00167       public async Task DisconnectAsync()
00168       {
00169           try
00170           {
00171               if (!IsConnected || _bleDevice == null) return;
00172
00173               IsConnected = false;
00174               _monitorTaskRunning = false;
00175               _oneWireTaskRunning = false;
00176
00177               var servicesResult = await _bleDevice.GetGattServicesAsync(BluetoothCacheMode.Uncached);
00178               if (servicesResult.Status == GattCommunicationStatus.Success)
00179               {
00180                   foreach (var service in servicesResult.Services)
00181                   {
00182                       service.Dispose();
00183                   }
```

```
00184             }
00185
00186             _readConfigurationCharacteristic = null;
00187             _writeConfigurationCharacteristic = null;
00188             _readMonitorCharacteristic = null;
00189             _readOneWireCharacteristic = null;
00190         }
00191         catch (Exception ex)
00192         {
00193             Debug.WriteLine($"BLE Disconnection failed: {ex.Message}");
00194         }
00195         finally
00196         {
00197             _bleDevice?.Dispose();
00198             _bleDevice = null;
00199             ConnectionStatusChanged?.Invoke(this, false);
00200         }
00201     }
00202
00208     private async Task<bool> SetupCharacteristics(GattDeviceServicesResult servicesResult)
00209     {
00210         try
00211         {
00212             foreach (var service in servicesResult.Services)
00213             {
00214                 if (service.Uuid == _serviceUuid)
00215                 {
00216                     var characteristicsResult = await service.GetCharacteristicsAsync();
00217                     if (characteristicsResult.Status != GattCommunicationStatus.Success)
00218                     {
00219                         continue;
00220                     }
00221
00222                     foreach (var characteristic in characteristicsResult.Characteristics)
00223                     {
00224                         if (characteristic.Uuid == _readConfigurationCharUuid)
00225                             _readConfigurationCharacteristic = characteristic;
00226                         else if (characteristic.Uuid == _writeConfigurationCharUuid)
00227                             _writeConfigurationCharacteristic = characteristic;
00228                         else if (characteristic.Uuid == _readMonitorCharUuid)
00229                             _readMonitorCharacteristic = characteristic;
00230                         else if (characteristic.Uuid == _readOneWireCharUuid)
00231                             _readOneWireCharacteristic = characteristic;
00232                     }
00233                 }
00234             }
00235
00236             if (_readConfigurationCharacteristic == null || _writeConfigurationCharacteristic == null ||
00237                 _readMonitorCharacteristic == null || _readOneWireCharacteristic == null)
00238             {
00239                 await DisconnectAsync();
00240                 Debug.WriteLine("BLE Required characteristics not found.");
00241                 return false;
00242             }
00243
00244             return true;
00245         }
00246         catch (Exception ex)
00247         {
00248             Debug.WriteLine($"BLE Characteristics Setup failed: {ex.Message}");
00249             return false;
00250         }
00251     }
00252
00256     public async Task RequestConfigurationAsync()
00257     {
00258         try
00259         {
00260             _jsonConfigurationBuffer.Clear();
00261             while (true)
00262             {
00263                 var readTask =
    _readConfigurationCharacteristic?.ReadValueAsync(BluetoothCacheMode.Uncached).AsTask();
00264                 if (readTask == null)
00265                 {
00266                     Debug.WriteLine("BLE Read configuration failed: Characteristic is null");
00267                     break;
00268                 }
00269                 if (await Task.WhenAny(readTask, Task.Delay(5000)) != readTask)
00270                 {
00271                     Debug.WriteLine("BLE Read configuration timeout");
00272                     break;
00273                 }
00274                 var readResult = await readTask;
00275
00276                 if (readResult.Status != GattCommunicationStatus.Success)
00277                 {
```

```
00278                     Debug.WriteLine("BLE Failed to read configuration");
00279                     break;
00280                 }
00281
00282                 var reader = DataReader.FromBuffer(readResult.Value);
00283                 byte[] data = new byte[reader.UnconsumedBufferLength];
00284                 reader.ReadBytes(data);
00285
00286                 if (data.Length == 0)
00287                 {
00288                     string completeJson = _jsonConfigurationBuffer.ToString();
00289                     _jsonConfigurationBuffer.Clear();
00290                     ConfigurationReceived?.Invoke(this, completeJson);
00291                     break;
00292                 }
00293
00294                 _jsonConfigurationBuffer.Append(Encoding.UTF8.GetString(data));
00295             }
00296         }
00297         catch (Exception ex)
00298         {
00299             Debug.WriteLine($"BLE Reading Configuration failed: {ex.Message}");
00300             await DisconnectAsync();
00301         }
00302     }
00303
00309     public async Task<bool> SendConfigurationAsync(string configJson)
00310     {
00311         try
00312         {
00313             if (!IsConnected) return false;
00314
00315             byte[] jsonBytes = Encoding.UTF8.GetBytes(configJson);
00316
00317             for (int i = 0; i < jsonBytes.Length; i += ChunkSize)
00318             {
00319                 int chunkLength = Math.Min(ChunkSize, jsonBytes.Length - i);
00320                 byte[] chunk = new byte[chunkLength];
00321                 Array.Copy(jsonBytes, i, chunk, 0, chunkLength);
00322
00323                 using var writer = new DataWriter();
00324                 writer.WriteBytes(chunk);
00325                 var writeResult = await _writeConfigurationCharacteristic?.WriteValueAsync(writer.DetachBuffer());
00326                 if (writeResult != GattCommunicationStatus.Success)
00327                 {
00328                     Debug.WriteLine("BLE Failed to write configuration");
00329                     return false;
00330                 }
00331             }
00332             return true;
00333         }
00334         catch (Exception ex)
00335         {
00336             Debug.WriteLine($"BLE Error writing configuration: {ex.Message}");
00337             return false;
00338         }
00339     }
00340
00344     private async Task ReadMonitorBle()
00345     {
00346         try
00347         {
00348             while (IsConnected && _bleDevice != null)
00349             {
00350                 var readTask = _readMonitorCharacteristic?.ReadValueAsync(BluetoothCacheMode.Uncached).AsTask();
00351                 if (readTask == null)
00352                 {
00353                     Debug.WriteLine("BLE Read Monitor Data failed: Characteristic is null");
00354                     break;
00355                 }
00356                 if (await Task.WhenAny(readTask, Task.Delay(5000)) != readTask)
00357                 {
00358                     Debug.WriteLine("BLE Read Monitor Data timeout");
00359                     break;
00360                 }
00361                 var readResult = await readTask;
00362
00363                 if (readResult.Status != GattCommunicationStatus.Success)
00364                 {
00365                     Debug.WriteLine("BLE Failed to read monitor data");
00366                     break;
00367                 }
00368
00369                 var reader = DataReader.FromBuffer(readResult.Value);
00370                 byte[] data = new byte[reader.UnconsumedBufferLength];
00371                 reader.ReadBytes(data);
00372
```

```
00373                 if (data.Length == 0)
00374                 {
00375                     string completeJson = _jsonMonitorBuffer.ToString();
00376                     if (!string.IsNullOrEmpty(completeJson))
00377                     {
00378                         MonitorDataReceived?.Invoke(this, completeJson);
00379                         _jsonMonitorBuffer.Clear();
00380                     }
00381                 }
00382                 else
00383                 {
00384                     _jsonMonitorBuffer.Append(Encoding.UTF8.GetString(data));
00385                 }
00386             }
00387         }
00388         catch (JsonException ex)
00389         {
00390             Debug.WriteLine($"BLE: Invalid JSON in monitor data: {ex.Message}");
00391         }
00392         catch (Exception ex)
00393         {
00394             Debug.WriteLine($"BLE Error reading monitor data: {ex.Message}");
00395         }
00396     }
00397
00401     private async Task ReadOneWireBle()
00402     {
00403         try
00404         {
00405             while (IsConnected && _bleDevice != null)
00406             {
00407                 var readTask = _readOneWireCharacteristic?.ReadValueAsync(BluetoothCacheMode.Uncached).AsTask();
00408                 if (readTask == null)
00409                 {
00410                     Debug.WriteLine("BLE Read One Wire Data failed: Characteristic is null");
00411                     break;
00412                 }
00413                 if (await Task.WhenAny(readTask, Task.Delay(5000)) != readTask)
00414                 {
00415                     Debug.WriteLine("BLE Read One Wire Data timeout");
00416                     break;
00417                 }
00418                 var readResult = await readTask;
00419
00420                 if (readResult.Status != GattCommunicationStatus.Success)
00421                 {
00422                     Debug.WriteLine("BLE Failed to read one wire data");
00423                     break;
00424                 }
00425
00426                 var reader = DataReader.FromBuffer(readResult.Value);
00427                 byte[] data = new byte[reader.UnconsumedBufferLength];
00428                 reader.ReadBytes(data);
00429
00430                 if (data.Length == 0)
00431                 {
00432                     string completeJson = _jsonOneWireBuffer.ToString();
00433                     if (!string.IsNullOrEmpty(completeJson))
00434                     {
00435                         OneWireDataReceived?.Invoke(this, completeJson);
00436                         _jsonOneWireBuffer.Clear();
00437                     }
00438                 }
00439                 else
00440                 {
00441                     _jsonOneWireBuffer.Append(Encoding.UTF8.GetString(data));
00442                 }
00443             }
00444         }
00445         catch (JsonException ex)
00446         {
00447             Debug.WriteLine($"BLE: Invalid JSON in one wire data: {ex.Message}");
00448         }
00449         catch (Exception ex)
00450         {
00451             Debug.WriteLine($"BLE Error reading one wire: {ex.Message}");
00452         }
00453     }
00454
00458     public void Dispose()
00459     {
00460         try
00461         {
00462             if (_bleDevice == null) return;
00463
00464             _monitorTaskRunning = false;
00465             _oneWireTaskRunning = false;
```

```
00466
00467                 var servicesTask = _bleDevice.GetGattServicesAsync(BluetoothCacheMode.Uncached);
00468                 var servicesResult = servicesTask.AsTask().GetAwaiter().GetResult();
00469                 if (servicesResult.Status == GattCommunicationStatus.Success)
00470                 {
00471                     foreach (var service in servicesResult.Services)
00472                     {
00473                         service.Dispose();
00474                     }
00475                 }
00476
00477                 _readConfigurationCharacteristic = null;
00478                 _writeConfigurationCharacteristic = null;
00479                 _readMonitorCharacteristic = null;
00480                 _readOneWireCharacteristic = null;
00481
00482                 _bleDevice.Dispose();
00483                 _bleDevice = null;
00484
00485                 GC.Collect();
00486                 GC.SuppressFinalize(this);
00487             }
00488             catch (Exception ex)
00489             {
00490                 Debug.WriteLine($"BLE Dispose failed: {ex.Message}");
00491             }
00492         }
00493     }
00494 }
```

7.53 ladder_diagram_app/Services/CommunicationServices/BLE/BluetoothSelection/BleDeviceWatcher.cs File Reference

Classes

- class ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection.BleDeviceWatcher
  Monitors and manages Bluetooth Low Energy (BLE) devices using a device watcher.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CommunicationServices
- namespace ladder_diagram_app.Services.CommunicationServices.BLE
- namespace ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection

7.54 BleDeviceWatcher.cs

Go to the documentation of this file.

```
00001 using System.Collections.ObjectModel;
00002 using System.Diagnostics;
00003 using Windows.Devices.Enumeration;
00004
00005 namespace ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection
00006 {
00010     public class BleDeviceWatcher : IDisposable
00011     {
00012         private readonly ObservableCollection<DeviceInformation> _devices = new
      ObservableCollection<DeviceInformation>();
00013         private DeviceWatcher? _deviceWatcher;
00014
00018         public ObservableCollection<DeviceInformation> Devices => _devices;
00019
00023         public void InitializeBle()
00024         {
00025             if (_deviceWatcher != null && _deviceWatcher.Status == DeviceWatcherStatus.Started)
```

```
00026         {
00027             return;
00028         }
00029 
00030         // Define properties to retrieve for each device
00031         string[] requestedProperties = { "System.Devices.Aep.DeviceAddress", "System.Devices.Aep.IsConnected" };
00032         // Filter for BLE devices using the BLE protocol ID
00033         string aqsFilter = "(System.Devices.Aep.ProtocolId:=\"{bb7bb05e-5972-42b5-94fc-76eaa7084d49}\")";
00034 
00035         // Create a device watcher for BLE association endpoints
00036         _deviceWatcher = DeviceInformation.CreateWatcher(
00037             aqsFilter,
00038             requestedProperties,
00039             DeviceInformationKind.AssociationEndpoint);
00040 
00041         // Attach event handlers
00042         _deviceWatcher.Added += DeviceWatcher_Added;
00043         _deviceWatcher.Updated += DeviceWatcher_Updated;
00044         _deviceWatcher.Removed += DeviceWatcher_Removed;
00045 
00046         _deviceWatcher.Start();
00047     }
00048 
00054     private void DeviceWatcher_Added(DeviceWatcher sender, DeviceInformation deviceInfo)
00055     {
00056         System.Windows.Application.Current.Dispatcher.Invoke(() =>
00057         {
00058             try
00059             {
00060                 // Only add devices with a non-empty name
00061                 if (!string.IsNullOrWhiteSpace(deviceInfo.Name))
00062                 {
00063                     _devices.Add(deviceInfo);
00064                 }
00065             }
00066             catch (Exception ex)
00067             {
00068                 Debug.WriteLine($"BLE Device Watcher Error adding device: {ex.Message}");
00069             }
00070         });
00071     }
00072 
00078     private void DeviceWatcher_Updated(DeviceWatcher sender, DeviceInformationUpdate deviceInfoUpdate)
00079     {
00080         System.Windows.Application.Current.Dispatcher.Invoke(() =>
00081         {
00082             try
00083             {
00084                 var device = _devices.FirstOrDefault(d => d.Id == deviceInfoUpdate.Id);
00085                 if (device != null)
00086                 {
00087                     device.Update(deviceInfoUpdate);
00088                 }
00089             }
00090             catch (Exception ex)
00091             {
00092                 Debug.WriteLine($"BLE Device Watcher Error updating device: {ex.Message}");
00093             }
00094         });
00095     }
00096 
00102     private void DeviceWatcher_Removed(DeviceWatcher sender, DeviceInformationUpdate deviceInfoUpdate)
00103     {
00104         System.Windows.Application.Current.Dispatcher.Invoke(() =>
00105         {
00106             try
00107             {
00108                 var device = _devices.FirstOrDefault(d => d.Id == deviceInfoUpdate.Id);
00109                 if (device != null)
00110                 {
00111                     _devices.Remove(device);
00112                 }
00113             }
00114             catch (Exception ex)
00115             {
00116                 Debug.WriteLine($"BLE Device Watcher Error removing device: {ex.Message}");
00117             }
00118         });
00119     }
00120 
00124     public void StopWatcher()
00125     {
00126         if (_deviceWatcher != null && _deviceWatcher.Status != DeviceWatcherStatus.Stopped)
00127         {
00128             _deviceWatcher.Stop();
00129         }
00130     }
```

```
00131
00135         public void Dispose()
00136         {
00137             StopWatcher();
00138             if (_deviceWatcher != null)
00139             {
00140                 _deviceWatcher.Added -= DeviceWatcher_Added;
00141                 _deviceWatcher.Updated -= DeviceWatcher_Updated;
00142                 _deviceWatcher.Removed -= DeviceWatcher_Removed;
00143                 _deviceWatcher = null;
00144             }
00145             GC.SuppressFinalize(this);
00146         }
00147     }
00148 }
```

7.55 ladder_diagram_app/Services/CommunicationServices/CommunicationServiceFactory.cs File Reference

Classes

- class ladder_diagram_app.Services.CommunicationServices.CommunicationServiceFactory

  Factory class for creating instances of IDeviceCommunicationService based on the specified connection type.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CommunicationServices

7.56 CommunicationServiceFactory.cs

Go to the documentation of this file.

```
00001 using ladder_diagram_app.Services.CommunicationServices.MQTT;
00002 using ladder_diagram_app.Services.CommunicationServices.BLE;
00003
00004 namespace ladder_diagram_app.Services.CommunicationServices
00005 {
00009     public static class CommunicationServiceFactory
00010     {
00017         public static IDeviceCommunicationService CreateService(string connectionType)
00018         {
00019             return connectionType switch
00020             {
00021                 "MQTT" => new MqttCommunicationService(),
00022                 "BLE" => new BleCommunicationService(),
00023                 _ => throw new ArgumentException("Unsupported connection type", nameof(connectionType))
00024             };
00025         }
00026     }
00027 }
```

7.57 ladder_diagram_app/Services/CommunicationServices/DeviceCommunicationManager.cs File Reference

Classes

- class ladder_diagram_app.Services.CommunicationServices.DeviceCommunicationManager

  Manages device communication, handling connection, disconnection, and configuration exchange for MQTT and BLE protocols.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CommunicationServices

7.58 DeviceCommunicationManager.cs

Go to the documentation of this file.

```
00001 using System.Windows;
00002 using System.Windows.Threading;
00003 using ladder_diagram_app.Services.CommunicationServices.BLE.BluetoothSelection;
00004 using ladder_diagram_app.Views;
00005
00006 namespace ladder_diagram_app.Services.CommunicationServices
00007 {
00011     public class DeviceCommunicationManager : IDisposable
00012     {
00013         public IDeviceCommunicationService? _communicationService { get; set; }
00014         private readonly BleDeviceWatcher _bleDeviceWatcher;
00015         private readonly Action<string> _onConfigurationReceived;
00016         private readonly Action<bool> _onConnectionStatusChanged;
00017         private readonly Action<string> _onMonitorDataReceived;
00018         private readonly Action<string> _onOneWireDataReceived;
00019
00028         public DeviceCommunicationManager(
00029             Action<string> onConfigurationReceived,
00030             Action<bool> onConnectionStatusChanged,
00031             Action<string> onMonitorDataReceived,
00032             Action<string> onOneWireDataReceived)
00033         {
00034             _onConfigurationReceived = onConfigurationReceived ?? throw new
    ArgumentNullException(nameof(onConfigurationReceived));
00035             _onConnectionStatusChanged = onConnectionStatusChanged ?? throw new
    ArgumentNullException(nameof(onConnectionStatusChanged));
00036             _onMonitorDataReceived = onMonitorDataReceived ?? throw new
    ArgumentNullException(nameof(onMonitorDataReceived));
00037             _onOneWireDataReceived = onOneWireDataReceived ?? throw new
    ArgumentNullException(nameof(onOneWireDataReceived));
00038             _bleDeviceWatcher = new BleDeviceWatcher();
00039         }
00040
00047         private string GetDeviceId(Window owner, string connectionType)
00048         {
00049             if (connectionType == "MQTT")
00050             {
00051                 var macInput = new NotificationWindow("Enter the device's MAC address", owner,
    NotificationButtons.OneInput, new[] { "" });
00052                 macInput.ShowDialog();
00053                 if (macInput.Result == true) return macInput.InputResults[0].ToUpper();
00054             }
00055             else
00056             {
00057                 _bleDeviceWatcher.InitializeBle();
00058                 var deviceSelection = new BleDeviceSelectionWindow(_bleDeviceWatcher.Devices, owner);
00059                 if (deviceSelection.ShowDialog() == true)
00060                 {
00061                     var selectedDevice = deviceSelection.SelectedDevice;
00062                     _bleDeviceWatcher.StopWatcher();
00063                     _bleDeviceWatcher.Dispose();
00064                     return selectedDevice?.Id ?? string.Empty;
00065                 }
00066                 _bleDeviceWatcher.StopWatcher();
00067                 _bleDeviceWatcher.Dispose();
00068             }
00069             return string.Empty;
00070         }
00071
00077         public async Task ConnectAsync(Window owner, string connectionType)
00078         {
00079             try
00080             {
00081                 if (_communicationService != null && _communicationService.IsConnected)
00082                 {
00083                     new NotificationWindow("Device is already connected", owner).Show();
00084                     return;
00085                 }
00086
00087                 _communicationService = CommunicationServiceFactory.CreateService(connectionType);
```

```
00088
00089             if (_communicationService != null)
00090             {
00091                 _communicationService.ConfigurationReceived -= (s, json) => _onConfigurationReceived(json);
00092                 _communicationService.MonitorDataReceived -= (s, json) => _onMonitorDataReceived(json);
00093                 _communicationService.OneWireDataReceived -= (s, json) => _onOneWireDataReceived(json);
00094                 _communicationService.ConnectionStatusChanged -= (s, isConnected) =>
    _onConnectionStatusChanged(isConnected);
00095             }
00096
00097             _communicationService.ConfigurationReceived += (s, json) => owner.Dispatcher.InvokeAsync(() =>
    _onConfigurationReceived(json));
00098             _communicationService.MonitorDataReceived += (s, json) => owner.Dispatcher.InvokeAsync(() =>
    _onMonitorDataReceived(json));
00099             _communicationService.OneWireDataReceived += (s, json) => owner.Dispatcher.InvokeAsync(() =>
    _onOneWireDataReceived(json));
00100             _communicationService.ConnectionStatusChanged += (s, isConnected) => owner.Dispatcher.InvokeAsync(()
    => _onConnectionStatusChanged(isConnected));
00101
00102
00103             string deviceId = GetDeviceId(owner, connectionType);
00104             if (string.IsNullOrEmpty(deviceId)) return;
00105
00106             bool success = await _communicationService.ConnectAsync(deviceId);
00107             if (!success) new NotificationWindow("Device Connected Unsuccessfully", owner).Show();
00108         }
00109         catch (Exception ex)
00110         {
00111             new NotificationWindow($"Connection failed: {ex.Message}", owner).Show();
00112         }
00113     }
00114
00119     public async Task DisconnectAsync(Window owner)
00120     {
00121         if (_communicationService != null && _communicationService.IsConnected)
00122         {
00123             await _communicationService.DisconnectAsync();
00124         }
00125         else
00126         {
00127             new NotificationWindow("Device is not connected", owner).Show();
00128         }
00129     }
00130
00136     public async Task SendConfigurationAsync(string jsonConfig, Window owner)
00137     {
00138         if (_communicationService == null || !_communicationService.IsConnected)
00139         {
00140             new NotificationWindow("Device is not connected", owner).Show();
00141             return;
00142         }
00143
00144         bool success = await _communicationService.SendConfigurationAsync(jsonConfig);
00145         new NotificationWindow(success ? "Configuration sent successfully" : "Configuration sent unsuccessfully",
    owner).Show();
00146     }
00147
00151     public void Dispose()
00152     {
00153         if (_communicationService != null)
00154         {
00155             _communicationService.ConfigurationReceived -= (s, json) => _onConfigurationReceived(json);
00156             _communicationService.MonitorDataReceived -= (s, json) => _onMonitorDataReceived(json);
00157             _communicationService.OneWireDataReceived -= (s, json) => _onOneWireDataReceived(json);
00158             _communicationService.ConnectionStatusChanged -= (s, isConnected) =>
    _onConnectionStatusChanged(isConnected);
00159             _communicationService.Dispose();
00160             _communicationService = null;
00161         }
00162         _bleDeviceWatcher?.Dispose();
00163     }
00164 }
00165 }
```

7.59 ladder_diagram_app/Services/CommunicationServices/IDeviceCommunicationService.cs File Reference

Classes

- interface ladder_diagram_app.Services.CommunicationServices.IDeviceCommunicationService

Defines the contract for device communication services, supporting connection management and data exchange.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CommunicationServices

7.60 IDeviceCommunicationService.cs

Go to the documentation of this file.

```
00001 namespace ladder_diagram_app.Services.CommunicationServices
00002 {
00006     public interface IDeviceCommunicationService : IDisposable
00007     {
00011         event EventHandler<string> ConfigurationReceived;
00012
00016         event EventHandler<string> MonitorDataReceived;
00017
00021         event EventHandler<string> OneWireDataReceived;
00022
00026         event EventHandler<bool> ConnectionStatusChanged;
00027
00031         bool IsConnected { get; }
00032
00036         string ConnectionType { get; }
00037
00043         Task<bool> ConnectAsync(string deviceIdentifier);
00044
00048         Task DisconnectAsync();
00049
00055         Task<bool> SendConfigurationAsync(string configJson);
00056
00060         Task RequestConfigurationAsync();
00061     }
00062 }
```

7.61 ladder_diagram_app/Services/CommunicationServices/MQTT/↩ MqttCommunicationService.cs File Reference

Classes

- class ladder_diagram_app.Services.CommunicationServices.MQTT.MqttCommunicationService

  Provides MQTT communication services for connecting to and interacting with a device using MQTT protocol.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.CommunicationServices
- namespace ladder_diagram_app.Services.CommunicationServices.MQTT

7.62 MqttCommunicationService.cs

Go to the documentation of this file.

```
00001 using System.Diagnostics;
00002 using System.Text;
00003 using MQTTnet;
00004 using MQTTnet.Client;
00005 using MQTTnet.Protocol;
00006 using System.Configuration;
00007
00008 namespace ladder_diagram_app.Services.CommunicationServices.MQTT
00009 {
00013     public class MqttCommunicationService : IDeviceCommunicationService, IDisposable
00014     {
00015         private readonly IMqttClient _mqttClient;
00016         private string _macAddress = string.Empty;
00017
00018         public static readonly string BrokerAddress = ConfigurationManager.AppSettings["MqttBrokerAddress"] ?? throw
        new ConfigurationErrorsException("MqttBrokerAddress is missing in App.config");
00019         public static readonly int BrokerPort = int.Parse(ConfigurationManager.AppSettings["MqttBrokerPort"] ?? throw
        new ConfigurationErrorsException("MqttBrokerPort is missing in App.config"));
00020         public static readonly string? BrokerUsername = ConfigurationManager.AppSettings["MqttBrokerUsername"];
00021         public static readonly string? BrokerPassword = ConfigurationManager.AppSettings["MqttBrokerPassword"];
00022
00023         private const string MqttTopicConnectionRequest = "/connection_request";
00024         private const string MqttTopicConnectionResponse = "/connection_response";
00025         private const string MqttTopicMonitor = "/monitor";
00026         private const string MqttTopicOneWire = "/one_wire";
00027         private const string MqttTopicConfigRequest = "/config_request";
00028         private const string MqttTopicConfigResponse = "/config_response";
00029         private const string MqttTopicConfig = "/config_device";
00030
00031         public event EventHandler<string>? ConfigurationReceived;
00032         public event EventHandler<string>? MonitorDataReceived;
00033         public event EventHandler<string>? OneWireDataReceived;
00034         public event EventHandler<bool>? ConnectionStatusChanged;
00035
00039         public bool IsConnected { get; private set; }
00040
00044         public string ConnectionType => "MQTT";
00045
00046         private System.Timers.Timer? _presentTimer;
00047         private DateTime? _lastMonitorMessageTime = null;
00048         private const int ChunkSize = 800; // Adjusted for ESP32-S3 buffer
00049         private bool _disposed = false;
00050
00054         public MqttCommunicationService()
00055         {
00056             var factory = new MqttFactory();
00057             _mqttClient = factory.CreateMqttClient();
00058             SetupEventHandlers();
00059         }
00060
00064         private void SetupEventHandlers()
00065         {
00066             _mqttClient.ConnectedAsync += async e =>
00067             {
00068                 await SubscribeToTopics();
00069                 await RequestConnectionAsync();
00070             };
00071
00072             _mqttClient.DisconnectedAsync += e =>
00073             {
00074                 IsConnected = false;
00075                 _presentTimer?.Stop();
00076                 _presentTimer?.Dispose();
00077                 ConnectionStatusChanged?.Invoke(this, false);
00078                 return Task.CompletedTask;
00079             };
00080
00081             _mqttClient.ApplicationMessageReceivedAsync += e =>
00082             {
00083                 var message = Encoding.UTF8.GetString(e.ApplicationMessage.PayloadSegment);
00084                 HandleIncomingMessage(e.ApplicationMessage.Topic, message);
00085                 return Task.CompletedTask;
00086             };
00087         }
00088
00094         public async Task<bool> ConnectAsync(string deviceId)
00095         {
00096             try
00097             {
00098                 if (IsConnected) return true;
00099
00100                 if (string.IsNullOrWhiteSpace(deviceId))
```

```
00101             {
00102                 Debug.WriteLine("MQTT Connection failed: Device MAC Address is null or empty");
00103                 return false;
00104             }
00105
00106             _macAddress = deviceId.ToUpper();
00107
00108             var mqttClientOptionsBuilder = new MqttClientOptionsBuilder()
00109                 .WithTcpServer(BrokerAddress, BrokerPort)
00110                 .WithClientId(Guid.NewGuid().ToString());
00111
00112             if (!string.IsNullOrEmpty(BrokerUsername) && !string.IsNullOrEmpty(BrokerPassword))
00113             {
00114                 mqttClientOptionsBuilder = mqttClientOptionsBuilder.WithCredentials(BrokerUsername, BrokerPassword);
00115             }
00116
00117             var mqttClientOptions = mqttClientOptionsBuilder.Build();
00118
00119             int retries = 3;
00120             for (int i = 0; i < retries; i++)
00121             {
00122                 try
00123                 {
00124                     await _mqttClient.ConnectAsync(mqttClientOptions);
00125                     return true;
00126                 }
00127                 catch (Exception ex)
00128                 {
00129                     if (i == retries - 1)
00130                     {
00131                         Debug.WriteLine($"MQTT Connection failed after {retries} retries: {ex.Message}");
00132                         return false;
00133                     }
00134                     await Task.Delay(1000);
00135                 }
00136             }
00137             return false;
00138         }
00139         catch (Exception ex)
00140         {
00141             Debug.WriteLine($"MQTT Connection setup failed: {ex.Message}");
00142             return false;
00143         }
00144     }
00145
00149     public async Task DisconnectAsync()
00150     {
00151         try
00152         {
00153             if (!IsConnected || _mqttClient == null) return;
00154
00155             await UnsubscribeFromTopics();
00156
00157             var message = new MqttApplicationMessageBuilder()
00158                 .WithTopic($"{_macAddress}{MqttTopicConnectionRequest}")
00159                 .WithPayload("Disconnect")
00160                 .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.AtLeastOnce)
00161                 .Build();
00162             await _mqttClient.PublishAsync(message);
00163
00164             await _mqttClient.DisconnectAsync();
00165         }
00166         catch (Exception ex)
00167         {
00168             Debug.WriteLine($"MQTT Disconnection failed: {ex.Message}");
00169         }
00170         finally
00171         {
00172             IsConnected = false;
00173             _presentTimer?.Stop();
00174             _presentTimer?.Dispose();
00175             ConnectionStatusChanged?.Invoke(this, false);
00176         }
00177     }
00178
00182     private async Task SubscribeToTopics()
00183     {
00184         try
00185         {
00186             var subscribeOptions = new MqttClientSubscribeOptionsBuilder()
00187                 .WithTopicFilter($"{_macAddress}{MqttTopicConnectionResponse}",
    MqttQualityOfServiceLevel.AtLeastOnce)
00188                 .WithTopicFilter($"{_macAddress}{MqttTopicConfigResponse}", MqttQualityOfServiceLevel.AtLeastOnce)
00189                 .WithTopicFilter($"{_macAddress}{MqttTopicMonitor}", MqttQualityOfServiceLevel.AtLeastOnce)
00190                 .WithTopicFilter($"{_macAddress}{MqttTopicOneWire}", MqttQualityOfServiceLevel.AtLeastOnce)
00191                 .Build();
00192             await _mqttClient.SubscribeAsync(subscribeOptions);
```

```
00193           }
00194           catch (Exception ex)
00195           {
00196               Debug.WriteLine($"MQTT Subscription failed: {ex.Message}");
00197               await DisconnectAsync();
00198           }
00199       }
00200
00204       private async Task UnsubscribeFromTopics()
00205       {
00206           try
00207           {
00208               var unsubscribeOptions = new MqttClientUnsubscribeOptionsBuilder()
00209                   .WithTopicFilter($"{_macAddress}{MqttTopicConnectionResponse}")
00210                   .WithTopicFilter($"{_macAddress}{MqttTopicMonitor}")
00211                   .WithTopicFilter($"{_macAddress}{MqttTopicOneWire}")
00212                   .WithTopicFilter($"{_macAddress}{MqttTopicConfigResponse}")
00213                   .Build();
00214               await _mqttClient.UnsubscribeAsync(unsubscribeOptions);
00215           }
00216           catch (Exception ex)
00217           {
00218               Debug.WriteLine($"MQTT Unsubscription failed: {ex.Message}");
00219           }
00220       }
00221
00227       private void HandleIncomingMessage(string topic, string message)
00228       {
00229           if (string.IsNullOrEmpty(_macAddress) || string.IsNullOrEmpty(topic)) return;
00230
00231           if (topic == $"{_macAddress}{MqttTopicConnectionResponse}" && message == "Connected")
00232           {
00233               IsConnected = true;
00234               ConnectionStatusChanged?.Invoke(this, true);
00235               _lastMonitorMessageTime = DateTime.Now;
00236               InitializePresentTimer();
00237               Task.Run(() => RequestConfigurationAsync());
00238           }
00239           else if (topic == $"{_macAddress}{MqttTopicMonitor}")
00240           {
00241               _lastMonitorMessageTime = DateTime.Now;
00242               MonitorDataReceived?.Invoke(this, message);
00243           }
00244           else if (topic == $"{_macAddress}{MqttTopicConfigResponse}")
00245           {
00246               ConfigurationReceived?.Invoke(this, message);
00247           }
00248           else if (topic == $"{_macAddress}{MqttTopicOneWire}")
00249           {
00250               OneWireDataReceived?.Invoke(this, message);
00251           }
00252       }
00253
00257       public async Task RequestConnectionAsync()
00258       {
00259           try
00260           {
00261               var message = new MqttApplicationMessageBuilder()
00262                   .WithTopic($"{_macAddress}{MqttTopicConnectionRequest}")
00263                   .WithPayload("Connect")
00264                   .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.AtLeastOnce)
00265                   .Build();
00266
00267               await _mqttClient.PublishAsync(message);
00268           }
00269           catch (Exception ex)
00270           {
00271               Debug.WriteLine($"MQTT Requesting Connection failed: {ex.Message}");
00272               await DisconnectAsync();
00273           }
00274       }
00275
00279       public async Task RequestConfigurationAsync()
00280       {
00281           try
00282           {
00283               var message = new MqttApplicationMessageBuilder()
00284                   .WithTopic($"{_macAddress}{MqttTopicConfigRequest}")
00285                   .WithPayload("Request Configuration")
00286                   .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.AtLeastOnce)
00287                   .Build();
00288
00289               await _mqttClient.PublishAsync(message);
00290           }
00291           catch (Exception ex)
00292           {
00293               Debug.WriteLine($"MQTT Requesting Configuration failed: {ex.Message}");
```

```
00294             await DisconnectAsync();
00295         }
00296     }
00297
00301     private void InitializePresentTimer()
00302     {
00303         _presentTimer = new System.Timers.Timer(1000) { AutoReset = true };
00304         _presentTimer.Elapsed += async (s, args) =>
00305         {
00306             try
00307             {
00308                 if (IsConnected)
00309                 {
00310                     if (_lastMonitorMessageTime.HasValue && (DateTime.Now -
    _lastMonitorMessageTime.Value).TotalSeconds > 10)
00311                     {
00312                         await _mqttClient.DisconnectAsync();
00313                         return;
00314                     }
00315
00316                     var message = new MqttApplicationMessageBuilder()
00317                         .WithTopic($"{_macAddress}{MqttTopicConnectionRequest}")
00318                         .WithPayload("Present")
00319                         .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.AtLeastOnce)
00320                         .Build();
00321                     await _mqttClient.PublishAsync(message);
00322                 }
00323             }
00324             catch (Exception ex)
00325             {
00326                 Debug.WriteLine($"Error in MQTT timer callback: {ex.Message}");
00327             }
00328         };
00329         _presentTimer.Start();
00330     }
00331
00337     public async Task<bool> SendConfigurationAsync(string configJson)
00338     {
00339         try
00340         {
00341             if (!IsConnected)
00342             {
00343                 Debug.WriteLine("MQTT Sending Configuration failed: Not connected");
00344                 return false;
00345             }
00346
00347             if (string.IsNullOrEmpty(configJson))
00348             {
00349                 Debug.WriteLine("MQTT Sending Configuration failed: Config JSON is null or empty");
00350                 return false;
00351             }
00352
00353             for (int i = 0; i < configJson.Length; i += ChunkSize)
00354             {
00355                 int chunkLength = Math.Min(ChunkSize, configJson.Length - i);
00356                 string chunk = configJson.Substring(i, chunkLength);
00357
00358                 var message = new MqttApplicationMessageBuilder()
00359                     .WithTopic($"{_macAddress}{MqttTopicConfig}")
00360                     .WithPayload(chunk)
00361                     .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.AtLeastOnce)
00362                     .Build();
00363
00364                 await _mqttClient.PublishAsync(message);
00365                 await Task.Delay(50); // Delay for reliable reception
00366             }
00367
00368             Debug.WriteLine("Configuration sent successfully");
00369             return true;
00370         }
00371         catch (Exception ex)
00372         {
00373             Debug.WriteLine($"MQTT Sending Configuration failed: {ex.Message}");
00374             return false;
00375         }
00376     }
00377
00382     protected virtual void Dispose(bool disposing)
00383     {
00384         if (!_disposed)
00385         {
00386             if (disposing)
00387             {
00388                 _presentTimer?.Stop();
00389                 _presentTimer?.Dispose();
00390                 if (_mqttClient != null && _mqttClient.IsConnected)
00391                 {
```

```
00392                     try
00393                     {
00394                         _ = _mqttClient.DisconnectAsync();
00395                     }
00396                     catch (Exception ex)
00397                     {
00398                         Debug.WriteLine($"MQTT Dispose disconnect failed: {ex.Message}");
00399                     }
00400                 }
00401                 _mqttClient?.Dispose();
00402             }
00403             _disposed = true;
00404         }
00405     }
00406
00410     public void Dispose()
00411     {
00412         Dispose(true);
00413         GC.SuppressFinalize(this);
00414     }
00415 }
00416 }
```

7.63 ladder_diagram_app/Services/ImportExportServices/ImportExportService.cs File Reference

Classes

- class ladder_diagram_app.Services.ImportExportServices.ImportExportService
  Handles importing and exporting of ladder diagram configurations to and from JSON format.
- class ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportData
- class ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportWire
- class ladder_diagram_app.Services.ImportExportServices.ImportExportService.ExportNode

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.ImportExportServices

7.64 ImportExportService.cs

Go to the documentation of this file.

```
00001 using System.Text.Json;
00002 using ladder_diagram_app.Models.CanvasElements;
00003 using ladder_diagram_app.Models.DeviceElement;
00004 using ladder_diagram_app.Models.Variables;
00005 using ladder_diagram_app.Views;
00006 using ladder_diagram_app.Models.CanvasElements.Instances;
00007 using System.Diagnostics;
00008 using ladder_diagram_app.Services.CanvasServices;
00009
00010 namespace ladder_diagram_app.Services.ImportExportServices
00011 {
00015     public class ImportExportService
00016     {
00017         private readonly VariablesManager _variablesManager;
00018         private readonly Device _device;
00019         private readonly WiresManager _wiresManager;
00020         private readonly CanvasManager _canvasManager;
00021         private readonly DevicePinManager _devicePinManager;
00022         private bool _abortExport;
00023
00032         public ImportExportService(
00033             VariablesManager variablesManager,
00034             Device device,
```

```
00035             WiresManager wiresManager,
00036             CanvasManager canvasManager,
00037             DevicePinManager devicePinManager)
00038         {
00039             _variablesManager = variablesManager;
00040             _device = device;
00041             _wiresManager = wiresManager;
00042             _canvasManager = canvasManager;
00043             _devicePinManager = devicePinManager;
00044         }
00045
00051         public string? ExportToJson(MainWindow owner)
00052         {
00053             var exportData = PrepareExportData(owner);
00054             if (exportData == null) return null;
00055
00056             try
00057             {
00058                 return JsonSerializer.Serialize(exportData);
00059             }
00060             catch (Exception ex)
00061             {
00062                 new NotificationWindow($"Export failed: {ex.Message}", owner).Show();
00063                 return null;
00064             }
00065         }
00066
00072         public void ImportFromJson(string jsonString, MainWindow owner)
00073         {
00074             try
00075             {
00076                 var importData = JsonSerializer.Deserialize<ExportData>(jsonString);
00077                 if (importData == null)
00078                 {
00079                     new NotificationWindow("Failed to parse JSON data.", owner).Show();
00080                     return;
00081                 }
00082
00083                 // Update Device
00084                 if (importData.Device != null)
00085                 {
00086                     _device.UpdateFrom(importData.Device);
00087                     if (!_device.Validate())
00088                     {
00089                         new NotificationWindow("Imported device data is invalid.", owner).Show();
00090                         return;
00091                     }
00092                     _devicePinManager.UpdateDevicePinOptions(_device);
00093                 }
00094                 else
00095                 {
00096                     new NotificationWindow("Imported device data is null.", owner).Show();
00097                     return;
00098                 }
00099
00100                 // Clear existing data
00101                 _variablesManager.ClearVariablesList();
00102                 _variablesManager.Device = _device;
00103                 _wiresManager.ClearWires();
00104                 owner.MainCanvas.Children.Clear();
00105
00106                 // Import Variables
00107                 if (importData.Variables != null)
00108                 {
00109                     foreach (var varData in importData.Variables)
00110                     {
00111                         if (!varData.TryGetValue("Type", out var typeObj) || typeObj == null ||
00112                             !varData.TryGetValue("Name", out var nameObj) || nameObj == null)
00113                         {
00114                             Debug.WriteLine("Skipping variable: Type or Name is missing or null");
00115                             continue;
00116                         }
00117
00118                         string type = ((JsonElement)typeObj).GetString() ?? string.Empty;
00119                         string name = ((JsonElement)nameObj).GetString() ?? string.Empty;
00120
00121                         if (string.IsNullOrEmpty(type) || string.IsNullOrEmpty(name))
00122                         {
00123                             Debug.WriteLine($"Skipping variable: Type='{type}', Name='{name}' is empty");
00124                             continue;
00125                         }
00126
00127                         string pinName = varData.TryGetValue("Pin", out var pinObj) && pinObj is JsonElement pinJe &&
    pinJe.ValueKind == JsonValueKind.String ? pinJe.ToString() : "";
00128                         string sensorType = varData.TryGetValue("Sensor Type", out var stObj) && stObj is JsonElement stJe
    && stJe.ValueKind == JsonValueKind.String ? stJe.ToString() : "";
00129                         string pdSck = varData.TryGetValue("PD_SCK", out var pdObj) && pdObj is JsonElement pdJe &&
```

```
pdJe.ValueKind == JsonValueKind.String ? pdJe.ToString() : "";
00130                   string dout = varData.TryGetValue("DOUT", out var doutObj) && doutObj is JsonElement doutJe &&
doutJe.ValueKind == JsonValueKind.String ? doutJe.ToString() : "";
00131                   string samplingRate = varData.TryGetValue("Sampling Rate", out var srObj) && srObj is JsonElement
srJe && srJe.ValueKind == JsonValueKind.String ? srJe.ToString() : "";
00132                   double mapLow = varData.TryGetValue("Map Low", out var mlObj) && mlObj is JsonElement mlJe &&
mlJe.ValueKind == JsonValueKind.Number ? mlJe.GetDouble() : 0.0;
00133                   double mapHigh = varData.TryGetValue("Map High", out var mhObj) && mhObj is JsonElement mhJe
&& mhJe.ValueKind == JsonValueKind.Number ? mhJe.GetDouble() : 0.0;
00134                   double gain = varData.TryGetValue("Gain", out var gainObj) && gainObj is JsonElement gainJe &&
gainJe.ValueKind == JsonValueKind.Number ? gainJe.GetDouble() : 0.0;
00135                   bool boolValue = varData.TryGetValue("Value", out var boolObj) && boolObj is JsonElement boolJe
&& boolJe.ValueKind == JsonValueKind.True;
00136                   double numValue = varData.TryGetValue("Value", out var numObj) && numObj is JsonElement numJe
&& numJe.ValueKind == JsonValueKind.Number ? numJe.GetDouble() : 0.0;
00137                   double timeValue = varData.TryGetValue("Value", out var timeObj) && timeObj is JsonElement timeJe
&& timeJe.ValueKind == JsonValueKind.Number ? timeJe.GetDouble() : 0.0;
00138                   double pv = varData.TryGetValue("PV", out var pvObj) && pvObj is JsonElement pvJe &&
pvJe.ValueKind == JsonValueKind.Number ? pvJe.GetDouble() : 0.0;
00139                   double cv = varData.TryGetValue("CV", out var cvObj) && cvObj is JsonElement cvJe &&
cvJe.ValueKind == JsonValueKind.Number ? cvJe.GetDouble() : 0.0;
00140                   bool cu = varData.TryGetValue("CU", out var cuObj) && cuObj is JsonElement cuJe &&
cuJe.ValueKind == JsonValueKind.True;
00141                   bool cd = varData.TryGetValue("CD", out var cdObj) && cdObj is JsonElement cdJe &&
cdJe.ValueKind == JsonValueKind.True;
00142                   double pt = varData.TryGetValue("PT", out var ptObj) && ptObj is JsonElement ptJe &&
ptJe.ValueKind == JsonValueKind.Number ? ptJe.GetDouble() : 0.0;
00143                   double et = varData.TryGetValue("ET", out var etObj) && etObj is JsonElement etJe &&
etJe.ValueKind == JsonValueKind.Number ? etJe.GetDouble() : 0.0;
00144
00145                   _variablesManager.AddVariable(
00146                       name: name,
00147                       type: type,
00148                       pinName: pinName,
00149                       sensorType: sensorType,
00150                       pdSck: pdSck,
00151                       dout: dout,
00152                       samplingRate: samplingRate,
00153                       mapLow: mapLow,
00154                       mapHigh: mapHigh,
00155                       gain: gain,
00156                       boolValue: boolValue,
00157                       numValue: numValue,
00158                       pv: pv,
00159                       cv: cv,
00160                       cu: cu,
00161                       cd: cd,
00162                       pt: pt,
00163                       et: et,
00164                       timeValue: timeValue,
00165                       owner: owner
00166                   );
00167               }
00168           }
00169
00170           // Import Wires
00171           if (importData.Wires != null)
00172           {
00173               foreach (var wireData in importData.Wires)
00174               {
00175                   var wire = new Wire();
00176                   if (wireData.Nodes != null)
00177                   {
00178                       wire.Nodes = ImportNodes(wireData.Nodes, wire);
00179                       _wiresManager.AddWire(wire);
00180                   }
00181               }
00182           }
00183
00184           _canvasManager.UpdateCanvas();
00185       }
00186       catch (Exception ex)
00187       {
00188           new NotificationWindow($"Import failed", owner).Show();
00189           Debug.WriteLine($"Import failed: {ex.Message}");
00190       }
00191   }
00192
00198   private ExportData? PrepareExportData(MainWindow owner)
00199   {
00200       if (!_variablesManager.ValidateVariables(owner))
00201           return null;
00202
00203       _abortExport = false;
00204
00205       var exportData = new ExportData
00206       {
```

```
00207               Device = _device,
00208               Variables = _variablesManager.VariablesList
00209                   .Where(v => !v.Name.Contains("      "))
00210                   .Select(v => v.ToExportDictionary())
00211                   .Where(dict => dict != null)
00212                   .ToList(),
00213               Wires = _wiresManager.Wires.Select(w => new ExportWire
00214               {
00215                   Nodes = ExportNodes(w.Nodes)
00216               }).ToList()
00217           };
00218
00219           if (_abortExport)
00220           {
00221               new NotificationWindow("All element comboboxes must have a selected value", owner).Show();
00222               return null;
00223           }
00224
00225           return exportData;
00226       }
00227
00233       private List<ExportNode> ExportNodes(List<Node> nodes)
00234       {
00235           var exportNodes = new List<ExportNode>();
00236           foreach (var node in nodes)
00237           {
00238               if (node is LadderElement element)
00239               {
00240                   exportNodes.Add(new ExportNode
00241                   {
00242                       Type = "LadderElement",
00243                       ElementType = element.Type,
00244                       ComboBoxValues = element.VariableComboBoxes.Select(cb => cb.SelectedItem?.ToString() ??
    string.Empty).ToList()
00245                   });
00246                   if (element.VariableComboBoxes.Any(cb => cb.SelectedIndex == -1))
00247                   {
00248                       _abortExport = true;
00249                   }
00250               }
00251               else if (node is Branch branch)
00252               {
00253                   exportNodes.Add(new ExportNode
00254                   {
00255                       Type = "Branch",
00256                       Nodes1 = ExportNodes(branch.Nodes1),
00257                       Nodes2 = ExportNodes(branch.Nodes2)
00258                   });
00259               }
00260           }
00261           return exportNodes;
00262       }
00263
00270       private List<Node> ImportNodes(List<ExportNode> exportNodes, Wire? parentWire = null)
00271       {
00272           var nodes = new List<Node>();
00273           foreach (var exportNode in exportNodes)
00274           {
00275               if (exportNode.Type == "LadderElement")
00276               {
00277                   if (exportNode.ElementType == null)
00278                   {
00279                       Debug.WriteLine("Skipping LadderElement: ElementType is null");
00280                       continue;
00281                   }
00282                   var element = new LadderElement(
00283                       exportNode.ElementType,
00284                       _variablesManager.VariablesListContacts,
00285                       _variablesManager.VariablesListCoils,
00286                       _variablesManager.VariablesListMath,
00287                       _variablesManager.VariablesListCompare,
00288                       _variablesManager.VariablesListCounter,
00289                       _variablesManager.VariablesListTimer,
00290                       _variablesManager.VariablesListReset
00291                   );
00292
00293                   if (exportNode.ComboBoxValues != null)
00294                   {
00295                       for (int i = 0; i < exportNode.ComboBoxValues.Count && i < element.VariableComboBoxes.Count; i++)
00296                       {
00297                           if (!string.IsNullOrEmpty(exportNode.ComboBoxValues[i]))
00298                           {
00299                               element.VariableComboBoxes[i].SelectedItem = exportNode.ComboBoxValues[i];
00300                           }
00301                       }
00302                   }
00303
```

```
00304                 element.Parent = parentWire;
00305                 nodes.Add(element);
00306             }
00307             else if (exportNode.Type == "Branch")
00308             {
00309                 var branch = new Branch();
00310                 branch.Parent = parentWire;
00311                 branch.Nodes1 = exportNode.Nodes1 != null ? ImportNodes(exportNode.Nodes1, parentWire) : new
    List<Node>();
00312                 branch.Nodes2 = exportNode.Nodes2 != null ? ImportNodes(exportNode.Nodes2, parentWire) : new
    List<Node>();
00313                 foreach (var node in branch.Nodes1) node.Parent = branch;
00314                 foreach (var node in branch.Nodes2) node.Parent = branch;
00315                 nodes.Add(branch);
00316             }
00317         }
00318         return nodes;
00319     }
00320
00321     private class ExportData
00322     {
00323         public Device? Device { get; set; }
00324         public List<Dictionary<string, object»? Variables { get; set; }
00325         public List<ExportWire>? Wires { get; set; }
00326     }
00327
00328     private class ExportWire
00329     {
00330         public List<ExportNode>? Nodes { get; set; }
00331     }
00332
00333     private class ExportNode
00334     {
00335         public string? Type { get; set; }
00336         public string? ElementType { get; set; }
00337         public List<string>? ComboBoxValues { get; set; }
00338         public List<ExportNode>? Nodes1 { get; set; }
00339         public List<ExportNode>? Nodes2 { get; set; }
00340     }
00341 }
00342 }
```

7.65 ladder_diagram_app/Services/MonitorServices/MonitorDataService.cs File Reference

Classes

- class ladder_diagram_app.Services.MonitorServices.MonitorDataService

  Processes and displays monitor data received from a device in the main window.
- class ladder_diagram_app.Services.MonitorServices.MonitorDataService.MonitorVariable

  Represents a variable in the monitor data with properties for various variable types.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.MonitorServices

7.66 MonitorDataService.cs

Go to the documentation of this file.

```
00001 using System.Text.Json;
00002 using System.Diagnostics;
00003
00004 namespace ladder_diagram_app.Services.MonitorServices
00005 {
```

```
00009   public class MonitorDataService
00010   {
00011       private readonly MainWindow _mainWindow;
00012
00017       public MonitorDataService(MainWindow mainWindow)
00018       {
00019           _mainWindow = mainWindow;
00020       }
00021
00026       public void OnMonitorDataReceived(string monitorData)
00027       {
00028           _mainWindow.Dispatcher.InvokeAsync(() =>
00029           {
00030               try
00031               {
00032                   var variables = JsonSerializer.Deserialize<List<MonitorVariable>>(monitorData);
00033                   if (variables == null)
00034                   {
00035                       _mainWindow.MonitorTextBlock.Text = "No data received";
00036                       return;
00037                   }
00038                   var formattedText = string.Join("\n", variables.Select(v => v.ToString()));
00039                   _mainWindow.MonitorTextBlock.Text = formattedText;
00040               }
00041               catch (Exception ex)
00042               {
00043                   Debug.WriteLine($"Error processing monitor data: {ex.Message}");
00044               }
00045           });
00046       }
00047
00051       private class MonitorVariable
00052       {
00053           public string? Type { get; set; }
00054           public string? Name { get; set; }
00055           public string? Pin { get; set; }
00056           public object? Value { get; set; }
00057           // ADC Sensor
00058           public string? SensorType { get; set; }
00059           public string? PD_SCK { get; set; }
00060           public string? DOUT { get; set; }
00061           public double? MapLow { get; set; }
00062           public double? MapHigh { get; set; }
00063           public double? Gain { get; set; }
00064           public string? SamplingRate { get; set; }
00065           // Counter
00066           public double? PV { get; set; }
00067           public double? CV { get; set; }
00068           public bool? CU { get; set; }
00069           public bool? CD { get; set; }
00070           public bool? QU { get; set; }
00071           public bool? QD { get; set; }
00072           // Timer
00073           public double? PT { get; set; }
00074           public double? ET { get; set; }
00075           public bool? IN { get; set; }
00076           public bool? Q { get; set; }
00077
00081           public override string ToString()
00082           {
00083               var parts = new List<string> { $"Type={Type}", $"Name={Name}" };
00084               if (!string.IsNullOrEmpty(Pin)) parts.Add($"Pin={Pin}");
00085               if (Value != null) parts.Add($"Value={Value}");
00086               // ADC Sensor
00087               if (!string.IsNullOrEmpty(SensorType)) parts.Add($"Sensor Type={SensorType}");
00088               if (!string.IsNullOrEmpty(PD_SCK)) parts.Add($"PD_SCK={PD_SCK}");
00089               if (!string.IsNullOrEmpty(DOUT)) parts.Add($"DOUT={DOUT}");
00090               if (MapLow.HasValue) parts.Add($"Map Low={MapLow.Value}");
00091               if (MapHigh.HasValue) parts.Add($"Map High={MapHigh.Value}");
00092               if (Gain.HasValue) parts.Add($"Gain={Gain.Value}");
00093               if (!string.IsNullOrEmpty(SamplingRate)) parts.Add($"Sampling Rate={SamplingRate}");
00094               // Counter
00095               if (PV.HasValue) parts.Add($"PV={PV.Value}");
00096               if (CV.HasValue) parts.Add($"CV={CV.Value}");
00097               if (CU.HasValue) parts.Add($"CU={CU.Value}");
00098               if (CD.HasValue) parts.Add($"CD={CD.Value}");
00099               if (QU.HasValue) parts.Add($"QU={QU.Value}");
00100              if (QD.HasValue) parts.Add($"QD={QD.Value}");
00101              // Timer
00102              if (PT.HasValue) parts.Add($"PT={PT.Value}");
00103              if (ET.HasValue) parts.Add($"ET={ET.Value}");
00104              if (IN.HasValue) parts.Add($"IN={IN.Value}");
00105              if (Q.HasValue) parts.Add($"Q={Q.Value}");
00106              return string.Join(", ", parts);
00107          }
00108      }
00109  }
```

00110 }

7.67 ladder_diagram_app/Services/MonitorServices/OneWireDataService.cs File Reference

Classes

- class ladder_diagram_app.Services.MonitorServices.OneWireDataService

  Manages one-wire sensor data processing and UI updates for the main window.
- class ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensorViewModel

  Represents a view model for one-wire sensors, used for UI display.
- class ladder_diagram_app.Services.MonitorServices.OneWireDataService.OneWireSensor

  Represents a one-wire sensor with address and type information.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Services
- namespace ladder_diagram_app.Services.MonitorServices

7.68 OneWireDataService.cs

Go to the documentation of this file.

```
00001 using System.Text.Json;
00002 using System.Windows;
00003 using System.Windows.Controls;
00004 using System.Diagnostics;
00005 using ladder_diagram_app.Models.DeviceElement;
00006 using ladder_diagram_app.Views;
00007
00008 namespace ladder_diagram_app.Services.MonitorServices
00009 {
00013     public class OneWireDataService
00014     {
00015         private readonly MainWindow _mainWindow;
00016         private readonly Device _device;
00017         private string? _lastOneWireMessage;
00018
00024         public OneWireDataService(MainWindow mainWindow, Device device)
00025         {
00026             _mainWindow = mainWindow;
00027             _device = device;
00028             _lastOneWireMessage = null;
00029         }
00030
00034         public void DeleteLastOneWireMessage()
00035         {
00036             _lastOneWireMessage = null;
00037         }
00038
00043         public void OnOneWireDataReceived(string oneWireData)
00044         {
00045             if (_mainWindow.Dispatcher.HasShutdownStarted || _mainWindow.Dispatcher.HasShutdownFinished)
00046                 return;
00047
00048             _mainWindow.Dispatcher.InvokeAsync(() =>
00049             {
00050                 ProcessOneWireMessage(oneWireData);
00051             });
00052         }
00053
00058         private void ProcessOneWireMessage(string message)
00059         {
00060             try
```

```
00061         {
00062             if (_lastOneWireMessage != message)
00063             {
00064                 _lastOneWireMessage = message;
00065 
00066                 var jsonDoc = JsonDocument.Parse(_lastOneWireMessage);
00067                 var root = jsonDoc.RootElement;
00068 
00069                 RefreshOneWireSensors(root);
00070             }
00071         }
00072         catch (Exception ex)
00073         {
00074             Debug.WriteLine($"Error processing one wire message: {ex.Message}");
00075         }
00076     }
00077 
00082     private void RefreshOneWireSensors(JsonElement? root = null)
00083     {
00084         var sensorViewModels = new List<OneWireSensorViewModel>();
00085 
00086         // Populate sensors from device configuration
00087         for (int i = 0; i < _device.one_wire_inputs.Count; i++)
00088         {
00089             int pin = _device.one_wire_inputs[i];
00090             var names = _device.one_wire_inputs_names[i];
00091             var types = _device.one_wire_inputs_devices_types[i];
00092             var addresses = _device.one_wire_inputs_devices_addresses[i];
00093 
00094             for (int j = 0; j < addresses.Count; j++)
00095             {
00096                 sensorViewModels.Add(new OneWireSensorViewModel
00097                 {
00098                     Pin = pin,
00099                     Address = addresses[j],
00100                     Type = types[j],
00101                     SensorName = names[j],
00102                     IsInDevice = true,
00103                     IsFromMqtt = false
00104                 });
00105             }
00106         }
00107 
00108         // Merge with MQTT data, if available
00109         if (root.HasValue && root.Value.TryGetProperty("pins", out var pinsArray))
00110         {
00111             foreach (var pinElement in pinsArray.EnumerateArray())
00112             {
00113                 if (pinElement.TryGetProperty("pin", out var pinProp) && pinElement.TryGetProperty("addresses", out
    var addressesProp))
00114                 {
00115                     int pinNumber = pinProp.GetInt32();
00116 
00117                     foreach (var address in addressesProp.EnumerateArray())
00118                     {
00119                         string? addressValue = address.GetString();
00120                         if (addressValue != null)
00121                         {
00122                             var sensor = new OneWireSensor(addressValue);
00123                             var existingSensor = sensorViewModels.FirstOrDefault(s => s.Pin == pinNumber && s.Address
    == sensor.Address);
00124                             if (existingSensor != null)
00125                             {
00126                                 existingSensor.IsFromMqtt = true;
00127                             }
00128                             else
00129                             {
00130                                 sensorViewModels.Add(new OneWireSensorViewModel
00131                                 {
00132                                     Pin = pinNumber,
00133                                     Address = sensor.Address,
00134                                     Type = sensor.Type,
00135                                     SensorName = "",
00136                                     IsInDevice = false,
00137                                     IsFromMqtt = true
00138                                 });
00139                             }
00140                         }
00141                     }
00142                 }
00143             }
00144         }
00145 
00146         _mainWindow.OneWireItemsControl.ItemsSource = sensorViewModels;
00147     }
00148 
00154     public void ActionButton_Click(object sender, RoutedEventArgs e)
```

```
00155       {
00156           Debug.WriteLine(_device.DeviceInfo());
00157
00158           var button = (Button)sender;
00159           var sensor = (OneWireSensorViewModel)button.Tag;
00160
00161           int pinIndex = _device.one_wire_inputs.IndexOf(sensor.Pin);
00162
00163           if (sensor.IsNotInDeviceAndFromMqtt)
00164           {
00165               // Add new sensor
00166               if (string.IsNullOrWhiteSpace(sensor.SensorName))
00167               {
00168                   new NotificationWindow("Sensor name cannot be empty!", _mainWindow).Show();
00169                   return;
00170               }
00171
00172               if (string.IsNullOrWhiteSpace(sensor.Type) || string.IsNullOrWhiteSpace(sensor.Address))
00173               {
00174                   new NotificationWindow("Sensor Type / Address cannot be empty!", _mainWindow).Show();
00175                   return;
00176               }
00177
00178               bool nameExists = _device.one_wire_inputs_names.SelectMany(list => list).Any(name =>
    name.Equals(sensor.SensorName, StringComparison.OrdinalIgnoreCase));
00179
00180               if (nameExists)
00181               {
00182                 new NotificationWindow($"Sensor with name '{sensor.SensorName}' already exists!", _mainWindow).Show();
00183                   return;
00184               }
00185
00186               _device.one_wire_inputs_names[pinIndex].Add(sensor.SensorName);
00187               _device.one_wire_inputs_devices_types[pinIndex].Add(sensor.Type);
00188               _device.one_wire_inputs_devices_addresses[pinIndex].Add(sensor.Address);
00189
00190               new NotificationWindow($"Sensor '{sensor.SensorName}' added successfully!", _mainWindow).Show();
00191
00192               _mainWindow._devicePinManager.OneWireInputOptions.Add(sensor.SensorName);
00193
00194               _mainWindow.Dispatcher.Invoke(() =>
00195               {
00196                   if (!string.IsNullOrEmpty(_lastOneWireMessage))
00197                   {
00198                       try
00199                       {
00200                           var jsonDoc = JsonDocument.Parse(_lastOneWireMessage);
00201                           RefreshOneWireSensors(jsonDoc.RootElement);
00202                       }
00203                       catch
00204                       {
00205                           RefreshOneWireSensors();
00206                       }
00207                   }
00208                   else
00209                   {
00210                       RefreshOneWireSensors();
00211                   }
00212               });
00213           }
00214           else if (sensor.IsInDevice && pinIndex != -1)
00215           {
00216               // Remove existing sensor
00217               int sensorIndex = sensor.Address != null ?
    _device.one_wire_inputs_devices_addresses[pinIndex].IndexOf(sensor.Address) : -1;
00218
00219               if (sensorIndex >= 0)
00220               {
00221                   string removedName = _device.one_wire_inputs_names[pinIndex][sensorIndex];
00222                   _device.one_wire_inputs_names[pinIndex].RemoveAt(sensorIndex);
00223                   _device.one_wire_inputs_devices_types[pinIndex].RemoveAt(sensorIndex);
00224                   _device.one_wire_inputs_devices_addresses[pinIndex].RemoveAt(sensorIndex);
00225
00226                   new NotificationWindow($"Sensor '{removedName}' removed successfully from pin {sensor.Pin}!",
    _mainWindow).Show();
00227
00228                   if (sensor.SensorName != null)
00229                   {
00230                       _mainWindow._devicePinManager.OneWireInputOptions.Remove(sensor.SensorName);
00231                   }
00232
00233                   _mainWindow.Dispatcher.Invoke(() =>
00234                   {
00235                       if (!string.IsNullOrEmpty(_lastOneWireMessage))
00236                       {
00237                           try
00238                           {
```

```
00239                           var jsonDoc = JsonDocument.Parse(_lastOneWireMessage);
00240                           RefreshOneWireSensors(jsonDoc.RootElement);
00241                       }
00242                       catch
00243                       {
00244                           RefreshOneWireSensors();
00245                       }
00246                   }
00247                   else
00248                   {
00249                       RefreshOneWireSensors();
00250                   }
00251               });
00252           }
00253           else
00254           {
00255               new NotificationWindow($"Sensor with address '{sensor.Address}' not found on pin {sensor.Pin}!",
    _mainWindow).Show();
00256           }
00257       }
00258       else
00259       {
00260           new NotificationWindow($"Pin {sensor.Pin} not found in device configuration!", _mainWindow).Show();
00261       }
00262   }
00263
00267   private class OneWireSensorViewModel
00268   {
00269       public int Pin { get; set; }
00270       public string? Address { get; set; }
00271       public string? Type { get; set; }
00272       public string? SensorName { get; set; }
00273       public bool IsInDevice { get; set; }
00274       public bool IsFromMqtt { get; set; }
00275       public bool IsInDeviceAndFromMqtt => IsInDevice && IsFromMqtt;
00276       public bool IsInDeviceAndNotFromMqtt => IsInDevice && !IsFromMqtt;
00277       public bool IsNotInDeviceAndFromMqtt => !IsInDevice && IsFromMqtt;
00278   }
00279
00283   private class OneWireSensor
00284   {
00285       public string Address { get; set; }
00286       public string Type { get; set; }
00287
00288       public OneWireSensor(string address)
00289       {
00290           Address = address;
00291           Type = GetSensorType(address);
00292       }
00293
00299       private string GetSensorType(string address)
00300       {
00301           if (string.IsNullOrEmpty(address)) return "Unknown";
00302
00303           string familyCode = address.Length >= 2 ? address.Substring(address.Length - 2, 2) : "00";
00304
00305           switch (familyCode)
00306           {
00307               case "10": return "DS18S20/DS1820 (Temperature Sensor)";
00308               case "22": return "DS1822 (Temperature Sensor)";
00309               case "28": return "DS18B20 (Temperature Sensor)";
00310               case "3B": return "MAX31850 (Temperature Sensor)";
00311               case "26": return "DS2438 (Smart Battery Monitor)";
00312               case "1D": return "DS2423 (4k RAM with Counter)";
00313               case "29": return "DS2408 (8-Channel Addressable Switch)";
00314               case "12": return "DS2406/DS2407 (Dual Addressable Switch)";
00315               case "20": return "DS2450 (4-Channel ADC)";
00316               case "21": return "DS1921 (Thermochron)";
00317               case "2D": return "DS2431 (1k EEPROM)";
00318               case "01": return "DS1990A (Serial Number)";
00319               case "04": return "DS2404 (RAM/Time)";
00320               case "14": return "DS1971 (256-bit EEPROM)";
00321               case "1F": return "DS2409 (MicroLAN Coupler)";
00322               default: return $"Unknown (Family Code: {familyCode})";
00323           }
00324       }
00325   }
00326 }
00327 }
```

7.69 ladder_diagram_app/UserControls/TimePicker.xaml File Reference

7.70 TimePicker.xaml

Go to the documentation of this file.

```
00001 <UserControl x:Class="ladder_diagram_app.UserControls.TimePicker"
00002             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
00003             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
00004             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
00005             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
00006             xmlns:local="clr-namespace:ladder_diagram_app.UserControls"
00007             mc:Ignorable="d"
00008             d:DesignHeight="25" d:DesignWidth="80">
00009     <Grid>
00010         <TextBlock x:Name="TimeDisplay"
00011                    Text="00:00:00"
00012                    MouseLeftButtonUp="TimeDisplay_MouseLeftButtonUp"
00013                    HorizontalAlignment="Center"/>
00014         <Popup x:Name="ConfigPopup"
00015                StaysOpen="True"
00016                Placement="Bottom"
00017                PlacementTarget="{Binding ElementName=TimeDisplay}"
00018                HorizontalOffset="0">
00019             <Border Background="White" BorderThickness="1" BorderBrush="Gray" Padding="5,5,5,0">
00020                 <StackPanel>
00021                     <StackPanel Orientation="Horizontal">
00022                         <ComboBox x:Name="HoursComboBox"
00023                                   Width="45"
00024                                   MaxDropDownHeight="100"
00025                                   SelectionChanged="ComboBox_SelectionChanged"/>
00026                         <TextBlock Text=":" Margin="5,0"/>
00027                         <ComboBox x:Name="MinutesComboBox"
00028                                   Width="45"
00029                                   MaxDropDownHeight="100"
00030                                   SelectionChanged="ComboBox_SelectionChanged"/>
00031                         <TextBlock Text=":" Margin="5,0"/>
00032                         <ComboBox x:Name="SecondsComboBox"
00033                                   Width="45"
00034                                   MaxDropDownHeight="100"
00035                                   SelectionChanged="ComboBox_SelectionChanged"/>
00036                     </StackPanel>
00037                     <Button Content="Apply"
00038                             Margin="10"
00039                             Height="25"
00040                             HorizontalAlignment="Stretch"
00041                             Click="ApplyButton_Click"/>
00042                 </StackPanel>
00043             </Border>
00044         </Popup>
00045     </Grid>
00046 </UserControl>
```

7.71 ladder_diagram_app/UserControls/TimePicker.xaml.cs File Reference

Classes

- class ladder_diagram_app.UserControls.TimePicker

  A user control for selecting and displaying time in a HH:mm:ss format.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.UserControls

7.72 TimePicker.xaml.cs

Go to the documentation of this file.

```
00001 using System;
00002 using System.Windows;
00003 using System.Windows.Controls;
00004
00005 namespace ladder_diagram_app.UserControls
00006 {
00010     public partial class TimePicker : UserControl
00011     {
00015         public static readonly DependencyProperty SelectedTimeProperty =
00016             DependencyProperty.Register(
00017                 nameof(SelectedTime),
00018                 typeof(double),
00019                 typeof(TimePicker),
00020                 new FrameworkPropertyMetadata(
00021                     0.0,
00022                     FrameworkPropertyMetadataOptions.BindsTwoWayByDefault,
00023                     OnSelectedTimeChanged));
00024
00028         public double SelectedTime
00029         {
00030             get => (double)GetValue(SelectedTimeProperty);
00031             set => SetValue(SelectedTimeProperty, value);
00032         }
00033
00037         public TimePicker()
00038         {
00039             InitializeComponent();
00040             InitializeComboBoxes();
00041         }
00042
00046         private void InitializeComboBoxes()
00047         {
00048             for (int i = 0; i <= 23; i++)
00049                 HoursComboBox.Items.Add(i.ToString("D2"));
00050             for (int i = 0; i <= 59; i++)
00051             {
00052                 MinutesComboBox.Items.Add(i.ToString("D2"));
00053                 SecondsComboBox.Items.Add(i.ToString("D2"));
00054             }
00055
00056             UpdateDisplayFromSelectedTime();
00057         }
00058
00064         private static void OnSelectedTimeChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)
00065         {
00066             var picker = (TimePicker)d;
00067             picker.UpdateDisplayFromSelectedTime();
00068         }
00069
00073         private void UpdateDisplayFromSelectedTime()
00074         {
00075             int hours = (int)(SelectedTime / 10000);
00076             int minutes = (int)((SelectedTime % 10000) / 100);
00077             int seconds = (int)(SelectedTime % 100);
00078
00079             hours = Math.Clamp(hours, 0, 23);
00080             minutes = Math.Clamp(minutes, 0, 59);
00081             seconds = Math.Clamp(seconds, 0, 59);
00082
00083             TimeDisplay.Text = $"{hours:D2}:{minutes:D2}:{seconds:D2}";
00084             HoursComboBox.SelectedItem = hours.ToString("D2");
00085             MinutesComboBox.SelectedItem = minutes.ToString("D2");
00086             SecondsComboBox.SelectedItem = seconds.ToString("D2");
00087         }
00088
00094         private void TimeDisplay_MouseLeftButtonUp(object sender, System.Windows.Input.MouseButtonEventArgs e)
00095         {
00096             ConfigPopup.IsOpen = true;
00097             HoursComboBox.Focus();
00098         }
00099
00105         private void ComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
00106         {
00107             UpdateTimePreview();
00108         }
00109
00115         private void ApplyButton_Click(object sender, RoutedEventArgs e)
00116         {
00117             UpdateTimePreview();
00118             ConfigPopup.IsOpen = false;
00119         }
00120
```

```
00124         private void UpdateTimePreview()
00125         {
00126             int hours = HoursComboBox.SelectedItem != null ? int.Parse(HoursComboBox.SelectedItem.ToString()) : 0;
00127             int minutes = MinutesComboBox.SelectedItem != null ? int.Parse(MinutesComboBox.SelectedItem.ToString()) : 0;
00128             int seconds = SecondsComboBox.SelectedItem != null ? int.Parse(SecondsComboBox.SelectedItem.ToString()) : 0;
00129 
00130             hours = Math.Clamp(hours, 0, 23);
00131             minutes = Math.Clamp(minutes, 0, 59);
00132             seconds = Math.Clamp(seconds, 0, 59);
00133 
00134             SelectedTime = double.Parse($"{hours:D2}{minutes:D2}{seconds:D2}.0");
00135             TimeDisplay.Text = $"{hours:D2}:{minutes:D2}:{seconds:D2}";
00136         }
00137     }
00138 }
```

7.73 ladder_diagram_app/Views/AddParentsWindow.xaml File Reference

7.74 AddParentsWindow.xaml

Go to the documentation of this file.

```
00001 <Window x:Class="ladder_diagram_app.Views.AddParentsWindow"
00002         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
00003         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
00004         WindowStyle="None"
00005         AllowsTransparency="True"
00006         Background="Transparent"
00007         Topmost="True"
00008         ShowInTaskbar="False"
00009         ResizeMode="NoResize"
00010         SizeToContent="WidthAndHeight"
00011         WindowStartupLocation="Manual">
00012 
00013     <Window.Resources>
00014         <!-- Stil za dugmad -->
00015         <Style TargetType="Button">
00016             <Setter Property="Background" Value="#FF616161"/>
00017             <Setter Property="Foreground" Value="#FFFFFFFF"/>
00018             <Setter Property="FontFamily" Value="Segoe UI"/>
00019             <Setter Property="FontSize" Value="14"/>
00020             <Setter Property="FontWeight" Value="SemiBold"/>
00021             <Setter Property="BorderThickness" Value="0"/>
00022             <Setter Property="Padding" Value="10,5"/>
00023             <Setter Property="Width" Value="90"/>
00024             <Setter Property="Height" Value="35"/>
00025             <Setter Property="Margin" Value="10,0,10,0"/>
00026             <Setter Property="Template">
00027                 <Setter.Value>
00028                     <ControlTemplate TargetType="Button">
00029                         <Border Background="{TemplateBinding Background}"
00030                                 CornerRadius="8"
00031                                 BorderThickness="0">
00032                             <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
00033                         </Border>
00034                         <ControlTemplate.Triggers>
00035                             <Trigger Property="IsMouseOver" Value="True">
00036                                 <Setter Property="Background" Value="#FF757575"/>
00037                             </Trigger>
00038                             <Trigger Property="IsPressed" Value="True">
00039                                 <Setter Property="Background" Value="#FF424242"/>
00040                             </Trigger>
00041                         </ControlTemplate.Triggers>
00042                     </ControlTemplate>
00043                 </Setter.Value>
00044             </Setter>
00045         </Style>
00046         <!-- Stil za TextBox -->
00047         <Style TargetType="TextBox">
00048             <Setter Property="Background" Value="#FF424242"/>
00049             <Setter Property="Foreground" Value="#FFE0E0E0"/>
00050             <Setter Property="FontFamily" Value="Segoe UI"/>
00051             <Setter Property="FontSize" Value="14"/>
00052             <Setter Property="Padding" Value="8"/>
00053             <Setter Property="Margin" Value="10,0,10,0"/>
00054             <Setter Property="Width" Value="200"/>
00055             <Setter Property="BorderThickness" Value="1"/>
```

```
00056           <Setter Property="BorderBrush" Value="#FF616161"/>
00057           <Setter Property="Template">
00058               <Setter.Value>
00059                   <ControlTemplate TargetType="TextBox">
00060                       <Border x:Name="Border" Background="{TemplateBinding Background}"
00061                               BorderBrush="{TemplateBinding BorderBrush}"
00062                               BorderThickness="{TemplateBinding BorderThickness}"
00063                               CornerRadius="4">
00064                           <ScrollViewer x:Name="PART_ContentHost"/>
00065                       </Border>
00066                       <ControlTemplate.Triggers>
00067                           <Trigger Property="IsFocused" Value="True">
00068                               <Setter TargetName="Border" Property="BorderBrush" Value="#FF757575"/>
00069                           </Trigger>
00070                       </ControlTemplate.Triggers>
00071                   </ControlTemplate>
00072               </Setter.Value>
00073           </Setter>
00074       </Style>
00075       <!-- Stil za ListBox -->
00076       <Style TargetType="ListBox">
00077           <Setter Property="Background" Value="#FF424242"/>
00078           <Setter Property="Foreground" Value="#FFE0E0E0"/>
00079           <Setter Property="BorderThickness" Value="1"/>
00080           <Setter Property="BorderBrush" Value="#FF616161"/>
00081           <Setter Property="Margin" Value="10,0,10,10"/>
00082           <Setter Property="MaxHeight" Value="200"/>
00083       </Style>
00084       <!-- Stil za ListBoxItem -->
00085       <Style TargetType="ListBoxItem">
00086           <Setter Property="Background" Value="Transparent"/>
00087           <Setter Property="Foreground" Value="#FFE0E0E0"/>
00088           <Setter Property="FontFamily" Value="Segoe UI"/>
00089           <Setter Property="FontSize" Value="14"/>
00090           <Setter Property="Padding" Value="5"/>
00091           <Setter Property="Template">
00092               <Setter.Value>
00093                   <ControlTemplate TargetType="ListBoxItem">
00094                       <Border Background="{TemplateBinding Background}"
00095                               BorderThickness="0"
00096                               CornerRadius="4">
00097                           <ContentPresenter HorizontalAlignment="Left" VerticalAlignment="Center"/>
00098                       </Border>
00099                       <ControlTemplate.Triggers>
00100                           <Trigger Property="IsSelected" Value="True">
00101                               <Setter Property="Background" Value="#FF616161"/>
00102                           </Trigger>
00103                           <Trigger Property="IsMouseOver" Value="True">
00104                               <Setter Property="Background" Value="#FF757575"/>
00105                           </Trigger>
00106                       </ControlTemplate.Triggers>
00107                   </ControlTemplate>
00108               </Setter.Value>
00109           </Setter>
00110       </Style>
00111       <!-- Stil za Delete dugme u ListBox -->
00112       <Style x:Key="DeleteButtonStyle" TargetType="Button">
00113           <Setter Property="Background" Value="#FFB00020"/>
00114           <Setter Property="Foreground" Value="#FFFFFFFF"/>
00115           <Setter Property="FontFamily" Value="Segoe UI"/>
00116           <Setter Property="FontSize" Value="12"/>
00117           <Setter Property="Padding" Value="5"/>
00118           <Setter Property="Width" Value="30"/>
00119           <Setter Property="Height" Value="25"/>
00120           <Setter Property="Margin" Value="5"/>
00121           <Setter Property="Template">
00122               <Setter.Value>
00123                   <ControlTemplate TargetType="Button">
00124                       <Border Background="{TemplateBinding Background}"
00125                               CornerRadius="4"
00126                               BorderThickness="0">
00127                           <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
00128                       </Border>
00129                       <ControlTemplate.Triggers>
00130                           <Trigger Property="IsMouseOver" Value="True">
00131                               <Setter Property="Background" Value="#FFD32F2F"/>
00132                           </Trigger>
00133                           <Trigger Property="IsPressed" Value="True">
00134                               <Setter Property="Background" Value="#FFC62828"/>
00135                           </Trigger>
00136                       </ControlTemplate.Triggers>
00137                   </ControlTemplate>
00138               </Setter.Value>
00139           </Setter>
00140       </Style>
00141   </Window.Resources>
00142
```

```
00143     <Border CornerRadius="12" Background="#D0323232" Padding="15">
00144         <StackPanel VerticalAlignment="Center" Margin="10">
00145             <TextBlock Text="Manage Parent Devices" Foreground="#FFE0E0E0" FontSize="16"
00146                        TextWrapping="Wrap" HorizontalAlignment="Center" TextAlignment="Center"
00147                        Margin="0,0,0,15" FontFamily="Segoe UI" FontWeight="Regular" MaxWidth="400"/>
00148
00149             <!-- Lista postojećih parent_devices -->
00150             <ListBox x:Name="ParentDevicesListBox" ItemsSource="{Binding ParentDevices}">
00151                 <ListBox.ItemTemplate>
00152                     <DataTemplate>
00153                         <StackPanel Orientation="Horizontal" VerticalAlignment="Center">
00154                             <TextBlock Text="{Binding}" VerticalAlignment="Center" Margin="5,0,10,0"/>
00155                             <Button Content="X" Style="{StaticResource DeleteButtonStyle}"
00156                                     Click="DeleteParentDevice_Click" Tag="{Binding}"/>
00157                         </StackPanel>
00158                     </DataTemplate>
00159                 </ListBox.ItemTemplate>
00160             </ListBox>
00161
00162             <!-- Polje za dodavanje novog elementa -->
00163             <StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="0,10,0,10">
00164                 <TextBox x:Name="NewParentTextBox" Width="200"/>
00165                 <Button Content="Add" Click="AddParentDevice_Click"/>
00166             </StackPanel>
00167
00168             <!-- Dugme za čuvanje -->
00169             <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
00170                 <Button Content="Save" Click="Save_Click"/>
00171                 <Button Content="Cancel" Click="Cancel_Click"/>
00172             </StackPanel>
00173         </StackPanel>
00174     </Border>
00175 </Window>
```

7.75 ladder_diagram_app/Views/AddParentsWindow.xaml.cs File Reference

Classes

- class ladder_diagram_app.Views.AddParentsWindow

  Window for adding and managing parent devices, centered relative to the owner window.
- struct ladder_diagram_app.Views.AddParentsWindow.RECT

  Represents a rectangle with left, top, right, and bottom coordinates.
- struct ladder_diagram_app.Views.AddParentsWindow.MONITORINFO

  Contains information about a monitor's size and work area.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Views

7.76 AddParentsWindow.xaml.cs

Go to the documentation of this file.

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Collections.ObjectModel;
00004 using System.Runtime.InteropServices;
00005 using System.Windows;
00006 using System.Windows.Controls;
00007 using System.Windows.Input;
00008 using System.Windows.Interop;
00009
00010 namespace ladder_diagram_app.Views
00011 {
```

```
00015   public partial class AddParentsWindow : Window
00016   {
00017       // Native Windows API methods for monitor information
00018       [DllImport("user32.dll")]
00019       private static extern IntPtr MonitorFromWindow(IntPtr hwnd, uint dwFlags);
00020
00021       [DllImport("user32.dll")]
00022       private static extern bool GetMonitorInfo(IntPtr hMonitor, ref MONITORINFO lpmi);
00023
00024       private const uint MONITOR_DEFAULTTONEAREST = 2;
00025
00029       [StructLayout(LayoutKind.Sequential)]
00030       private struct RECT
00031       {
00032           public int Left;
00033           public int Top;
00034           public int Right;
00035           public int Bottom;
00036       }
00037
00041       [StructLayout(LayoutKind.Sequential)]
00042       private struct MONITORINFO
00043       {
00044           public int cbSize;          // Size of the structure
00045           public RECT rcMonitor;      // Monitor rectangle
00046           public RECT rcWork;         // Working area rectangle
00047           public uint dwFlags;        // Monitor flags
00048       }
00049
00050       private readonly List<string> _parentDevices;
00054       public ObservableCollection<string> ParentDevices { get; set; }
00055
00061       public AddParentsWindow(List<string> parentDevices, Window owner)
00062       {
00063           InitializeComponent();
00064           _parentDevices = parentDevices ?? new List<string>();
00065           ParentDevices = new ObservableCollection<string>(_parentDevices);
00066           DataContext = this;
00067
00068           // Set window size constraints
00069           MinWidth = 200;
00070           MaxWidth = 600;
00071
00072           this.Owner = owner;
00073
00074           // Subscribe to owner window events for position and size updates
00075           owner.LocationChanged += Owner_PositionOrSizeChanged;
00076           owner.SizeChanged += Owner_PositionOrSizeChanged;
00077           owner.StateChanged += Owner_StateChanged;
00078
00079           // Unsubscribe from owner events when window closes
00080           Closed += (s, e) =>
00081           {
00082               owner.LocationChanged -= Owner_PositionOrSizeChanged;
00083               owner.SizeChanged -= Owner_PositionOrSizeChanged;
00084               owner.StateChanged -= Owner_StateChanged;
00085           };
00086
00087           // Position window and set focus on load
00088           Loaded += (s, e) =>
00089           {
00090               UpdatePosition();
00091               NewParentTextBox.Focus();
00092           };
00093
00094           // Handle keyboard input for adding devices and closing window
00095           KeyDown += (s, e) =>
00096           {
00097               if (e.Key == Key.Enter)
00098               {
00099                   string newParent = NewParentTextBox.Text.Trim();
00100                   if (!string.IsNullOrWhiteSpace(newParent) && !ParentDevices.Contains(newParent,
    StringComparer.OrdinalIgnoreCase))
00101                   {
00102                       ParentDevices.Add(newParent.ToUpper());
00103                       NewParentTextBox.Text = string.Empty;
00104                       NewParentTextBox.Focus();
00105                   }
00106                   else if (string.IsNullOrWhiteSpace(newParent))
00107                   {
00108                       new NotificationWindow("Parent device name cannot be empty!", this).Show();
00109                   }
00110                   else
00111                   {
00112                       new NotificationWindow($"Parent device '{newParent}' already exists!", this).Show();
00113                   }
00114               }
```

```
00115               else if (e.Key == Key.Escape)
00116               {
00117                   Close();
00118               }
00119           };
00120       }
00121
00127       private void AddParentDevice_Click(object sender, RoutedEventArgs e)
00128       {
00129           string newParent = NewParentTextBox.Text.Trim();
00130           if (!string.IsNullOrWhiteSpace(newParent) && !ParentDevices.Contains(newParent,
      StringComparer.OrdinalIgnoreCase))
00131           {
00132               ParentDevices.Add(newParent.ToUpper());
00133               NewParentTextBox.Text = string.Empty;
00134               NewParentTextBox.Focus();
00135           }
00136           else if (string.IsNullOrWhiteSpace(newParent))
00137           {
00138               new NotificationWindow("Parent device name cannot be empty!", this).Show();
00139           }
00140           else
00141           {
00142               new NotificationWindow($"Parent device '{newParent}' already exists!", this).Show();
00143           }
00144       }
00145
00151       private void DeleteParentDevice_Click(object sender, RoutedEventArgs e)
00152       {
00153           if (sender is Button button && button.Tag is string parent)
00154           {
00155               ParentDevices.Remove(parent);
00156           }
00157       }
00158
00164       private void Save_Click(object sender, RoutedEventArgs e)
00165       {
00166           _parentDevices.Clear();
00167           _parentDevices.AddRange(ParentDevices);
00168           DialogResult = true; // Indicate successful save
00169           Close();
00170       }
00171
00177       private void Cancel_Click(object sender, RoutedEventArgs e)
00178       {
00179           DialogResult = false; // Indicate cancellation
00180           Close();
00181       }
00182
00186       private void UpdatePosition()
00187       {
00188           if (Owner != null)
00189           {
00190               var hwnd = new WindowInteropHelper(Owner).Handle;
00191               var monitor = MonitorFromWindow(hwnd, MONITOR_DEFAULTTONEAREST);
00192
00193               if (monitor != IntPtr.Zero)
00194               {
00195                   var monitorInfo = new MONITORINFO();
00196                   monitorInfo.cbSize = Marshal.SizeOf(typeof(MONITORINFO));
00197                   GetMonitorInfo(monitor, ref monitorInfo);
00198
00199                   var source = PresentationSource.FromVisual(Owner);
00200                   if (source != null)
00201                   {
00202                       var dpiScale = source.CompositionTarget.TransformToDevice.M11;
00203
00204                       if (Owner.WindowState == WindowState.Maximized)
00205                       {
00206                           // Center on the monitor's work area for maximized owner
00207                           double screenWidth = (monitorInfo.rcWork.Right - monitorInfo.rcWork.Left) / dpiScale;
00208                           double screenHeight = (monitorInfo.rcWork.Bottom - monitorInfo.rcWork.Top) / dpiScale;
00209                           double screenLeft = monitorInfo.rcWork.Left / dpiScale;
00210                           double screenTop = monitorInfo.rcWork.Top / dpiScale;
00211
00212                           this.Left = screenLeft + (screenWidth - this.ActualWidth) / 2;
00213                           this.Top = screenTop + (screenHeight - this.ActualHeight) / 2;
00214                       }
00215                       else
00216                       {
00217                           // Center relative to the owner window
00218                           this.Left = Owner.Left + (Owner.ActualWidth - this.ActualWidth) / 2;
00219                           this.Top = Owner.Top + (Owner.ActualHeight - this.ActualHeight) / 2;
00220                       }
00221
00222                       // Adjust height if the dialog is taller than the owner
00223                       if (this.ActualHeight > Owner.ActualHeight)
```

```
00224                     {
00225                         this.Top = Owner.Top;
00226                         this.Height = Owner.ActualHeight;
00227                     }
00228                 }
00229             }
00230         }
00231     }
00232
00238     private void Owner_PositionOrSizeChanged(object sender, EventArgs e)
00239     {
00240         UpdatePosition();
00241     }
00242
00248     private void Owner_StateChanged(object sender, EventArgs e)
00249     {
00250         UpdatePosition();
00251     }
00252 }
00253 }
```

7.77 ladder_diagram_app/Views/BleDeviceSelectionWindow.cs File Reference

Classes

- class ladder_diagram_app.Views.BleDeviceSelectionWindow

  Represents a window for selecting a Bluetooth Low Energy (BLE) device from a list.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Views

7.78 BleDeviceSelectionWindow.cs

Go to the documentation of this file.

```
00001 using System;
00002 using System.Collections.ObjectModel;
00003 using System.Windows;
00004 using System.Windows.Controls;
00005 using System.Windows.Input;
00006 using Windows.Devices.Enumeration;
00007
00008 namespace ladder_diagram_app.Views
00009 {
00013     public class BleDeviceSelectionWindow : Window
00014     {
00018         public DeviceInformation? SelectedDevice { get; private set; }
00019
00023         private readonly ObservableCollection<DeviceInformation> _devices;
00024
00030         public BleDeviceSelectionWindow(ObservableCollection<DeviceInformation> devices, Window owner)
00031         {
00032             Owner = owner;
00033             _devices = devices;
00034             InitializeComponents();
00035         }
00036
00040         private void InitializeComponents()
00041         {
00042             // Set window properties
00043             Width = 300;
00044             Height = double.NaN;
00045             SizeToContent = SizeToContent.Height; // Dynamically adjust height based on content
00046             Title = "Select BLE Device";
00047             WindowStartupLocation = WindowStartupLocation.CenterOwner;
00048             ResizeMode = ResizeMode.NoResize;
```

```
00049
00050           // Create main layout panel
00051           var stackPanel = new StackPanel();
00052
00053           // Initialize ListBox for displaying BLE devices
00054           var listBox = new ListBox
00055           {
00056               Margin = new Thickness(5),
00057               DisplayMemberPath = "Name",
00058               MaxHeight = 300
00059           };
00060           listBox.ItemsSource = _devices;
00061
00062           // Auto-select first item and set focus when ListBox is loaded
00063           listBox.Loaded += (s, e) =>
00064           {
00065               if (listBox.Items.Count > 0)
00066               {
00067                   listBox.SelectedIndex = 0;
00068                   listBox.Focus();
00069               }
00070           };
00071
00072           // Handle keyboard navigation and selection
00073           listBox.KeyDown += (s, e) =>
00074           {
00075               if (e.Key == Key.Enter && listBox.SelectedItem != null)
00076               {
00077                   // Select device and close window on Enter key
00078                   SelectedDevice = listBox.SelectedItem as DeviceInformation;
00079                   DialogResult = true;
00080                   Close();
00081               }
00082               else if (e.Key == Key.Up)
00083               {
00084                   // Navigate up in the list
00085                   int newIndex = Math.Max(0, listBox.SelectedIndex - 1);
00086                   listBox.SelectedIndex = newIndex;
00087                   e.Handled = true;
00088               }
00089               else if (e.Key == Key.Down)
00090               {
00091                   // Navigate down in the list
00092                   int newIndex = Math.Min(listBox.Items.Count - 1, listBox.SelectedIndex + 1);
00093                   listBox.SelectedIndex = newIndex;
00094                   e.Handled = true;
00095               }
00096           };
00097
00098           // Create Select button
00099           var selectButton = new Button
00100           {
00101               Content = "Select",
00102               Margin = new Thickness(5),
00103               Width = 100,
00104               Height = 30
00105           };
00106           selectButton.Click += (s, e) =>
00107           {
00108               // Set selected device and close window
00109               SelectedDevice = listBox.SelectedItem as DeviceInformation;
00110               DialogResult = SelectedDevice != null;
00111               Close();
00112           };
00113
00114           // Create Cancel button
00115           var cancelButton = new Button
00116           {
00117               Content = "Cancel",
00118               Margin = new Thickness(5),
00119               Width = 100,
00120               Height = 30
00121           };
00122           cancelButton.Click += (s, e) =>
00123           {
00124               // Close window without selecting a device
00125               DialogResult = false;
00126               Close();
00127           };
00128
00129           // Create button panel for Select and Cancel buttons
00130           var buttonPanel = new StackPanel
00131           {
00132               Orientation = Orientation.Horizontal,
00133               HorizontalAlignment = HorizontalAlignment.Center,
00134               Height = 60
00135           };
```

```
00136             buttonPanel.Children.Add(selectButton);
00137             buttonPanel.Children.Add(cancelButton);
00138
00139             // Add components to main panel
00140             stackPanel.Children.Add(listBox);
00141             stackPanel.Children.Add(buttonPanel);
00142
00143             // Set window content
00144             Content = stackPanel;
00145         }
00146     }
00147 }
```

7.79 ladder_diagram_app/Views/NotificationWindow.xaml File Reference

7.80 NotificationWindow.xaml

Go to the documentation of this file.

```
00001 <Window x:Class="ladder_diagram_app.Views.NotificationWindow"
00002         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
00003         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
00004         WindowStyle="None"
00005         AllowsTransparency="True"
00006         Background="Transparent"
00007         Topmost="True"
00008         ShowInTaskbar="False"
00009         ResizeMode="NoResize"
00010         SizeToContent="WidthAndHeight"
00011         WindowStartupLocation="Manual">
00012
00013     <Window.Resources>
00014         <!-- Stil za dugmad -->
00015         <Style TargetType="Button">
00016             <Setter Property="Background" Value="#FF616161"/>
00017             <Setter Property="Foreground" Value="#FFFFFFFF"/>
00018             <Setter Property="FontFamily" Value="Segoe UI"/>
00019             <Setter Property="FontSize" Value="14"/>
00020             <Setter Property="FontWeight" Value="SemiBold"/>
00021             <Setter Property="BorderThickness" Value="0"/>
00022             <Setter Property="Padding" Value="10,5"/>
00023             <Setter Property="Width" Value="90"/>
00024             <Setter Property="Height" Value="35"/>
00025             <Setter Property="Margin" Value="10,0,10,0"/>
00026             <Setter Property="Template">
00027                 <Setter.Value>
00028                     <ControlTemplate TargetType="Button">
00029                         <Border Background="{TemplateBinding Background}"
00030                                 CornerRadius="8"
00031                                 BorderThickness="0">
00032                             <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
00033                         </Border>
00034                         <ControlTemplate.Triggers>
00035                             <Trigger Property="IsMouseOver" Value="True">
00036                                 <Setter Property="Background" Value="#FF757575"/>
00037                             </Trigger>
00038                             <Trigger Property="IsPressed" Value="True">
00039                                 <Setter Property="Background" Value="#FF424242"/>
00040                             </Trigger>
00041                         </ControlTemplate.Triggers>
00042                     </ControlTemplate>
00043                 </Setter.Value>
00044             </Setter>
00045         </Style>
00046         <!-- Stil za TextBox -->
00047         <Style TargetType="TextBox">
00048             <Setter Property="Background" Value="#FF424242"/>
00049             <Setter Property="Foreground" Value="#FFE0E0E0"/>
00050             <Setter Property="FontFamily" Value="Segoe UI"/>
00051             <Setter Property="FontSize" Value="14"/>
00052             <Setter Property="Padding" Value="8"/>
00053             <Setter Property="Margin" Value="10,0,0,0"/>
00054             <Setter Property="Width" Value="200"/>
00055             <Setter Property="BorderThickness" Value="1"/>
00056             <Setter Property="BorderBrush" Value="#FF616161"/>
00057             <Setter Property="Template">
00058                 <Setter.Value>
```

```
00059                 <ControlTemplate TargetType="TextBox">
00060                     <Border x:Name="Border" Background="{TemplateBinding Background}"
00061                             BorderBrush="{TemplateBinding BorderBrush}"
00062                             BorderThickness="{TemplateBinding BorderThickness}"
00063                             CornerRadius="4">
00064                         <ScrollViewer x:Name="PART_ContentHost"/>
00065                     </Border>
00066                     <ControlTemplate.Triggers>
00067                         <Trigger Property="IsFocused" Value="True">
00068                             <Setter TargetName="Border" Property="BorderBrush" Value="#FF757575"/>
00069                         </Trigger>
00070                     </ControlTemplate.Triggers>
00071                 </ControlTemplate>
00072             </Setter.Value>
00073         </Setter>
00074     </Style>
00075 </Window.Resources>
00076
00077 <Border CornerRadius="12" Background="#D0323232" Padding="15">
00078     <StackPanel VerticalAlignment="Center" Margin="10">
00079         <TextBlock x:Name="MessageText" Foreground="#FFE0E0E0" FontSize="16" TextWrapping="Wrap"
00080                    HorizontalAlignment="Center" TextAlignment="Center" Margin="0,0,0,15"
00081                    FontFamily="Segoe UI" FontWeight="Regular" MaxWidth="400"/>
00082         <!-- Polja za unos -->
00083         <StackPanel x:Name="InputPanel" Visibility="Collapsed">
00084             <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
00085                 <TextBlock x:Name="Input1Label" Foreground="#FFE0E0E0" FontSize="14"
00086                            VerticalAlignment="Center" TextAlignment="Right"/>
00087                 <TextBox x:Name="Input1TextBox" Visibility="Collapsed"/>
00088             </StackPanel>
00089             <StackPanel Orientation="Horizontal" Margin="0,0,0,10" Visibility="Collapsed">
00090                 <TextBlock x:Name="Input2Label" Foreground="#FFE0E0E0" FontSize="14"
00091                            VerticalAlignment="Center" TextAlignment="Right"/>
00092                 <TextBox x:Name="Input2TextBox" Visibility="Collapsed"/>
00093             </StackPanel>
00094             <StackPanel Orientation="Horizontal" Margin="0,0,0,10" Visibility="Collapsed">
00095                 <TextBlock x:Name="Input3Label" Foreground="#FFE0E0E0" FontSize="14"
00096                            VerticalAlignment="Center" TextAlignment="Right"/>
00097                 <TextBox x:Name="Input3TextBox" Visibility="Collapsed"/>
00098             </StackPanel>
00099         </StackPanel>
00100         <StackPanel x:Name="ButtonPanel" Orientation="Horizontal" HorizontalAlignment="Center"
    Visibility="Collapsed">
00101             <Button x:Name="YesButton" Content="YES"/>
00102             <Button x:Name="NoButton" Content="NO"/>
00103             <Button x:Name="OkButton" Content="CLOSE" Visibility="Collapsed"/>
00104             <Button x:Name="ConfirmButton" Content="CONFIRM" Visibility="Collapsed"/>
00105             <Button x:Name="CancelButton" Content="CANCEL" Visibility="Collapsed"/>
00106         </StackPanel>
00107     </StackPanel>
00108   </Border>
00109 </Window>
```

7.81 ladder_diagram_app/Views/NotificationWindow.xaml.cs File Reference

Classes

- class ladder_diagram_app.Views.NotificationWindow

  A customizable notification window that supports various button configurations and input fields.
- struct ladder_diagram_app.Views.NotificationWindow.RECT

  Represents a rectangle with left, top, right, and bottom coordinates.
- struct ladder_diagram_app.Views.NotificationWindow.MONITORINFO

  Contains information about a monitor's size and work area.

Namespaces

- namespace ladder_diagram_app
- namespace ladder_diagram_app.Views

Enumerations

- enum ladder_diagram_app.Views.NotificationButtons { ladder_diagram_app.Views.None , ladder_diagram_app.Views.YesNo , ladder_diagram_app.Views.Ok , ladder_diagram_app.Views.OneInput , ladder_diagram_app.Views.TwoInputs , ladder_diagram_app.Views.ThreeInputs }

  Defines the types of button configurations for the notification window.

7.82 NotificationWindow.xaml.cs

Go to the documentation of this file.

```
00001 using System;
00002 using System.Runtime.InteropServices;
00003 using System.Windows;
00004 using System.Windows.Controls;
00005 using System.Windows.Input;
00006 using System.Windows.Interop;
00007 using System.Windows.Threading;
00008
00009 namespace ladder_diagram_app.Views
00010 {
00014     public enum NotificationButtons
00015     {
00016         None,        // No buttons, auto-closes after a delay
00017         YesNo,       // Yes and No buttons
00018         Ok,          // Single OK button
00019         OneInput,    // One input field with Confirm and Cancel buttons
00020         TwoInputs,   // Two input fields with Confirm and Cancel buttons
00021         ThreeInputs  // Three input fields with Confirm and Cancel buttons
00022     }
00023
00027     public partial class NotificationWindow : Window
00028     {
00029         // Native Windows API methods for monitor information
00030         [DllImport("user32.dll")]
00031         private static extern IntPtr MonitorFromWindow(IntPtr hwnd, uint dwFlags);
00032
00033         [DllImport("user32.dll")]
00034         private static extern bool GetMonitorInfo(IntPtr hMonitor, ref MONITORINFO lpmi);
00035
00036         private const uint MONITOR_DEFAULTTONEAREST = 2;
00037
00041         [StructLayout(LayoutKind.Sequential)]
00042         private struct RECT
00043         {
00044             public int Left;
00045             public int Top;
00046             public int Right;
00047             public int Bottom;
00048         }
00049
00053         [StructLayout(LayoutKind.Sequential)]
00054         private struct MONITORINFO
00055         {
00056             public int cbSize;          // Size of the structure
00057             public RECT rcMonitor;      // Monitor rectangle
00058             public RECT rcWork;         // Working area rectangle
00059             public uint dwFlags;        // Monitor flags
00060         }
00061
00062         private bool? _result = null;          // Stores the dialog result (true/false for YesNo/Confirm/Cancel)
00063         private string[] _inputResults = null;  // Stores input field values for input-based dialogs
00064
00068         public bool? Result => _result;
00069
00073         public string[] InputResults => _inputResults;
00074
00082         public NotificationWindow(string poruka, Window owner, NotificationButtons buttons = NotificationButtons.None,
    string[] inputLabels = null)
00083         {
00084             InitializeComponent();
00085             MessageText.Text = poruka;
00086
00087             // Set minimum and maximum window dimensions
00088             MinWidth = 200;
00089             MaxWidth = 600;
00090
00091             this.Owner = owner;
```

```
00092
00093           // Subscribe to owner window events for position and size updates
00094           owner.LocationChanged += Owner_PositionOrSizeChanged;
00095           owner.SizeChanged += Owner_PositionOrSizeChanged;
00096           owner.StateChanged += Owner_StateChanged;
00097
00098           // Unsubscribe from events when window closes
00099           Closed += (s, e) =>
00100           {
00101               owner.LocationChanged -= Owner_PositionOrSizeChanged;
00102               owner.SizeChanged -= Owner_PositionOrSizeChanged;
00103               owner.StateChanged -= Owner_StateChanged;
00104           };
00105
00106           // Position and focus handling on window load
00107           Loaded += (s, e) =>
00108           {
00109               UpdatePosition();
00110               if (buttons == NotificationButtons.OneInput || buttons == NotificationButtons.TwoInputs || buttons ==
      NotificationButtons.ThreeInputs)
00111               {
00112                   Input1TextBox.Focus(); // Set focus to the first input field
00113               }
00114               else
00115               {
00116                   this.Focus(); // Set focus to the window for YesNo/Ok/None
00117               }
00118           };
00119
00120           // Handle keyboard input
00121           KeyDown += (s, e) =>
00122           {
00123               switch (buttons)
00124               {
00125                   case NotificationButtons.YesNo:
00126                       if (e.Key == Key.Enter)
00127                       {
00128                           _result = true; // Yes
00129                           Close();
00130                       }
00131                       else if (e.Key == Key.Escape)
00132                       {
00133                           _result = false; // No
00134                           Close();
00135                       }
00136                       break;
00137
00138                   case NotificationButtons.Ok:
00139                       if (e.Key == Key.Enter || e.Key == Key.Escape)
00140                       {
00141                           Close(); // Close on Enter or Escape
00142                       }
00143                       break;
00144
00145                   case NotificationButtons.OneInput:
00146                   case NotificationButtons.TwoInputs:
00147                   case NotificationButtons.ThreeInputs:
00148                       if (e.Key == Key.Enter)
00149                       {
00150                           if (AreInputsValid(buttons))
00151                           {
00152                               // Store input values based on button configuration
00153                               if (buttons == NotificationButtons.OneInput)
00154                                   _inputResults[0] = Input1TextBox.Text;
00155                               else if (buttons == NotificationButtons.TwoInputs)
00156                               {
00157                                   _inputResults[0] = Input1TextBox.Text;
00158                                   _inputResults[1] = Input2TextBox.Text;
00159                               }
00160                               else
00161                               {
00162                                   _inputResults[0] = Input1TextBox.Text;
00163                                   _inputResults[1] = Input2TextBox.Text;
00164                                   _inputResults[2] = Input3TextBox.Text;
00165                               }
00166                               _result = true; // Confirm
00167                               Close();
00168                           }
00169                       }
00170                       else if (e.Key == Key.Escape)
00171                       {
00172                           _result = false; // Cancel
00173                           Close();
00174                       }
00175                       break;
00176               }
00177           };
```

```
00178
00179         // Configure buttons and input fields based on button type
00180         switch (buttons)
00181         {
00182             case NotificationButtons.None:
00183                 Loaded += (s, e) =>
00184                 {
00185                     // Auto-close after 3 seconds
00186                     var timer = new DispatcherTimer { Interval = TimeSpan.FromSeconds(3) };
00187                     timer.Tick += (sender, args) => { timer.Stop(); Close(); };
00188                     timer.Start();
00189                 };
00190                 break;
00191
00192             case NotificationButtons.YesNo:
00193                 ButtonPanel.Visibility = Visibility.Visible;
00194                 YesButton.Visibility = Visibility.Visible;
00195                 NoButton.Visibility = Visibility.Visible;
00196                 OkButton.Visibility = Visibility.Collapsed;
00197                 ConfirmButton.Visibility = Visibility.Collapsed;
00198                 CancelButton.Visibility = Visibility.Collapsed;
00199                 YesButton.Click += (s, e) => { _result = true; Close(); }; // Yes button
00200                 NoButton.Click += (s, e) => { _result = false; Close(); }; // No button
00201                 break;
00202
00203             case NotificationButtons.Ok:
00204                 ButtonPanel.Visibility = Visibility.Visible;
00205                 YesButton.Visibility = Visibility.Collapsed;
00206                 NoButton.Visibility = Visibility.Collapsed;
00207                 OkButton.Visibility = Visibility.Visible;
00208                 ConfirmButton.Visibility = Visibility.Collapsed;
00209                 CancelButton.Visibility = Visibility.Collapsed;
00210                 OkButton.Click += (s, e) => { Close(); }; // OK button
00211                 break;
00212
00213             case NotificationButtons.OneInput:
00214                 InputPanel.Visibility = Visibility.Visible;
00215                 Input1Label.Text = inputLabels != null && inputLabels.Length > 0 ? inputLabels[0] : "Input 1";
00216                 Input1Label.Visibility = Visibility.Visible;
00217                 Input1TextBox.Visibility = Visibility.Visible;
00218                 ButtonPanel.Visibility = Visibility.Visible;
00219                 YesButton.Visibility = Visibility.Collapsed;
00220                 NoButton.Visibility = Visibility.Collapsed;
00221                 OkButton.Visibility = Visibility.Collapsed;
00222                 ConfirmButton.Visibility = Visibility.Visible;
00223                 CancelButton.Visibility = Visibility.Visible;
00224                 _inputResults = new string[1];
00225                 ConfirmButton.Click += (s, e) =>
00226                 {
00227                     if (AreInputsValid(buttons))
00228                     {
00229                         _inputResults[0] = Input1TextBox.Text;
00230                         _result = true; // Confirm
00231                         Close();
00232                     }
00233                 };
00234                 CancelButton.Click += (s, e) => { _result = false; Close(); }; // Cancel
00235                 AdjustLabelWidths();
00236                 break;
00237
00238             case NotificationButtons.TwoInputs:
00239                 InputPanel.Visibility = Visibility.Visible;
00240                 Input1Label.Text = inputLabels != null && inputLabels.Length > 0 ? inputLabels[0] : "Input 1";
00241                 Input1Label.Visibility = Visibility.Visible;
00242                 Input1TextBox.Visibility = Visibility.Visible;
00243                 Input2Label.Text = inputLabels != null && inputLabels.Length > 1 ? inputLabels[1] : "Input 2";
00244                 Input2Label.Visibility = Visibility.Visible;
00245                 Input2TextBox.Visibility = Visibility.Visible;
00246                 InputPanel.Children[1].Visibility = Visibility.Visible;
00247                 ButtonPanel.Visibility = Visibility.Visible;
00248                 YesButton.Visibility = Visibility.Collapsed;
00249                 NoButton.Visibility = Visibility.Collapsed;
00250                 OkButton.Visibility = Visibility.Collapsed;
00251                 ConfirmButton.Visibility = Visibility.Visible;
00252                 CancelButton.Visibility = Visibility.Visible;
00253                 _inputResults = new string[2];
00254                 ConfirmButton.Click += (s, e) =>
00255                 {
00256                     if (AreInputsValid(buttons))
00257                     {
00258                         _inputResults[0] = Input1TextBox.Text;
00259                         _inputResults[1] = Input2TextBox.Text;
00260                         _result = true; // Confirm
00261                         Close();
00262                     }
00263                 };
00264                 CancelButton.Click += (s, e) => { _result = false; Close(); }; // Cancel
```

```
00265                 AdjustLabelWidths();
00266                 break;
00267
00268             case NotificationButtons.ThreeInputs:
00269                 InputPanel.Visibility = Visibility.Visible;
00270                 Input1Label.Text = inputLabels != null && inputLabels.Length > 0 ? inputLabels[0] : "Input 1";
00271                 Input1Label.Visibility = Visibility.Visible;
00272                 Input1TextBox.Visibility = Visibility.Visible;
00273                 Input2Label.Text = inputLabels != null && inputLabels.Length > 1 ? inputLabels[1] : "Input 2";
00274                 Input2Label.Visibility = Visibility.Visible;
00275                 Input2TextBox.Visibility = Visibility.Visible;
00276                 Input3Label.Text = inputLabels != null && inputLabels.Length > 2 ? inputLabels[2] : "Input 3";
00277                 Input3Label.Visibility = Visibility.Visible;
00278                 Input3TextBox.Visibility = Visibility.Visible;
00279                 InputPanel.Children[1].Visibility = Visibility.Visible;
00280                 InputPanel.Children[2].Visibility = Visibility.Visible;
00281                 ButtonPanel.Visibility = Visibility.Visible;
00282                 YesButton.Visibility = Visibility.Collapsed;
00283                 NoButton.Visibility = Visibility.Collapsed;
00284                 OkButton.Visibility = Visibility.Collapsed;
00285                 ConfirmButton.Visibility = Visibility.Visible;
00286                 CancelButton.Visibility = Visibility.Visible;
00287                 _inputResults = new string[3];
00288                 ConfirmButton.Click += (s, e) =>
00289                 {
00290                     if (AreInputsValid(buttons))
00291                     {
00292                         _inputResults[0] = Input1TextBox.Text;
00293                         _inputResults[1] = Input2TextBox.Text;
00294                         _inputResults[2] = Input3TextBox.Text;
00295                         _result = true; // Confirm
00296                         Close();
00297                     }
00298                 };
00299                 CancelButton.Click += (s, e) => { _result = false; Close(); }; // Cancel
00300                 AdjustLabelWidths();
00301                 break;
00302         }
00303     }
00304
00310     private bool AreInputsValid(NotificationButtons buttons)
00311     {
00312         if (buttons == NotificationButtons.OneInput)
00313         {
00314             return !string.IsNullOrWhiteSpace(Input1TextBox.Text);
00315         }
00316         else if (buttons == NotificationButtons.TwoInputs)
00317         {
00318             return !string.IsNullOrWhiteSpace(Input1TextBox.Text) &&
00319                    !string.IsNullOrWhiteSpace(Input2TextBox.Text);
00320         }
00321         else if (buttons == NotificationButtons.ThreeInputs)
00322         {
00323             return !string.IsNullOrWhiteSpace(Input1TextBox.Text) &&
00324                    !string.IsNullOrWhiteSpace(Input2TextBox.Text) &&
00325                    !string.IsNullOrWhiteSpace(Input3TextBox.Text);
00326         }
00327         return true;
00328     }
00329
00333     private void AdjustLabelWidths()
00334     {
00335         TextBlock[] labels = new[] { Input1Label, Input2Label, Input3Label };
00336         double maxWidth = 0;
00337
00338         foreach (var label in labels)
00339         {
00340             if (label.Visibility == Visibility.Visible)
00341             {
00342                 label.Measure(new Size(double.PositiveInfinity, double.PositiveInfinity));
00343                 maxWidth = Math.Max(maxWidth, label.DesiredSize.Width);
00344             }
00345         }
00346
00347         foreach (var label in labels)
00348         {
00349             if (label.Visibility == Visibility.Visible)
00350             {
00351                 label.Width = maxWidth;
00352             }
00353         }
00354     }
00355
00359     private void UpdatePosition()
00360     {
00361         if (Owner != null)
00362         {
```

```
00363             var hwnd = new WindowInteropHelper(Owner).Handle;
00364             var monitor = MonitorFromWindow(hwnd, MONITOR_DEFAULTTONEAREST);
00365
00366             if (monitor != IntPtr.Zero)
00367             {
00368                 var monitorInfo = new MONITORINFO();
00369                 monitorInfo.cbSize = Marshal.SizeOf(typeof(MONITORINFO));
00370                 GetMonitorInfo(monitor, ref monitorInfo);
00371
00372                 var source = PresentationSource.FromVisual(Owner);
00373                 if (source != null)
00374                 {
00375                     var dpiScale = source.CompositionTarget.TransformToDevice.M11;
00376
00377                     if (Owner.WindowState == WindowState.Maximized)
00378                     {
00379                         // Center on the monitor's work area for maximized owner
00380                         double screenWidth = (monitorInfo.rcWork.Right - monitorInfo.rcWork.Left) / dpiScale;
00381                         double screenHeight = (monitorInfo.rcWork.Bottom - monitorInfo.rcWork.Top) / dpiScale;
00382                         double screenLeft = monitorInfo.rcWork.Left / dpiScale;
00383                         double screenTop = monitorInfo.rcWork.Top / dpiScale;
00384
00385                         this.Left = screenLeft + (screenWidth - this.ActualWidth) / 2;
00386                         this.Top = screenTop + (screenHeight - this.ActualHeight) / 2;
00387                     }
00388                     else
00389                     {
00390                         // Center relative to the owner window
00391                         this.Left = Owner.Left + (Owner.ActualWidth - this.ActualWidth) / 2;
00392                         this.Top = Owner.Top + (Owner.ActualHeight - this.ActualHeight) / 2;
00393                     }
00394
00395                     // Adjust height if the dialog is taller than the owner
00396                     if (this.ActualHeight > Owner.ActualHeight)
00397                     {
00398                         this.Top = Owner.Top;
00399                         this.Height = Owner.ActualHeight;
00400                     }
00401                 }
00402             }
00403         }
00404     }
00405
00409     private void Owner_PositionOrSizeChanged(object sender, EventArgs e)
00410     {
00411         UpdatePosition();
00412     }
00413
00417     private void Owner_StateChanged(object sender, EventArgs e)
00418     {
00419         UpdatePosition();
00420     }
00421 }
00422 }
```

Index