Table of Contents

Allen-Bradley Device Plugin

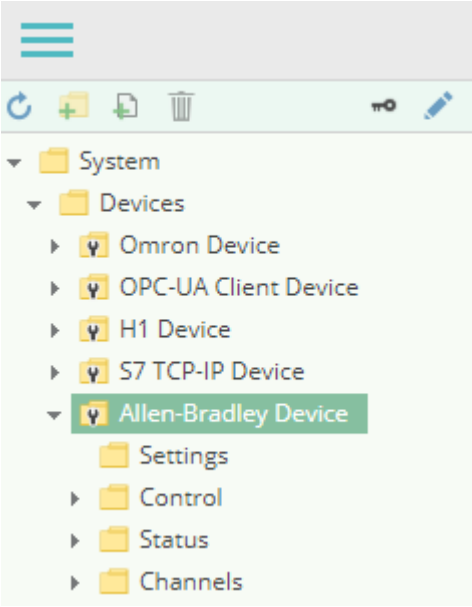
The Allen-Bradley Device Plugin allows reading and writing values from Allen-Bradley PLC devices over EtherNet/IP.

The following device types are supported:

- ControlLogix/CompactLogix
- Micro800
- MicroLogix
- PLC-5
- SLC 500

Configuration

The whole Allen-Bradley Device Plugin configuration is located in the node path `/System/Devices/Allen-Bradley Device`.



Channel

An Allen-Bradley Device Channel represents the connection to a Allen-Bradley PLC.

Settings

Address

IP address of the Allen-Bradley PLC.

Device Type

The device type of the Allen-Bradley PLC. The following device types are supported:

- ControlLogix/CompactLogix
- Micro800
- MicroLogix
- PLC-5
- SLC 500

CIP Path

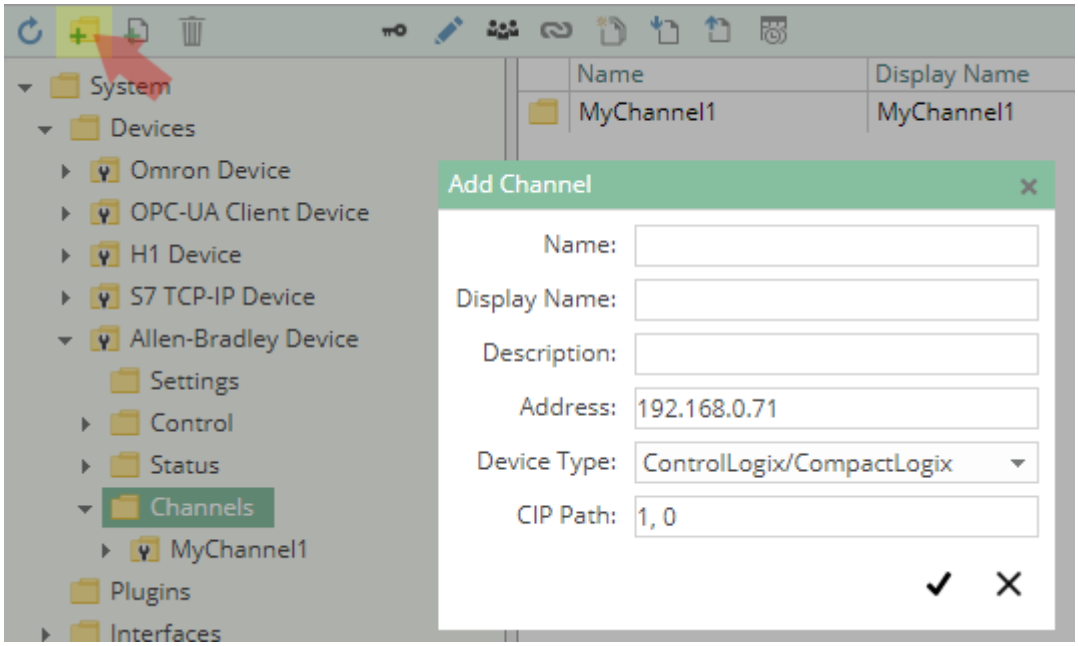
The CIP path to the PLC. This field must be specified for ControlLogix/CompactLogix as well as for a DH+ protocol bridge (i.e. a DHRIO module).
This field can have the format `A,B` where A specifies the port type (1=Backplane) and B specifies the slot in which the CPU is plugged.

Example: `1,0`

Operation Timeout

The timeout which shall be applied for read and write operations.

Adding a Channel



To add a new Allen-Bradley Channel, please follow these steps:

1. Add a Folder Node within the node `Allen-Bradley Device/Channels`, or right-click on the `Allen-Bradley Device/Channels` node and select `Add Channel`.
2. In the `Add Channel` dialog, specify the settings for the Allen-Bradley connection.
3. After clicking on “Save”, the channel node is created.
4. You can start the channel by selecting the channel node and clicking the start button.

Variables

Within the node `Variables` you can create datapoint nodes which can be read and written from/to the Allen-Bradley PLC as **tags**.
The `Value Type` property must be set to the corresponding tag type. Currently the following tag types are supported:

Tag Type	**CoDaBix Type**
`SINT`	`SByte` or `SByte-Array`
`USINT`	`Byte` or `Byte-Array`
`INT`	`Int16` or `Int16-Array`
`UINT`	`UInt16` or `UInt16-Array`
`DINT`	`Int32` or `Int32-Array`
`UDINT`	`UInt32` or `UInt32-Array`
`LINT`	`Int64` or `Int64-Array`
`ULINT`	`UInt64` or `UInt64-Array`
`REAL`	`Single` or `Single-Array`
`STRING` (or user-defined String data type)	`String` or `String-Array`

For **String** data types, an ASCII/ISO-8859-1-like encoding is assumed (the high byte of the 16-bit char will be cut off). While this can mean that surrogate pairs (for characters outside of the Unicode BMP) are not correctly handled, it will mean that the number of bytes is the same as the string length.

Note: Each signed/unsigned pendant of a datatype can also be used in place of the datatype. For example, SINT can also be used as Byte instead of SByte, but the MSB (most significant bit) will have a different interpretation, as is interpreted as sign bit for signed data types.

Path

The Path property of the node is used to specify the tag name, optionally a type specification, and (for arrays) optionally the offset and the length of the data:

```
<TagName>
<TagName>, <Length>
<TagName>, <Type>
<TagName>, <Type>[<Length>]
```

Placeholder	Description
<TagName>	The full name of the tag. If the tag is a program tag, it needs to be prefixed with Program:<Programmname>. using the corresponding program name. If the tag is a structure, you can specify the name of the field to access after a dot (.). If the tag (or a structure field) is an array, you can specify an array offset using the syntax [Offset].
<Length>	If the tag is an array, you can specify the array length here. It will only be used for reading, because when writing an array, its length will be determined from the value that is written.
<Type>	If present, contains a type specification, if it cannot be derived from the CoDaBix type. Currently, the following types can be specified: • STRING:<MaxLen>: Specifies the maximum string characters if the variable is of type String or String-Array. This can be used for user-defined String types; otherwise the predefined STRING data type with a maximum of 82 characters will be used.

Examples

Value Type	Path	Meaning
Int16	Local:1:O.Data	Data field (INT) of the digital output (Int16 contains bitmask of the 16 digital outputs (Digital Output Points))
Int16	Local:1:I.Data	Data field (INT) of the digital input (Int16 contains bitmask of the 16 digital inputs (Digital Input Points))
Int32	MyControllerTag	Controller Tag MyControllerTag (DINT)
Int32	Program:MainProgram.MyTag1.MyField1	From program MainProgram the tag MyTag1 (Structure) using field MyField1 (DINT)

Value Type	Path	Meaning
Byte-Array	Program:MainProgram.MyTag2, 20	From program MainProgram the tag MyTag2 (SINT[20]) with length 20 (range 0..20)
Byte-Array	Program:MainProgram.MyTag2[2], 16	From program MainProgram the tag MyTag2 (SINT[20]) from index 2 with length 16 (range 2..18)
Byte-Array	Program:MainProgram.MyTag3[4].MyField1[1], 5	From program MainProgram the tag MyTag3 (Structure[5]) at index 4 the field MyField1 (SInt[10]) from index 1 with length 5 (range 1..6)
String	MyStringTag1	Controller Tag MyStringTag1 (STRING with max. 82 characters)
String	MyStringTag2, STRING:20	Controller Tag MyStringTag2 (user-defined String type with max. 20 characters)
String-Array	MyStruct.MyStringTag3[2], STRING:20[3]	Controller Tag MyStruct (Structure) using field MyStringTag3 (user-defined String type with max. 20 characters) from index 2 with length 3 (range 2..5)

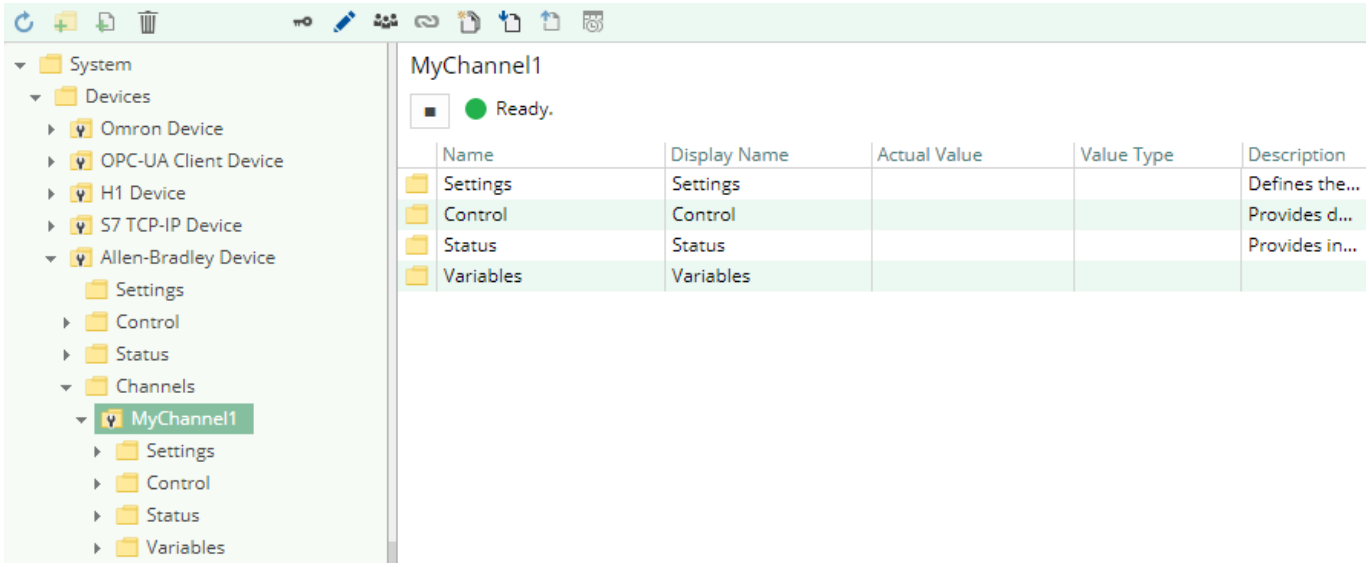
Name	Display Name	Actual Value	Value Type	Description	Path	Status
Output	Output	234	Int16		Local:1:O.Data	Good
Input	Input	0	Int16		Local:1:I.Data	Good
InputFault	InputFault	0	Int32		Local:1:I.Fault	Good
MyRealControllerTag	MyRealControllerTag	0	Single		MyRealControllerTag	Good
Tag1	Tag1	20	Int32		Program:MainProgram.Tag1	Good
Tag4	Tag4	30,5678	Single		Program:MainProgram.Tag4_REAL	Good
Array	Array	[10, 20, 30, 99, 200]	Int32-Array		Program:MainProgram.Tag5Array[0], 5	Good
StructTest1	StructTest1	9999	Int16		Program:MainProgram.MyStructTag1.MyField1	Good
StructTest2	StructTest2	3,1415	Single		Program:MainProgram.MyStructTag1.MyField2	Good

Diagnostics

The Allen-Bradley Device Plugin provides different status information depending on the layer to inspect. In general the channel-based diagnostic information is produced by the connection status of the channel to the PLC. The variable-based diagnostic information is produced during the read/write access of the different variables.

Channel

To monitor and diagnose the status of a Allen-Bradley Channel, take a look at the following image:



The image above depicts the Allen-Bradley Channel's Control Panel which displays all status relevant information. The control panel will automatically update its status information when a new status is available.

Status Circle

Color	Meaning
	The channel is stopped. Click the ► button to start it.
	The channel is currently in the progress of starting or stopping or is waiting to establish a connection.
	The channel is ready for doing read/write operations. You can stop it by clicking the ■ button.
	The channel is running, but the connection is currently in an error state. Please check the status text for more information.

Variables

To monitor and diagnose the status of the different variables, take a look at the variable's `Status` property displayed in CoDaBix. Use the button “Read actual Value” to read the values from the PLC and store the result into the variables.

Name	Display Name	Actual Value	Value Type	Description	Path	Status
Output	Output		Int16		Local:2:O.Data	Bad: PLCTAG_ERR_NOT_FOUND
Input	Input		Int16		Local:1:I.Data2	Bad: PLCTAG_ERR_BAD_PARAM
InputFault	InputFault		Int32		Local:1:I.	Bad: PLCTAG_ERR_TOO_LARGE
MyRealControllerTag	MyRealControllerTag		Single-Array		MyRealControllerTag, 5	Bad: PLCTAG_ERR_OUT_OF_BOUNDS
Tag2	Tag2		Int32		Program:MainProgram.Tag1	Bad: The operation has timed out.
Tag1	Tag1	20	Int32		Program:MainProgram.Tag1	Good

Common error messages:

- **PLCTAG_ERR_NOT_FOUND**: The tag was not found in the PLC. Check that the name is spelled correctly.
- **PLCTAG_ERR_BAD_PARAM**: The type or syntax of the tag is not valid. Check the tag name and if the correct value type is used.
- **PLCTAG_ERR_TOO_LARGE**: The read value cannot be stored in a variable of this type, because more data was returned than what fits into the type. Chose a value type which corresponds to the PLC type. If the tag is a structure, please specify the name of the field you want to access.
- **PLCTAG_ERR_OUT_OF_BOUNDS**: It was tried to access array indexes that are out of the array

bounds. Check the specified length and the offset of the array.
- **The operation has timed out.**: The PLC access has exceeded a specific timeout, or the PLC isn't available.

Log file

All channel related status information is also logged into the channel-specific log file stored in the `[LoggingFolder]`. Each log file is named in the naming scheme `Allen-Bradley Device.<ChannelName>.log`.

The content of such a log file can look as follows:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1],
Text=[The operation has timed-out.], Details=[]
...
```

Entities

Like every device plugin the Allen-Bradley Device Plugin extends the basic CoDaBix Device Model.

Device

The plugin's device type `AllenBradleyDevice` also defines the `AllenBradleyDeviceChannel` and therefore extends the basic `CodabixDevice` and `CodabixDeviceChannel` entities. While the `AllenBradleyDevice` just represents a concretization of the `CodabixDevice`, the `AllenBradleyDeviceChannel` extends the `CodabixDeviceChannel` with the Allen-Bradley Variable Entities.

Channel

Each channel is handled by a channel worker which establishes a TCP socket connection to the PLC.

By default, the worker does not read any values. When a client or plugin requests a synchronous read of the channel's variables in CoDaBix (e.g. using the CoDaBix Web Configuration's function “Read actual value”), the channel worker reads them from the underlying PLC and then writes them into the corresponding CoDaBix Nodes.

Similarly, when a client or plugin writes values into the channel's variables, the channel worker will write those values to the underlying PLC.

To have an Allen-Bradley variable being read steadily, you can edit the Node in the CoDaBix Web Configuration and set “History Options” to `Yes` (which will create an internal subscription), or you can use e.g. a OPC UA Client connected to the OPC UA Server plugin and create a subscription for the Allen-Bradley variable Nodes. In these cases, the channel worker reads the variables from the PLC at a regular interval and, if the value of one of the variables has changed, writes the new value into the

corresponding CoDaBix Node.

Folders & Files

Folders

Content	Path	Usage
AssemblyFolder	<CodabixInstallDir>/plugins/AllenBradleyDevicePlugin/	Contains the plugin's assembly file.
ConfigFolder	<CodabixProjectDir>/plugins/AllenBradleyDevicePlugin/	Contains the plugin's configuration file.
LoggingFolder	<CodabixProjectDir>/log/	Contains the plugin's log files.

Files

Type	Path	Usage
Assembly	[AssemblyFolder]/CoDaBix.AllenBradleyDevicePlugin.dll	The plugin's assembly file.
Logging	[LoggingFolder]/Allen-Bradley Device.<ChannelName>.log	The log file.

About Versions

This Document

Date	2019-11-11
Version	1.0

Plugin

Name	Allen-Bradley Device Plugin
Node	/System/Devices/Allen-Bradley Device
Version	1.0.0

Assembly

Name	CoDaBix.AllenBradleyDevicePlugin.dll
Date	2019-11-11
Version	1.0.0.0