v24.2.0

Drivers

eScada.Drivers.OpcUaTcp

eScada.Drivers.OpcUaTcp

(OPCUA – TCP Based transport)

OS availability
Windows, Linux, RaspBian

Atomic data type
Following OPCUA specifications for implemented data types.

Documentation reference
https://opcfoundation.org
Driver based on library https://www.open62541.org/

Parameters available in every section

Channel:	none		
Device:	Service end point		example: opc.tcp://192.168.1.105:49320 Multiple addresses can be expressed using multiple rows or a comma.
	Security mode		Anonymous, Login (not encrypted)
	Network timeout [ms]		
	Per-call timeout [ms]		
	Reconnect timeout [ms]		Waiting time before a reconnection after COMM break-down
	Tag prefix ID	**(1)**	
	Tag prefix	**(2)**	
Group:	Tag prefix	**(2)**	This property will be selected as first option even in case a global tag prefix is expressed on device.
Tag:	none		

(1)
This property must be used to select a different “Tag prefix” text, if it is specified using multi rows.
The value must be set accordingly with the row number in “Tag prefix” you want to use as tag prefix.

(2)
If your device uses a long prefix for tags addressing, instead of writing it every tag address, you can specify it in this property.
e.g. |var|CODESYS Control for Raspberry Pi MC SL.Application. HMI.

Multiple prefixes are allowed, please specify them on different rows as shown below.

|var|CODESYS Control for Raspberry Pi MC SL.Application.HMI.	selected with “Tag prefix ID” = **0**
OPCUA-Server.KEPServer.	selected with “Tag prefix ID” = **1**

The tag prefix will be inserted into tag address after the text s=
e.g. If your specified tag address is ns=2;s=OilPressure, the resulting final address will be:

ns=2;s=OPCUA-Server.KEPServer.OilPressure	with a prefix ID = **0**
ns=2;s=|var|CODESYS Control for Raspberry Pi MC SL.Application.HMI.OilPressure	with a prefix ID = **1**

remark: In case of a specific device doesn’t need tag prefix, <empty> text must be specified instead.

This particular setting plus multiple addresses, should be useful to keep a unique HMI project even using different devices; obviously is up to the programmer keep the same variables names among different devices.

Remarks for devices
The following attributes can be expressed for each device.

Bytes order actions	None, Swap bytes order, Swap bytes order in DWords, Swap words order, Swap bytes order in DWords then words order
String actions	None, Swap bytes in words

Implemented data types

OPCUA data type		Single element	HMI Array
Boolean	single bit	Yes	Yes
Byte, SByte	8 bit	Yes	Yes
UInt16, Int16	16 bit	Yes	Yes
UInt32, Int32	32 bit	Yes	Yes
UInt64, Int64	64 bit	Yes	Yes
Float	32 bit	Yes	Yes
Double	64 bit	Yes	Yes
String	1 byte per character	Yes	Yes
ByteString	1 byte per character	Yes	Yes

OPCUA Arrays are supported

Addressing

You can address every variable with a basic data type, using its node-id syntax

NodeId - XML Notation

The format of the node-id is:
ns=<namespaceIndex>;<identifiertype>=<identifier>

<namespace index>
The namespace index formatted as a base 10 number.
If the index is 0, then the entire “ns=0;” clause is omitted.

<identifier type>
A flag that specifies the identifier type. The flag has the following values:

Flag	Identifier Type
i	NUMERIC (UInteger)
s	STRING (String)
g	GUID not supported
b	Opaque/ByteString not supported

<identifier>
The identifier encoded as string. The identifier is formatted using the XML data type mapping for the identifier type. Note that the identifier may contain any non-null UTF8 character including whitespace.

Examples
ns=2;s=Channel1.Device1.array_double
name-space index 2, string identifier

s=Channel1.Device1.tag
name-space index 0, string identifier

ns=2;i=2048
name-space index 2, numeric identifier

i=2048
name-space index 0, numeric identifier

Multiple addresses

You can specify more than one address, using multiple rows
The property specified in device parameters called “Tag prefix ID”, will be used to select the address to use.

ns=1;s=MyVariable	with a prefix ID = **0**
ns=2;s=AnotherMyVariable	with a prefix ID = **1**

remark: please you should refer to your device documentation in order to get information about the <namespace index> and <identifier> to use; or you could use “UA Expert” client to browse UA servers:
https://www.unified-automation.com/products/development-tools

Variable type	Type	OPCUA type	Items
Boolean The number of items used declaring TAGs, must be a multiple of source PLC data size. Every group of booleans, must start from the first bit.			
Single bit	Bit	Boolean and all others numeric data type except for floating point	(C)
Byte			
Unsigned 8 bit	UInt8	Byte, SByte	(C)
Signed 8 bit	Int8		
16 bit			
Unsigned integer 16 bit	UInt16	UInt16, Int16	(C)
Signed integer 16 bit	Int16		
32 bit			
Unsigned integer 32 bit	UInt32	UInt32, Int32	(C)
Signed integer 32 bit	Int32		
Single precision 32 bit - (IEEE 754)	Float		
64 bit			
Unsigned integer 64 bit	UInt64	UInt64, Int64	(C)
Signed integer 64 bit	Int64		
Double precision 64 bit - (IEEE 754)	Double		
Strings String bytes can be interpreted as ASCII, UTF-7, UTF-8, UTF-16 or UTF-32 encoding			
Strings	String	String, ByteString	(C)
(C) It depends on how the OPCUA node is implemented on PLC side			

Consecutive items
The number of consecutive read/write items, depends on the device model.
Please have a look at ‘Implemented data types’ to understand which type of basic object can be addressed using array of items.