DNP 3.0 Slave Driver

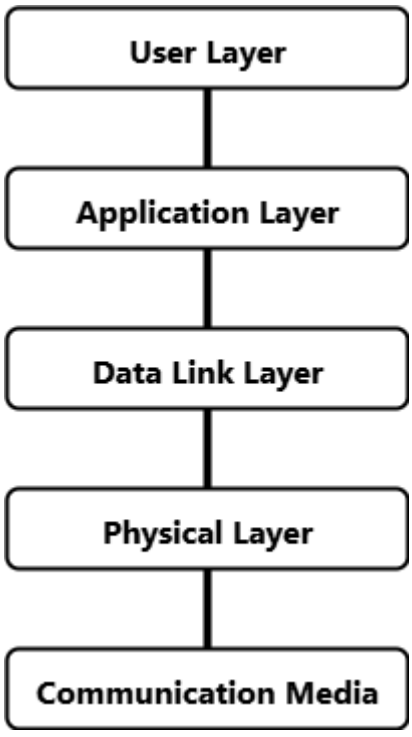
File Name	DNPSlave.dll
Manufacturer	DNP (Distributed Network Protocol)
Devices	
Protocol	DNP 3.0
Version	4.0.75
Last Update	02/09/2025
Platform	Win32
Dependencies	IOKit version 1.15 or later
Superblock Readings	No
Level	31201

Introduction

This Driver implements DNP 3.0 protocol in **Slave** mode (*Outstation*), according to levels 2 (two) and 3 (three).

DNP (*Distributed Network Protocol*) is an open and non-proprietary communication protocol, based on IEC (*International Electrotechnical Commission*) specifications, adapted for use in highly secure applications, on moderated speed and amount of data. It is extremely flexible and can be used in any hardware platform.

The model specified by ISO - OSI (*International Standards Organization - Open System Interconnection*) establishes 7 (seven) layers for a network protocol. IEC, on the other hand, specifies a simplified model, which is formed only by physical, data link, and application layers. Such model is called **EPA** (*Enhanced Performance Architecture*). The next figure displays EPA model's structure and its communication system.



Enhanced Performance Architecture model

A **User Layer** can be defined as the location where users manipulate data, after all communications. In **Elipse Software** applications, it is represented by a user application. The User layer uses this Driver's Application layer to send or receive full messages to or from a station.

An **Application Layer** is responsible for specifying in details the requests from the User layer, and back to it when a message comes from the Data Link layer. In other words, it gathers messages from the User layer, called fragments, into a message of multiple fragments with full information to be processed and sent to a station through the Data Link layer.

A **Data Link Layer** is used to pass messages from a primary, or origin, to a secondary, or destination, station. It also packs data, check it for transmission errors, and then send it to the TCP/IP network.

DNP protocol can be configured to exchange messages via polling or constant communication, or via integrity or changes, which is more efficient. Sending changes, also known as RBE (*Report by Exception*), can occur spontaneously, or not requested, or non-spontaneously by an explicit request from the **Master** side for these changes.

Function Codes

A function code identifies a message's purpose. There are groups of functions for requests and for answers. There are several types of functions for requests, as shown on the next table. Functions for answers are used internally and they are not available to users.

Function Codes

CODE	FUNCTION	DESCRIPTION
1	Read	Requests the specified Objects from a remote station and responds with the requested Objects that are available
2	Write	Stores the specified Objects on a remote station and responds with the operation status
3	Select	Selects or triggers output points but does not configure or produce any output, such as controls, setpoints, or analog outputs, and responds with the operation status. The **Operate** function must be used to activate these outputs
4	Operate	Configures or produces actions on outputs or previously selected points using the **Select** function
5	Direct Operate	Selects and operates the specified outputs and responds with the status of control points
6	Direct Operate NO ACK	Selects and operates the specified outputs and does not send a response, but it takes less time
7	Immediate Freeze	Copies the specified Objects to a freezing buffer
9	Freeze and Clear	Copies the specified Objects to a freezing buffer, then zeroes these Objects
10	Freeze and Clear No ACK	Copies the specified Objects to a freezing buffer and then zeroes these

CODE	FUNCTION	DESCRIPTION
		Objects, but does not send a confirmation response
11	Freeze with Time	Copies the specified Objects to a freezing buffer at the specified moment and responds with the operation status
13	Cold Restart	Performs a reset sequence
14	Warm Restart	Performs a partial reset sequence
20	Enable Unsolicited Messages	Enables a spontaneous notification for the specified Objects
21	Disable Unsolicited Messages	Disables a spontaneous notification for the specified Objects
22	Assign Classes	Links the specified Objects to a Class
23	Delay Measurement	Allows an application to calculate a delay or propagation time for a specific station

After specifying a message's purpose (APCI), there is a second part, if needed, called ASDU. Each ASDU is formed by one or more data identifiers (DUI), Object headers, and Object information or data fields.

Object Headers

An Object header from a message identifies data Objects in that message or the ones used as a response for that message. It is basically composed by an **Object**, a **Variation**, a **Qualifier**, and a **Range**.

Objects

Smart devices using DNP protocol's Application layer can monitor, control, and produce a large amount of data. This data, called **Information Elements** are processed and stored as information Objects, which are standardized so that they can be described and represented in a unique way. The categories of the existing data Objects are the following:

- **Static Objects**: Objects reflecting the current value of a field or internal variable
- **Event Objects**: Objects generated as the result of a value change or any other event. These are historical Objects, that is, they reflect a data value at any instant in the past
- **Frozen Static Objects**: Reflect the current frozen value of a field or internal variable. Data is frozen as the result of a data freezing request
- **Frozen Event Objects**: Objects resulting from a change in a frozen value or any other event. These are historical Objects, that is, they reflect a data value at any instant in the past

Each category is represented by a different Object, as described on the next table.

Data Objects

OBJECT	DESCRIPTION
Digital Inputs	The group of digital inputs contains all Objects representing binary inputs, that is, status or Boolean

OBJECT	DESCRIPTION
	attributes. Possible values vary between 1 (one) and 9 (nine)
Digital Outputs	The group of digital outputs contains all Objects representing binary outputs or information on relay control. Possible values vary between 10 and 19
Counters	This group contains all counter Objects. Possible values vary between 20 and 29
Analog Inputs	Contains all analog inputs. Possible values vary between 30 and 39
Analog Outputs	Contains all analog outputs. Possible values vary between 40 and 49
Time	Contains all Objects representing time in an absolute or relative form. Possible values vary between 50 and 59
Classes	This group contains all Objects representing data Classes or priorities. Possible values vary between 60 and 69
Files	Files or system files. Possible values vary between 70 and 79
Devices	Possible values vary between 80 and 89
Applications	Objects representing applications or operating system processes. Possible values vary between 90 and 99
Alternative numbers	Custom numerical representations. Possible values vary between 100 and 109

It is important to notice that a static Object, when varying, is capable of generating an event Object indicating that change. However, both represent the same Object.

Variation

These are modifications or sub-types that can occur on Objects, such as a digital input, which can be represented by a single bit (zero or one), by a status word (a byte), or even contain or not a time information (timestamp). Thus, the combination of an Object plus a Variation fully describes an information, according to the next examples.

- **Object 01**: Variation 01 represents a digital input without status (single bit)
- **Object 01**: Variation 02 represents a digital input with status (one byte)
- **Object 02**: Variation 01 represents a change to a digital input without a timestamp
- **Object 02**: Variation 02 represents a change to a digital input with a timestamp

Although both Objects are linked to the same digital input, this information can be represented in different ways.

Qualifier

Specifies the meaning of the **Range** field.

Range

Indicates the number of Objects, initial and final indexes, or identifiers for Objects.

Classes

Objects declared in a system or device implementing DNP protocol in **Slave** mode can be grouped into Classes. DNP protocol defines the standardized Classes described next.

- **Class 0**: This means all Objects, that is, during the initialization of this Driver on the **Master** side can perform a Class 0 (zero) request and in the response the **Slave** side sends the current value of all declared Objects
- **Classes 1 to 3**: These are entities that temporarily store lists of events or changes in Objects. Each Object must be configured on the **Slave** side to generate events when there are changes, and usually there is a pattern among DNP users to reserve Class 1 (one) for digital events, Class 2 (two) for analog events, and Class 3 (three) for counters

Driver Configuration

This topic contains information about the configuration of DNP 3.0 Slave Driver.

Extra Configuration

This topic contains information about this Driver's Properties Window, according to the next topics.

In addition to the Properties Window, this Driver's configurations can also be defined at run time in **Elipse E3**, **Elipse Power**, or **Elipse Water** applications. To do so, start this Driver in **Offline** mode, that is, execute the application with the **Start driver OFFLINE** option enabled, configurable on the **Setup** tab of Properties Window. The configuration options of this Driver are described on the next table.

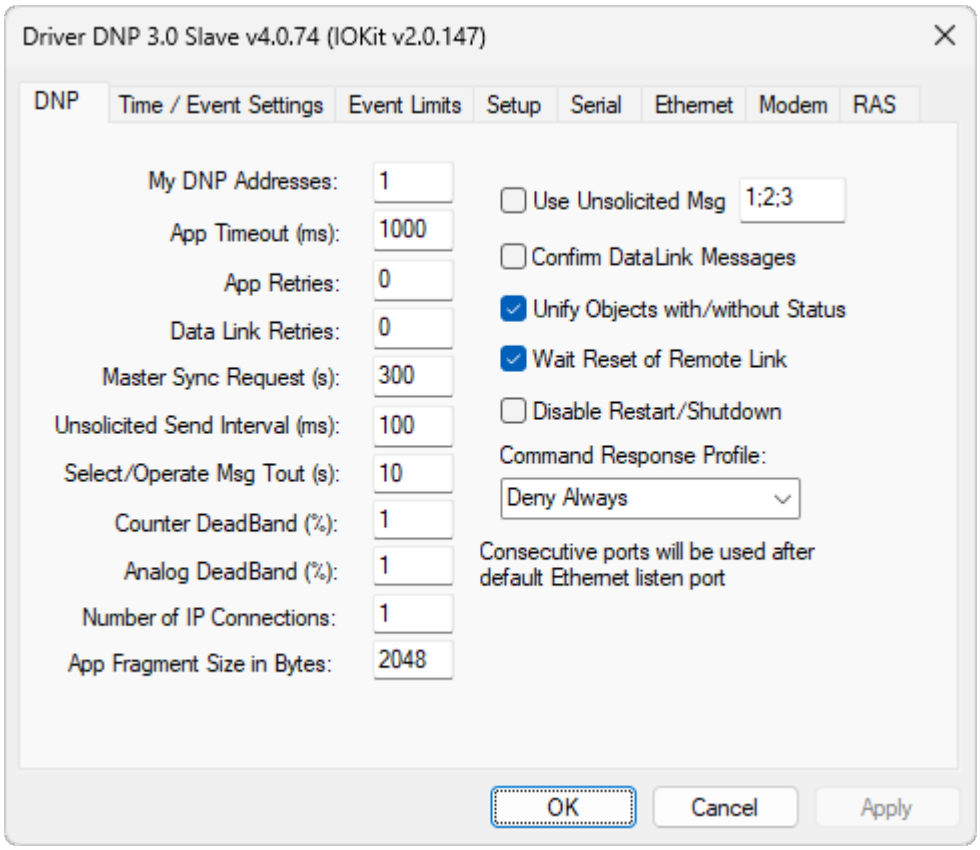
Configuration options for DNP 3.0 Slave Driver

PARAMETER	OFFLINE STRING	DATA TYPE
My DNP Addresses	DNP.myaddress	Word
App Retries	DNP.App_retries	Word
Data Link Retries	DNP.Dll_retries	Word
Select / Operate Msg Tout (s)	DNP.select_operate_tout	Word
Counter DeadBand (%)	DNP.counter_deadband	Float
Analog DeadBand (%)	DNP.analog_deadband	Float
App Timeout (ms)	DNP.App_timeout	Word
Use Unsolicited Messages	DNP.UseUnsol	Boolean
Unsolicited Classes	DNP.UnsolicitedClasses	String
Unify Objects With/Without Status	DNP.UnifyObjects	Boolean
Confirm DataLink Messages	DNP.ConfirmDLink	Boolean
Master Sync Request (s)	DNP.SyncTime	Word
Unsolicited Send Interval (ms)	DNP.SendInterval	Word
Enable Time Sync	DNP.EnableSync	Boolean
Sync Events with Stby Driver	DNP.SyncEvents	Boolean
Command Response Profile	DNP.CmdProfile	Word

PARAMETER	OFFLINE STRING	DATA TYPE
App Fragment Size in Bytes	DNP.MaxAppSize	DWord
Stop Analog Events if not connected	DNP.StopAnalogEventsWhenDisconnected	Boolean
Wait Reset of Remote Link	DNP.WaitResetOfRemoteLink	Boolean
Sort Analog events by Index	DNP.SortAnalogByIndex	Boolean
Ignore Assign Class Requests	DNP.IgnoreAssignClass	Boolean
Limit Event Class Msg Size - Single Datalink fragment	DNP.LimitEventMsg	Boolean
Report Events in GMT Time	DNP.ReportEventsInGMT	Boolean
Enable Time Zone Offset on sync	DNP.SyncTZOffset	Boolean
One Analog Event per Index	DNP.OneAnalogEvent	Boolean
Use Callbacks for commands	DNP.UseCallbacks	Boolean
Class 1 max events	DNP.MaxEventsClass1	DWord
Class 2 max events	DNP.MaxEventsClass2	DWord
Class 3 max events	DNP.MaxEventsClass3	DWord
Disconnect after unconfirmed unsolicited sequence count	DNP.MaxUnconfirmedUnsolicitedCount	Word
Generate event on first write	DNP.GenerateEventOnFirstWrite	Boolean
Number of IP Connections	DNP.IPConnections	Word
Disable Restart / Shutdown	DNP.DisableRestartShutdown	Boolean
Add new timestamped events on first connection	DNP.ReportEventsFirstConnection	Boolean
Add new timestamped events on reconnection	DNP.RepostEventsReconnection	Boolean
Allow multi app fragment Class 2,3 - No Confirm	DNP.MultiFragAppClass2_3_No_Confirm	Boolean

These properties can be written using a PLC Tag with the *N1* parameter equal to -1 (minus one), the *N2* parameter equal to 0 (zero), the *N3* parameter equal to 0 (zero), and the *N4* parameter equal to 3 (three).

DNP Tab



DNP tab

The available options on this tab are described on the next table.

Available options on the DNP tab

OPTION	DESCRIPTION
My DNP Addresses	Informs the DNP address of this Driver. Users can inform more than one DNP address by providing a numerical list separated by semicolons, such as "1;3;5". The first address on that list is considered the main or default DNP address
App Timeout (ms)	Maximum time the Application layer waits for a full response from the Data Link layer. If the Data Link layer is currently receiving a request, this time is automatically extended up to the end of Data Link's reception. The default value of this option depends on the baud rate used. Data Link's time-out, or byte by byte, is defined by **IOKit** library's time-out, configured on the **Setup** tab. The value in this option must be greater or equal to Data Link's time-out
App Retries	Number of I/O retries performed by the Application layer in case of a transaction error. Default value of this option is 0 (zero)
Data Link Retries	Number of I/O retries performed by the Data Link layer in case of a transaction error. Default value of this option is 0 (zero)
Master Sync Request (s)	Time interval in which this Driver requests a clock syncing to a **Master**, which may or may not be accepted,

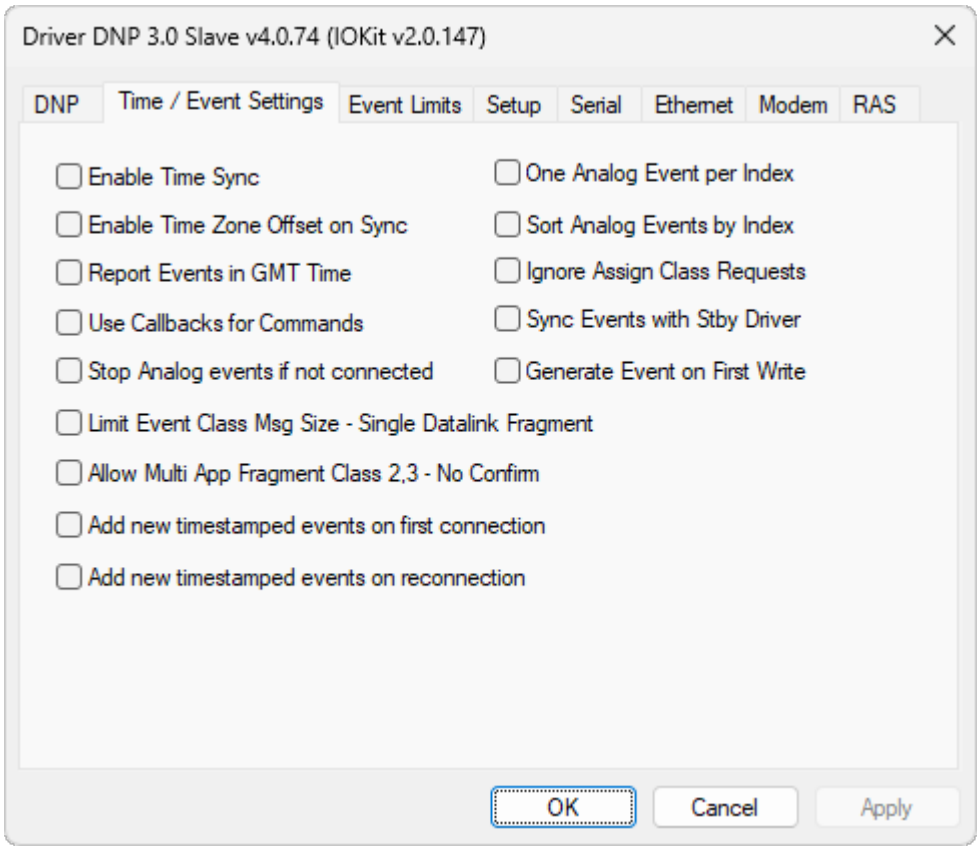
OPTION	DESCRIPTION
	depending on the configuration of the **Enable Time Sync** option on the **Time / Event Settings** tab. To disable sending, configure this option with the value 0 (zero)
Unsolicited Send Interval (ms)	Defines a time interval in which this Driver checks for events of Classes 1 (one), 2 (two), or 3 (three) available for sending through unsolicited messages, if the **Use Unsolicited Messages** option is enabled. The **Master** side can also enable or disable sending unsolicited messages from the **Slave** by sending functions **Enable Unsolicited Messages** (20) and **Disable Unsolicited Messages** (21)
Select / Operate Msg Tout (s)	Maximum time, in seconds, between a **Select** command and an **Operate** command. After this time, the **Operate** command is no longer accepted by this Driver
Counter DeadBand (%)	Informs a dead band, as a percentage, to notify events for counters
Analog DeadBand (%)	Informs a dead band, as a percentage, to notify events for analog points. To specify individual dead bands, please check dead band settings per Tag on topic **Tag Addressing**
Number of IP Connections	Defines the number of simultaneous connections, TCP or UDP, accepted from a TCP/IP port defined on the **Ethernet** tab. For example, when defining three connections and a TCP/IP port 20000, then this Driver can receive three connections on TCP/IP ports 20000, 20001, and 20002. Users can define a value between 1 (one) and 5 (five) connections
App Fragment Size in Bytes	Inform the maximum size of an application fragment. The default value for most systems is 2048 bytes
Use Unsolicited Messages	Informs whether this Driver sends unsolicited messages
Confirm DataLink Messages	Informs whether this Driver requests confirmation messages on the Data Link layer
Unify Objects With/Without Status	If this option is selected, then different Variations of an Object are unified into a single Variation with status. For example, if an application contains digital Tags of type Object 1 Variation 1 and also Variation 2, all indexes are unified into the same Object with Variation 2. If this option is no selected, all different Variations are kept as independent Objects. For example, if an application contains a Tag of type Object 1 Variation 1 Index 25, and another Tag of type Object 1 Variation 2 Index 26, they are reported to integrity as different Objects. It is not recommended that Objects with different Variations share the same Indexes, because both can generate events with the same Index, thus causing misinterpretations. This option is currently valid for Objects 101, 301, 1001, 1101, and 3004
Wait Reset of Remote Link	Instructs this Driver to only respond to communications after receiving a **Reset of Remote Link** command from a **Master**. The default value of this option is selected

OPTION	DESCRIPTION
Disable Restart / Shutdown	Disables the command to shut down a computer, which can be sent by a **Master** using DNP's function **13** (*Cold Restart*)
Command Response Profile	Informs how to handle commands requested to this Driver (Object 12 Variation 1 and Object 41 Variations 1, 2, 3, and 4). The available options are **Deny Always**: All commands are responded negatively and instantaneously, with their **Status** field reporting a code 4 (four, unsupported command), **Accept Always**: All commands are responded positively, with their **Status** field reporting a code 0 (zero, command OK), after an application read the respective command Tag. This indicates that a command was understood and an application is going to process it, and **Wait for Application Response**: Commands are only responded after an application reads and writes the respective Tag. The purpose of reading is for an application to understand that there was a request from a **Master**, which is then sent to another Driver or system output. After this processing, an application must write back to the Tag the same value read to indicate a successful processing, or a different value to indicate a failure. For more information, please check topic **Command Handling**

NOTE

A script is needed to send a message from a Driver to another. For more information, please check topic **Event Syncing with Redundancy**.

Time / Event Settings Tab



Time / Event Settings tab

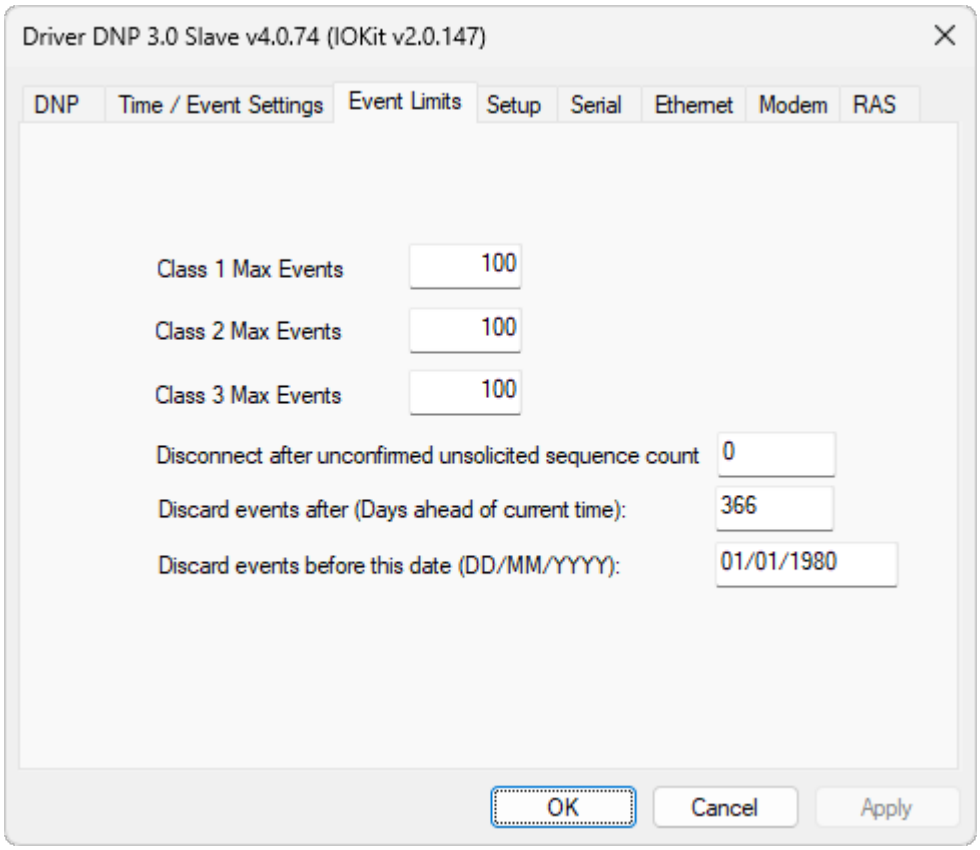
The available options on this tab are described on the next table.

Available options on the Time / Event Settings tab

OPTION	DESCRIPTION
Enable Time Sync	Informs whether this Driver accepts requests for clock syncing, which then sync Windows clock
Enable Time Zone Offset on Sync	Applies a local timezone to the clock to sync
Report Events in GMT TIme	Changes the timestamp of Tags, changing them to GMT (*Greenwich Mean Time*) based on Windows regional settings and daylight saving time
Use Callbacks for Commands	Notifies application Tags immediately when commands are received from a **Master**. If this option is not selected, that notification is based on the scan of each command Tag
Generate Event on First Write	Indicates whether the first writing of a Tag must generate an event or not. If this option is not selected, the first writing only creates a point on this Driver's internal database, and that value is used for comparison starting with the second writing, checking whether there was an event or change of value or quality. If this option is selected, the first writing creates a point on this Driver's internal database and generates an event
Limit Event Class Msg Size - Single Datalink Fragment	Limits event messages to a Data Link fragment at a time
Sync Events with Stby Driver	If there is a second **Slave** in the same application that is operating redundantly, that is, the **Master** side selects by

OPTION	DESCRIPTION
	which channel the communication is performed, this option instructs this Driver that for each received confirmation message of Class events, it generates a corresponding message to send to the redundant Driver
Stop Analog events if not connected	Stops accumulating analog events when the connection to a **Master** is interrupted. The current value of each analog Tag keeps being stored. This applies to Objects **32** and **33**
One Analog Event per Index	Sends only one event per index at each communication, so that communication become more compatible with implementations of **Masters** supporting a single event at a time
Sort Analog Events by Index	Instructs this Driver to sort analog events (Objects 3X with any Variation) by their Index and not by their timestamp, according to what is performed by default for all other events. This allows **Sag/Swell**-type events of a point to be reported together, so that the **Master** side understand this is an event of that type
Ignore Assign Class Requests	Instructs this Driver to ignore *Assign Class* requests received. An *Assign Class* request is a DNP function that allows linking certain Objects to a Class 1 (one), 2 (two), or 3 (three) event at run time
Allow Multi App Fragment Class 2,3 - No Confirm	Informs this Driver to send pending Class 2 (two) or 3 (three) events in multiple application fragments without a confirmation, in a response to a Class 2 (two) or 3 (three) request, if the response buffer does not contain any Class 1 (one) event. If it contains, then the response is limited to a single application fragment with confirmation
Add new timestamped event on First Connection	Informs, when the first connection of a **Master** occurs, whether this **Slave** Driver must generate new events with timestamp based on the current value of points. The goal is that the **Master** can receive values with timestamp before performing a Class 0 (zero), which by definition of DNP protocol, occurs with static values, or current, without a timestamp. This way, when connecting, if the **Master** requests events before a Class 0 (zero), or if unsolicited messages are enabled, then the **Master** receives the current values with a timestamp, if they are configured this way. This option is only used for Tags that are configured with **SOETYPE** with timestamp, that is, the *N1* parameter
Add new timestamped event on Reconnection	The same as the previous option, but when there are no reconnections

Event Limits Tab



Event Limits tab

Configure on this tab the maximum number of events this Driver must keep in memory for event Classes 1 (one), 2 (two), and 3 (three) in the **Class 1 Max Events**, **Class 2 Max Events**, and **Class 3 Max Events** options, respectively. In case the limit of these events is reached, the oldest events are removed from that list.

Users must also inform the maximum number of unsolicited messages that may remain unconfirmed. After this limit, this Driver closes the connection and waits for a new connection from a **Master**.

Tag Addressing

N1	SOETYPE × 10 + Class
N2	Function code. For more information, please check topic **Supported Function Codes**
N3	Object and Variation code. For more information, please check topic **Supported Objects**
N4	Address of a variable or number
Device	Optional parameter informing the logical DNP address, informed in the **My DNP Addresses** parameter. If it is not informed, a Tag is addressed to the first address of the list, or the main DNP address
Item	Optional parameter that informs the individual dead band of a Tag

The *N1* parameter contains a configuration for Classes and Events, according to the formula **SOETYPE × 10 + Class**. **SOETYPE** can be 0 (zero, without SOE), 1 (one, COS or Change of State without timestamp), or 2 (two, SOE or Sequence of

Events with timestamp). **Class** can be 1 (one), 2 (two), or 3 (three). The special cases for the value in the *N1* parameter are the following:

- If the *N1* parameter is equal to 999, this Tag represent a protocol's **On/Off** Tag, whose default value is 1 (one). When writing the value 1 (one), it turns off any communication, returning on the writing the value 1 (one)
- If the *N2* parameter is less than 0 (zero, special Functions), the *N1* parameter is used to identify a connection, ranging between 1 (one) and 5 (five)

The *N3* parameter must be informed according to the formula **ObjectCode × 100 + Variation**. **ObjectCode** is a type of Object, such as Binary Outputs, and **Variation** is a sub-type.

The *N4* parameter is the address of a variable or number, regardless of being a physical or logical point.

If it is necessary to inform an individual dead band per Tag, users can use the *Item* parameter, with the following options:

- **Percent Dead Band**: This parameter must be in the format **DB:X%**, in which *X* is a decimal value. For example, the value **DB:3%** corresponds to 3 (three) percent and the value **DB:0,6%** corresponds to 0,6 percent. Alternatively, users can also specify a relative dead band, using the format **DBR:X%**, in which *X* is a decimal value. Therefore, requests for snapshot values, such as Class 0 (zero), always report an updated value, while events are only generated if the value is above or below the dead band relative to the last reported event
- **Absolute Dead Band**: This parameter must be in the format **DB:XA**, in which *X* is a decimal value. For example, the value **DB:3A** corresponds to 3 (three) units in the scale of the variable itself. Like the previous option, users can specify a relative dead band, using in the Item parameter the format **DBR:XA**, in which *X* is a decimal value
- **BitString Dead Band**: Users can also specify a special dead band for integer analog Tags that represent a set of bits, such as **Word** or **DWord**, in which each bit represents a digital state. To do so, configure the *Item* parameter in the format **DB:BSTR**. In this case, if there is a change in any bit of this Tag, an event is generated, regardless of options **One Analog Event per Index** and **Stop Analog if not Connected**, allowing that changes in this Tag are not lost, regardless of the configuration of other analog Tags

Supported Function Codes

Supported function codes

N2	READING OR WRITING	OPERATION
-13	Reading	Statistics of communication per channel. Possible values for the *N4* parameter are **1**: Frames sent, **2**: Unanswered frames, **3**: CRC (*Cyclic Redundancy Check*) errors in the response frame, **4**: Sending errors, or **5**: Transmission retries
-20	Reading and writing	Communication turned on or off per channel. Possible values are **0**: Turned off or **1**: Turned on
-21	Reading	Communication inactive per channel. Possible values are **1**: Inactive or **0**: Active
-22	Reading	Reading quit events to send to a redundant Driver, per channel
-23	Writing	Writing event to be quit by this Driver, per channel

N2	READING OR WRITING	OPERATION
-42	Writing	Forces the generation of events based on the current value of all Objects of the same type declared in the *N3* parameter, *Object* × *100* + *Variation*. The event considers the configuration defined in the *N1* parameter, *SOETYPE* × *10* + *Class*
1	Writing	Read
2	Reading and writing	Write
3	Reading and writing	Select
4	Reading and writing	Operate
5	Reading and writing	Direct Operate
13	-	Cold Restart
14	-	Warm Restart
20	-	Enable Unsolicited Messages
21	-	Disable Unsolicited Messages
22	-	Assign Classes
23	-	Delay Measurement
24	-	Record Current Time

Supported Objects

Supported objects

OBJECT	TYPE	VARIATION	OBJECT NAME	FUNCTION CODE
1	Static	0 (zero)	Returns Object 1 (one) Variation 2 (two)	6 (six)
1	Static	2 (two)	Binary Input with Status	1 (one)
2	Event	1 (one)	Binary Input Change without Time	1 (one)
2	Event	2 (two)	Binary Input Change with Time	1 (one)
2	Event	3 (three)	Binary Input Change with Relative Time	1 (one)
3	Static	2 (two)	Double Bit Binary Input with Status	1 (one)
4	Event	1 (one)	Double Bit Binary Input Change without Time	1 (one)
4	Event	2 (two)	Double Bit Binary Input Change with Time	1 (one)
10	Static	1 (one)	Binary Output	1 (one)
10	Static	2 (two)	Binary Output Status	1 (one)

OBJECT	TYPE	VARIATION	OBJECT NAME	FUNCTION CODE
12	--	1 (one)	Control Relay Output Block	3 (three), 4 (four), 5 (five), 6 (six)
20	Static	2 (two)	Returns Object 20 Variations 1 (one) and 2 (two)	6 (six)
20	Static	1 (one)	32-bit Counter	1 (one), 7 (seven), 9 (nine), 10, 11
20	Static	2 (two)	16-bit Binary Counter	1 (one), 7 (seven), 9 (nine), 10, 11
20	Static	3 (three)	32-bit Delta Counter	1 (one), 7 (seven), 9 (nine), 10, 11
20	Static	4 (four)	16-bit Delta Counter	1 (one), 7 (seven), 9 (nine), 10, 11
20	Static	5 (five)	32-bit Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11
20	Static	6 (six)	16-bit Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11
20	Static	7 (seven)	32-bit Delta Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11
20	Static	8 (eight)	16-bit Delta Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	0 (zero)	Returns Object 21 Variations 1 (one) and 2 (two)	6 (six)
21	Static	1 (one)	32-bit Frozen Counter	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	2 (two)	16-bit Frozen Counter	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	3 (three)	32-bit Frozen Delta Counter	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	4 (four)	16-bit Frozen Delta Counter	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	5 (five)	32-bit Frozen Counter with Time Of Freeze	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	6 (six)	16-bit Frozen Counter with Time Of Freeze	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	7 (seven)	32-bit Frozen Delta Counter with Time Of Freeze	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	8 (eight)	16-bit Frozen Delta Counter with Time Of Freeze	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	9 (nine)	32-bit Frozen Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11

OBJECT	TYPE	VARIATION	OBJECT NAME	FUNCTION CODE
21	Static	10	16-bit Frozen Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	11	32-bit Frozen Delta Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11
21	Static	12	16-bit Frozen Delta Counter without Flag	1 (one), 7 (seven), 9 (nine), 10, 11
22	Event	1 (one)	32-bit Counter Change Event without Time	1 (one)
22	Event	2 (two)	16-bit Counter Change Event without Time	1 (one)
22	Event	3 (three)	32-bit Delta Counter Change Event without Time	1 (one)
22	Event	4 (four)	16-bit Delta Counter Change Event without Time	1 (one)
22	Event	5 (five)	32-bit Counter Change Event with Time	1 (one)
22	Event	6 (six)	16-bit Counter Change Event with Time	1 (one)
22	Event	7 (seven)	32-bit Delta Counter Change Event with Time	1 (one)
22	Event	8 (eight)	16-bit Delta Counter Change Event with Time	1 (one)
23	Event	1 (one)	32-bit Counter Change Event without Time	1 (one)
23	Event	2 (two)	16-bit Frozen Counter Event without Time	1 (one)
23	Event	3 (three)	32-bit Frozen Delta Counter Event without Time	1 (one)
23	Event	4 (four)	16-bit Frozen Delta Counter without Time	1 (one)
23	Event	5 (five)	32-bit Frozen Counter Event with Time	1 (one)
23	Event	6 (six)	16-bit Frozen Counter Event with Time	1 (one)
23	Event	7 (seven)	32-bit Frozen Delta Counter Event with Time	1 (one)
23	Event	8 (eight)	16-bit Frozen Delta Counter Event with Time	1 (one)

OBJECT	TYPE	VARIATION	OBJECT NAME	FUNCTION CODE
30	Static	0 (zero)	Returns Object 30 Variations 1 (one), 2 (two), and 5 (five)	6 (six)
30	Static	1 (one)	32-bit Analog Input	1 (one)
30	Static	2 (two)	16-bit Analog Input	1 (one)
30	Static	3 (three)	32-bit Analog Input without Flag	1 (one)
30	Static	4 (four)	16-bit Analog Input without Flag	1 (one)
30	Static	5 (five)	32-bit Analog Input Floating Point	1 (one)
30	Static	6 (six)	64-bit Analog Input Double Floating Point	1 (one)
31	Static	0 (zero)	Returns Object 31 Variations 1 (one), 2 (two), and 5 (five)	6 (six)
31	Static	1 (one)	32-bit Frozen Analog Input	1 (one)
31	Static	2 (two)	16-bit Frozen Analog Input	1 (one)
31	Static	3 (three)	32-bit Frozen Analog Input with Time Of Freeze	1 (one)
31	Static	4 (four)	16-bit Frozen Analog Input with Time Of Freeze	1 (one)
31	Static	5 (five)	32-bit Frozen Analog Input without Flag	1 (one)
31	Static	6 (six)	16-bit Frozen Analog Input without Flag	1 (one)
31	Static	7 (seven)	32-bit Frozen Analog Input Floating Point	1 (one)
32	Event	1 (one)	32-bit Change Event without Time	1 (one)
32	Event	2 (two)	16-bit Change Event without Time	1 (one)
32	Event	3 (three)	32-bit Analog Change with Time	1 (one)
32	Event	4 (four)	16-bit Analog Change Event with Time	1 (one)
32	Event	5 (five)	32-bit Analog Change Floating Point without Time	1 (one)
32	Event	6 (six)	64-bit Analog Change Double Floating Point	1 (one)

OBJECT	TYPE	VARIATION	OBJECT NAME	FUNCTION CODE
			without Time	
32	Event	7 (seven)	32-bit Analog Change Floating Point with Time	1 (one)
33	Event	1 (one)	32-bit Frozen Analog Event without Time	1 (one)
33	Event	2 (two)	16-bit Frozen Analog Event without Time	1 (one)
33	Event	3 (three)	32-bit Frozen Analog Event with Time	1 (one)
33	Event	4 (four)	16-bit Frozen Analog Event with Time	1 (one)
33	Event	5 (five)	32-bit Frozen Analog Floating Point without Time	1 (one)
33	Event	7 (seven)	32-bit Frozen Analog Floating Point with Time	1 (one)
34	Static	1 (one)	16-bit Analog Input Dead Band	1 (one)
34	Static	2 (two)	32-bit Analog Input Dead Band	1 (one)
34	Static	3 (three)	32-bit Analog Input Floating Point Dead Band	1 (one)
40	Event	1 (one)	32-bit Analog Output Status	1 (one)
40	Event	2 (two)	16-bit Analog Output Status	1 (one)
40	Event	3 (three)	32-bit Analog Output Status Floating Point	1 (one)
41	--	1 (one)	32-bit Analog Output Block	2 (two), 3 (three), 4 (four), 5 (five), 6 (six)
41	--	2 (two)	16-bit Analog Output Block	2 (two), 3 (three), 4 (four), 5 (five), 6 (six)
41	--	3 (three)	32-bit Floating Point Analog Output Block	2 (two), 3 (three), 4 (four), 5 (five), 6 (six)
41	--	4 (four)	64-bit Double Analog Output Block	2 (two), 3 (three), 4 (four), 5 (five), 6 (six)
50	Static	1 (one)	Time and Date	1 (one), 2 (two)
50	--	3 (three)	Last Recorded Time	2 (two)
51	Static	1 (one)	Time and Date CTO (*Common Time of Occurrence*)	1 (one)

OBJECT	TYPE	VARIATION	OBJECT NAME	FUNCTION CODE
51	Static	2 (two)	Unsynced Time and CTO	1 (one)
52	Static	2 (two)	Time Delay Fine	1 (one)
60	Event	1 (one)	Class 0 (zero) Data	1 (one)
60	Event	2 (two)	Class 1 (one) Data	1 (one)
60	Event	3 (three)	Class 2 (two) Data	1 (one)
60	Event	4 (four)	Class 3 (three) Data	1 (one)
80	--	1 (one)	Internal Indications	1 (one), 2 (two)

Supported Qualifiers

Supported qualifiers

OBJECT OR VARIATION	REQUEST QUALIFIER	RESPONSE QUALIFIER
Static Objects in General (Read-only)	0 (zero), 1 (one), 7 (seven), 8 (eight), 17, 18, 27, 28, 47, 58	Follows the request
Object 34 (Write-only)	0 (zero), 1 (one), 7 (seven), 8 (eight), 17, 18, 27, 28, 47, 58	Follows the request
Object 60 (Classes) or General Integrity Requests (Read-only)	6 (six)	17, 18, 27, 28, 47, 58
Object 50 (Write-only)	7 (seven), 2 (two)	7 (seven), 2 (two)
Object 80 (Write-only)	1 (one), 7 (seven)	7 (seven)
Objects 12 and 41 (Select and Operate, among others)	0 (zero), 1 (one), 7 (seven), 8 (eight), 17, 18, 27, 28, 47, 58	Follows the request

Examples of Tag Configuration

Assuming that a **Slave** DNP Driver wants to report to another system, which implements a **Master** DNP Driver, variables received from a certain protocol, the next examples show parameters *N1*, *N2*, *N3*, and *N4* in the format **N1.N2.N3.N4**:

- Digital Input *X*, generating events with timestamp in Class 1 (one): 21.1.102.X
- Digital Input *Y*, generating events without timestamp in Class 1 (one): 11.1.102.Y
- Analog Input *Z* with 16 bits generating events without timestamp in Class 3 (three): 13.1.3002.Z

General Comments

This topic provides additional information about the DNP 3.0 Slave Driver.

Functionality

This Driver must be used to operate in **Slave** mode, sending data to any DNP Driver in **Master** mode via Serial, Modem, or Ethernet TCP/IP.

Data sent to a **Master** through reading requests, usually commands for reading static variables or events, must be constantly informed to this Driver by writing to Tags, in which each Tag must reference a point on a database.

For the purpose of protocol's communication, this Driver only recognizes the existence of a DNP Object when the first value is written to an Object. From this moment on, the **Master** side can perform requests for this Driver, the **Slave** side, which then responds according to the requested Objects.

NOTE

Objects commonly read by the **Master** side, by using Function 1 (one) for reading, must have their corresponding Tags registered on this **Slave** Driver using the *N2* parameter equal to 1 (one), that is, writing this Tag to the **Slave** Driver informs an Object's current value. Starting with this writing, this Driver checks whether the new value informed, when compared to the previous value, corresponds to the generation of an event, according to the configuration of each Tag, which is informed in the *N1* parameter.

For example, a **Master** requests the value of Object 1 (one), Variation 2 (two), and Index 100. This digital point must be registered on the **Slave**'s database using a Tag with the *N1* parameter equal to 21, assuming that this point generates a SOE and is linked to Class 1 (one), the *N2* parameter equal to 1 (one), the *N3* parameter equal to 102, and the *N4* parameter equal to 100. When starting an application, this point does not contains any value. By writing a value, 0 (zero) or 1 (one), to this Tag, it is effectively created as an online point on this Driver's database.

NOTE

Therefore, an application must always receive data from somewhere and set it to this Driver's Tags to update their values.

For Tags declared as belonging to Classes 1 (one), 2 (two), or 3 (three), with events with or without timestamp, when performing an attribution, an automatic check is performed to verify whether a new event must be generated, according to the dead band if they are analog or counters, setting this new event available to the Class.

Even if the **DeadBand** property is declared as 0 (zero), users must perform a small change in value or quality to generate a new event. To prevent a dead band from checking, the value 100 can be added to the *N1* parameter, according to the next examples.

- A Tag with the *N1* parameter equal to 22 (SOE, Class 2), the *N2* parameter equal to 1 (one), the *N3* parameter equal to 3002, and the *N4* parameter equal to 10 (a 16-bit Analog point 10, with SOE)
- Current value equal to 100, current quality equal to 192, and timestamp equal to 1/1/2012 12:00:00

Writing to this Tag with the same value and quality, but with a different timestamp in the configuration with the *N1* parameter equal to 22, does not generate a new event. But if the *N1* parameter changes to 122, even if value and quality are the same, a new event is generated on writing. This is useful to represent trip events without a response or **Sag/Swell** events.

Single Block of Events

Users can create a single Block Tag to insert all events in this Driver, thus avoiding the creation of several individual Tags for each information. This Block contains the following configuration:

- **N1**: 50 (fixed value)
- **N2, N3, N4**: Not used
- **Size**: It must contain seven Elements:
  - **Element 0**: The *N1* parameter contains information *SOETYPE* × *10* + *Class*. For more information, please check topic **Tag Addressing**

- **Element 1**: The *N2* parameter contains a function code
- **Element 2**: The *N3* parameter contains information *Object* × *100* + *Variation*. For more information, please check topic **Tag Addressing**
- **Element 3**: The *N4* parameter contains the index of a point
- **Element 4**: Value of a point
- **Element 5**: Timestamp in system's native time format
- **Element 6**: Quality

NOTE

A Single Block of Events cannot be used together with other event Tags in an application, because in this case these events are duplicated.

Commands

A *Control Relay Output* object (Object 12, Variation 1) defines the fields of a message, described on the next table.

Byte 0 (zero, Control Code)

Bit	7	6	5	4	3	2	1	0
Meaning	Trip/Close		Clear	Queue	Code			

- **Trip/Close**: This field determines which control relay is activated in a system where a pair of trip and close relays is used to energize or de-energize points in the field. Possible values are, in binary format, **00**: NUL, **01**: Close, and **10**: Trip. The **NUL** value can be used to activate the selection relay without activating trip or close relays. In a system without selection relays, the **NUL** value does not perform any operation. In a system without trip or close relays, on the other hand, this field must always be **NUL** to indicate a normal operation of digital control, in which the exact point of control is implicit or fully known. This field does not support trip and close commands simultaneously
- **Clear**: If the command contains this field set to 1 (one, turned on), all control operations are removed from the queue, including the command currently executed, and this control operation is performed
- **Queue**: Indicates that a command was placed on a queue in a device. If this field is equal to 0 (zero, **NUL**), then no operation is placed on that queue and that queue is cleared from all controls, including the command currently executing if the **Clear** field is turned on. When the control function is executed and completed, it is removed from that queue. If the command contains the *Queue* attribute turned on, then this operation is queued again, that is, placed at the end of that queue for that point
- **Code**: This field specifies the type of operation. This command can be used with devices that support queuing commands, point to point, or other control mechanisms. In the first type, any control command must be queued for a point. In the second type, each control is performed until completed before the next command be accepted for that point

Possible values for Code, in binary format

VALUE	OPERATION	DESCRIPTION
0000	NUL	No operation is performed
0001	PULSE ON	The point, or points, is turned on by the time specified in **On Time**, turned off by the time specified in **Off Time**, and left in the **OFF** status
0010	PULSE OFF	The point, or points, is turned off by the time specified in **Off Time**, turned on by the time specified in **On Time**, and left in the **ON** status
0011	LATCH ON	Keeps a point or points in the **ON** status
0100	LATCH OFF	Keeps a point or points in the **OFF** status

NOTE
Values outside the previous table are not defined.

Byte 1 (one, Count)

This byte indicates how many times an operation is executed. This value is kept fixed in 1 (one) by this Driver.

Bytes 2 to 5 (On Time)

On-time, in milliseconds, defined on this Driver's extra configurations. This value is fixed for all commands.

Bytes 6 to 9 (Off Time)

Off-time, in milliseconds, defined on this Driver's extra configurations. This value is fixed for all commands.

Byte 10 (Status)

Status of an operation returned by this Driver if that operation was successfully finished. This status is only interpreted on the response, and it can be used by an application to check whether a command was executed successfully. These codes are the following:

- **0**: Command executed correctly, including **Select** and **Operate** operations
- **1**: **Operate** command sent after the maximum time of a **Select** command defined by the **Slave**
- **2**: **Operate** command sent without a previous **Select** command
- **3**: Message with formatting errors
- **4**: Operation not supported for this point
- **5**: Queue is full or this point is already active

- **6**: Hardware problems
- **Others**: Non-standard error codes

Handling Commands

This Driver supports receiving **Select**, **Operate**, or **Direct Operate** commands for Objects **12 Variation 1 (one)** and **41 Variations 1 (one)**, **2 (two)**, **3 (three)**, or **4 (four)**.

If the **Command Response Profile** option on the **DNP** tab is defined with the value **Wait for Application Response**, the full sequence to handle a command is the following:

1. A **Select**, **Operate**, or **Direct Operate** command is received.
2. A PLC or Block Tag with the *N2* parameter equal to 3 (three), 4 (four), or 5 (five) must be read and handled by an application, such as by sending to another Driver that then sends it to a device.
3. When receiving the response for this command to a device, users must write back the status to the same Tag whose command was read. If the response of this command to a device was positive, it must write the same value back. Otherwise, it must write a different value back.
4. If no writing was performed to the Tag during a 10-second interval, this Driver sends a response with a status equal to 6 (six, *Request Not Accepted. Hardware Problems*).

To receive a command request for these Objects, users can use an individual Tag or a Block Tag.

Object 12 Variation 1 (one)

For an individual Tag, this Tag's value when read contains the value of the **Control Code** field. To use **Select** and **Operate** operations, two Tags must be created and each one receives the reading at the moment the corresponding operation, **Select** or **Operate**, occurs. To use a **Direct Operate** operation (five), only a single Tag is needed with the following parameters:

- **N1**: Not used
- **N2**: 3 (three), 4 (four), or 5 (five)
- **N3**: 1201 or 41XX
- **N4**: Index of a point

For a Block Tag, the same previous configurations are valid, but this Block Tag can have up to 6 (six) Elements. The difference is that regardless the operation, **Select**, **Operate**,or **Direct Operate**, only a single Block Tag is needed, because the operation returns in Element 5 (five):

- **Element 0**: Control Code
- **Element 1**: Count
- **Element 2**: On Time
- **Element 3**: Off Time
- **Element 4**: Status
- **Element 5**: Operation (**3**: Select, **4**: Operate, or **5**: Direct Operate)
- **Element 6**: **Master** address

Object 41 Variations 1 (one), 2 (two), 3 (three), or 4 (four)

For an individual Tag, this Tag's value when read contains the value of the **Control Code** field. To use **Select** and **Operate** operations, two Tags must be created, and each one receives the reading at the moment the corresponding operation, **Select** or **Operate**, occurs. To use a **Direct Operate** operation (five), only a single operation is needed, with the following parameters:

- **N1**: Not used
- **N2**: 5 (five)
- **N3**: 1201 or 41XX
- **N4**: Index of a point

For a Block Tag, the same previous configurations are valid, but this Block Tag can have up to 3 (three) Elements:

- **Element 0**: Value
- **Element 1**: Status
- **Element 2**: Operation

Example of a script to handle the reception of commands in an Object 12 Variation 1 (one).

```
// The OpenClose Block is formed by 5 (five) Elements
// This example assumes sending the received command
// to another DNP Master Driver

Sub OpenClose_OnRead()
  Set Digital = Parent.Parent.Item("DigitalReading")
  ControlCode = Item("ControlCode").Value

  Trip = 1
  Close = 0
  CmdOk = 1

  Select Case ControlCode
    Case 65 'Pulse On Close
      VCommand = Close
    Case 66 'Pulse Off close
      VCommand = Close
    Case 67 'Latch On Close
      VCommand = Close
    Case 68 'Latch Off Close
      VCommand = Close
    Case 129 'Pulse On Trip
      VCommand = Trip
    Case 130 'Pulse Off Trip
      VCommand = Trip
    Case 131 'Latch On Trip
      VCommand = Trip
    Case 132 'Latch Off Trip
      VCommand = Trip
    Case Else 'Invalid or not formatted command
      CmdOk = 0
  End Select

  If Not(CmdOk) Then
    WStatus = 7
  Else
    If Digital.Item("STATUSDL01").WriteEx(VCommand, , , WStatus) Then
      WStatus = 0 'Forces a value (0) or success when writing
      Status expected by a Master in case of error:
      Select Case WStatus
        Case 1
          Endtext = " Operate received after selection time-out"
        Case 2
          Endtext = " No previous selection message"
        Case 3
          Endtext = " Format error in command"
        Case 4
          Endtext = " Operation not supported for this point"
        Case 5
          Endtext = " Queue is full or this point is already active"
        Case 6
          Endtext = " Hardware problems"
        Case Else
          Endtext = " Undefined problem"
      End Select
    Else
      WStatus = 7
    End If
  End If
  Item("Status").Value = WStatus
  Write(EWriteSyncMode)
End Sub
```

Information on Quality

All Tags use their informed quality when writing to propagate the status of the original point.

To do so, a simple mapping is performed. If a Tag contains a quality equal to or greater than 192, the DNP status of the corresponding point has its Online bit equal to 1 (one). Otherwise, its Online bit is equal to 0 (zero), according to the examples on the next table for Objects 1 (one), 2 (two), and 10.

Bit	7	6	5	4	3	2	1	0
Meaning	XX	XX	CF	LF	RF	CL	RS	OL

Event Syncing with Redundancy

If the **Sync Events with Stby Driver** option on the **Time / Event Settings** tab is enabled, each redundant **Slave** DNP Driver must have 2 (two) Block Tags to receive and send events to quit.

By using a Block Tag with the *B2* parameter equal to -22, users can read quit events in one Driver and by using a Block Tag with the *B2* parameter equal to -23, users can write events to quit to another Driver.

In both cases, this Block Tag must have 6 (six) Elements, described next:

- **Element 0**: Object × 100 + Variation
- **Element 1**: Index
- **Element 2**: Class (one, two, or three)
- **Element 3**: Value
- **Element 4**: Valid timestamp (**1**: Valid or **0**: Invalid)
- **Element 5**: Timestamp in 64-bit floating point format

Example of a code in **Elipse E3**, **Elipse Power**, or **Elipse Water**, assuming the existence of **SlaveDriver1** quitting events in **SlaveDriver2** and vice versa.

```
Sub SyncEvtsRead_OnRead() //Script in SlaveDriver1
  Set EvtTag = Application.GetObject("SlaveDriver2.SyncEvtsWrite")
  EvtTag.Item("Element1").Value = Item("Element1").Value
  EvtTag.Item("Element2").Value = Item("Element2").Value
  EvtTag.Item("Element3").Value = Item("Element3").Value
  EvtTag.Item("Element4").Value = Item("Element4").Value
  EvtTag.Item("Element5").Value = Item("Element5").Value
  EvtTag.Item("Element6").Value = Item("Element6").Value
  EvtTag.Write()
End Sub

Sub SyncEvtsRead_OnRead() //Script in SlaveDriver2
  Set EvtTag = Application.GetObject("SlaveDriver1.SyncEvtsWrite")
  EvtTag.Item("Element1").Value = Item("Element1").Value
  EvtTag.Item("Element2").Value = Item("Element2").Value
  EvtTag.Item("Element3").Value = Item("Element3").Value
  EvtTag.Item("Element4").Value = Item("Element4").Value
  EvtTag.Item("Element5").Value = Item("Element5").Value
  EvtTag.Item("Element6").Value = Item("Element6").Value
  EvtTag.Write()
End Sub
```

Cold and Warm Restart

When receiving a **Cold Restart** command, after sending a response message, this Driver restarts the operating system.

When receiving a **Warm Restart** command, after sending a response message, this Driver clears the queue of events waiting to be sent or quit and puts itself in an initialization status, waiting for a **Reset of Remote Link** message. The current values of Tags on this Driver's database remain in memory.

Documentation of I/O Interfaces

This section contains the documentation of I/O Interfaces referring to the **DNPSlave** Driver.

Configuration of a Driver

I/O Interface configuration is performed on a Driver's configuration dialog box. To access the configuration of this dialog box in **Elipse E3** in version 1.0, follow these steps:

1. Right-click a Driver object (IODriver).
2. Select the **Properties** item on the contextual menu.
3. Select the **Driver** tab.
4. Click **Other parameters**.

In **Elipse E3** version 2.0 or later, click **Configure driver** on a Driver's toolbar. In **Elipse SCADA**, follow these steps:

1. Open the Organizer.
2. Select a Driver on Organizer's tree.
3. Click **Extras** on the **Driver** tab.

Currently, an I/O Interface allows opening only one connection for each Driver. This means that, if users want to access two serial ports, they must add two Drivers to an application and then configure each one of these Drivers for each serial port.

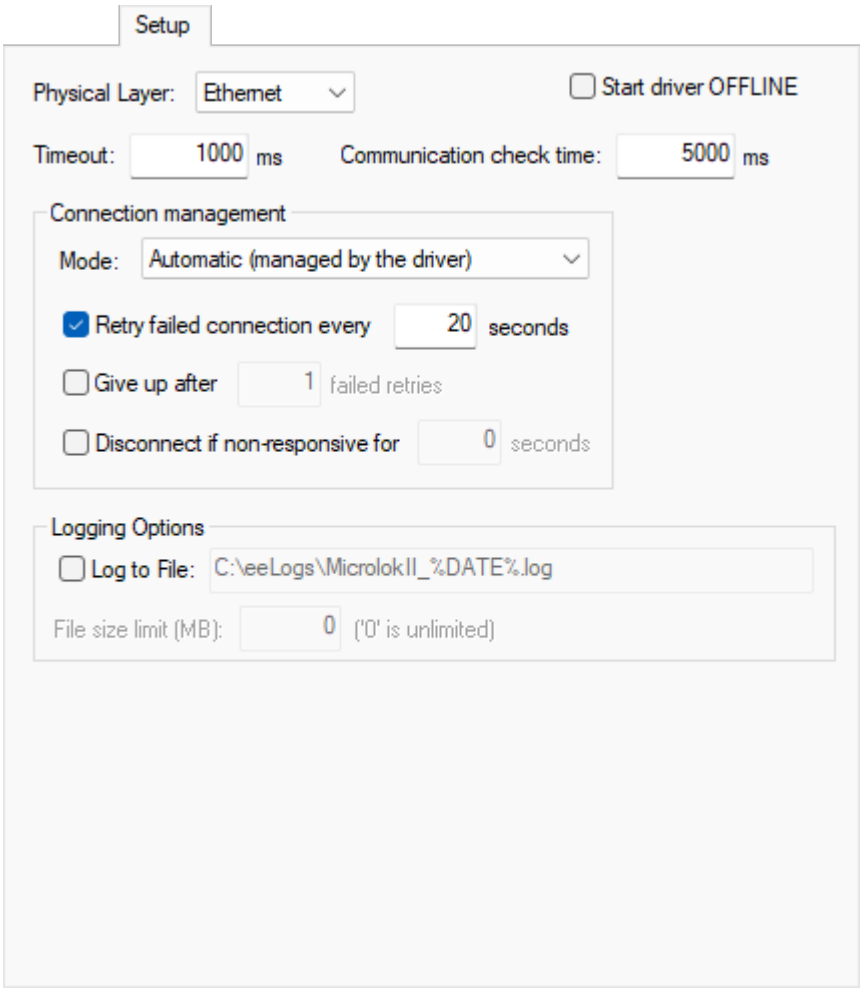
Configuration Dialog Box

The dialog box of I/O Interfaces allows configuring the I/O connection used by a Driver. This dialog box contains the **Setup**, **Serial**, **Ethernet**, **Modem**, and **RAS** tabs, described on the next topics. If a Driver does not implement a specific I/O connection, its corresponding tab is not available for configuration. Some Drivers may contain additional tabs, specific for that Driver, on the configuration dialog box.

Setup Tab

The **Setup** tab contains general configurations of a Driver. This tab is divided into the following groups:

- **General configurations**: Configurations of a Driver's physical layer, time-out, and initialization mode
- **Connection management**: Configurations on how the I/O Interface keeps a connection and which recovery policy is used on failure
- **Logging options**: Controls the generation of log files



Setup tab

General options on the Setup tab

OPTION	DESCRIPTION
Physical Layer	Select the physical layer on a list. Available options are **Serial**, **Ethernet**, **Modem**, and **RAS**. The selected interface must be configured on its specific tab
Timeout	Configure a time-out, in milliseconds, for the physical layer. This is the amount of time an I/O interface waits to receive any byte from the reception's buffer
Communication check time	Set the time, in milliseconds, to define the interval at which communication is considered to be in an inactive state. As long as an I/O Driver receives valid data, its communication state is considered active. However, if during operation an I/O Driver does not receive valid data inside this period of time, the state is considered inactive. The communication state is shown in the **IO.CommunicationStatus** Tag
Start driver OFFLINE	Select this option so that a Driver starts in **Offline** mode or stopped. This means that the I/O interface is not created until this Driver is configured to **Online** mode by using a Tag in an application. This mode enables a dynamic configuration of an I/O interface at run time

Options on the Connection management group

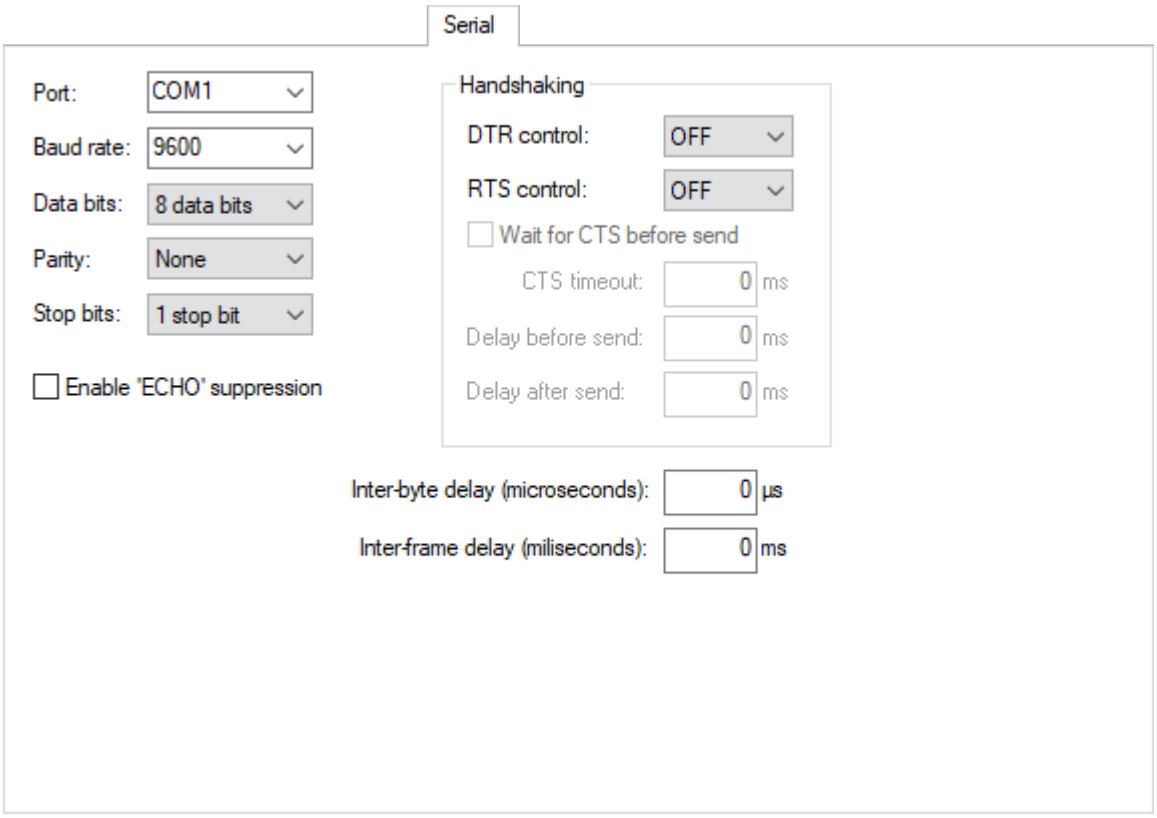
OPTION	DESCRIPTION
Mode	Selects a management mode of a connection. Selecting the **Automatic** option allows a Driver to manage the connection automatically, as specified in the next options. Selecting the **Manual** option allows an application to fully manage a connection
Retry failed connection every ... seconds	Select this option to enable a Driver's connection retry in a certain interval, in seconds. If the **Give up after failed retries** option is not selected, this Driver keeps retrying until a connection is performed, or until the application is stopped
Give up after ... failed retries	Enable this option to define a maximum number of connection retries. When the specified number of consecutive connection retries is reached, a Driver goes to the **Offline** mode, assuming that a hardware problem was detected. If a Driver establishes a successful connection, the number of unsuccessful retries is cleared. If this new connection is lost, then the retry counter starts at zero
Disconnect if non-responsive for ... seconds	Enable this option to force a Driver to disconnect if no byte was received by the I/O interface during the specified time-out, in seconds. This time-out must be greater than the time-out configured in the **Timeout** option

Options on the Logging Options group

OPTION	DESCRIPTION
Log to File	Enable this option and configure the name of a file to write a log. Log files can be large, so use this option for short periods of time, only for testing and debugging purposes. If the **%PROCESS%** macro is used in the log file name, it is replaced by the identifier of the current process. This option is particularly useful when using several instances of the same Driver in **Elipse E3**, thus allowing each instance to generate a separate log file. For example, when configuring this option with value "c:\e3logs\drivers\sim_%PROCESS%.log", it generates a file named c:\e3logs\drivers\sim_00000FDA.log for process **0FDAh**. Users can also use the **%DATE%** macro in the file name. In this case a log file is generated every day, in the format **aaaa_mm_dd**. For example, when configuring this option with value "c:\e3logs\drivers\sim_%DATE%.log", it generates a file named c:\e3logs\drivers\sim_2005_12_31.log in 12/31/2005 and a file named c:\e3logs\drivers\sim_2006_01_01.log in 01/01/2006. Similarly, the **%DATE_HOUR%** macro generates one log file per hour, in the format **aaaa_mm_dd_hh**
File size limit (MB)	Configure the log file size limit, in megabytes. A value equal to 0 (zero) means that there is no size limit for the log file

Serial Tab

Use this tab to configure parameters for a **Serial** Interface.



Serial tab

General options on the Serial tab

OPTION	DESCRIPTION
Port	Select a serial port on the list, from **COM1** to **COM4**, or type the name of a serial port in the format **COM*n***, such as "COM15". When typing the name of a serial port manually, the dialog box only accepts names of serial ports starting with the expression "COM"
Baud rate	Select a baud rate on the list (**1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600**, or **115200**) or type a baud rate, such as 600
Data bits	Select 7 (seven) or 8 (eight) data bits on the list
Parity	Select a parity on the list. The available options are **None**, **Even**, **Odd**, **Mark**, or **List**
Stop bits	Select the number of stop bits on the list. The available options are **1**, **1.5**, or **2** stop bits
Enable 'ECHO' supression	Enable this option to remove the echo received after the I/O Interface sends data via serial port. If this echo is not equal to the bytes just sent, then the I/O Interface aborts communication
Inter-byte delay (microseconds)	Defines a delay between each byte transmitted by the I/O Interface, in millionths of a second, that is, 1000000 is equal to a second. This option must be used with small delays of less than a millisecond
Inter-frame delay (milliseconds)	Defines a delay between packets sent or received by the I/O Interface, in thousandths of a second, that is, 1000 is equal to a second. This delay is applied if the I/O Interface

OPTION	DESCRIPTION
	sends two consecutive packets, or between a received packet and the next sending

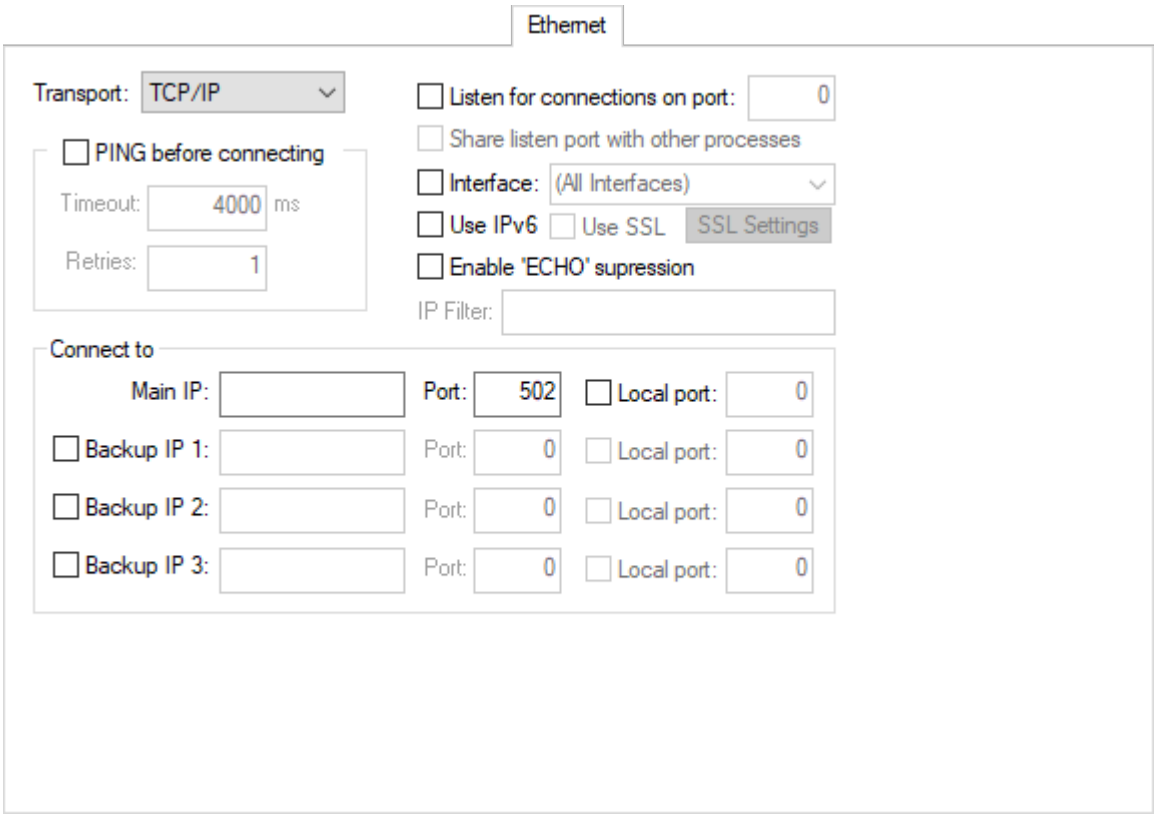
The **Handshaking** group configures the usage of **RTS**, **CTS**, and **DTR** signals in the handshaking process, that is, it controls when data can be sent or received via serial line. Most of the time, configuring the **DTR control** option to **ON** and the **RTS control** option to **Toggle** works with **RS232**-type serial lines as well as with **RS485**-type serial lines.

Available options on the Handshaking group

OPTION	DESCRIPTION
DTR control	Select the value **ON** to keep the **DTR** signal always on while the serial port is open. Select the value **OFF** to turn the **DTR** signal off while the serial port is open. Some devices require the **DTR** signal always on to allow communication
RTS control	Select the value **ON** to keep the **RTS** signal always on while the serial port is open. Select the value **OFF** to turn the **RTS** signal off while the serial port is open. Select the value **Toggle** to turn the **RTS** signal on while sending bytes via serial port and turn it off when not sending bytes, therefore enabling the reception
Wait for CTS before send	Available only when the **RTS control** option is configured with the value **Toggle**. Use this option to force a Driver to check the **CTS** signal before sending bytes via serial port, after turning the **RTS** signal on. In this mode, the **CTS** signal is handled as a permission flag for sending
CTS timeout	Determines a maximum time, in milliseconds, that a Driver waits for the **CTS** signal after turning the **RTS** signal on. If the **CTS** signal is not turned on within this time-out, that Driver then fails the current communication and returns an error
Delay before send	Some serial port devices have a delay when enabling a data sending circuit after the **RTS** signal is turned on. Configure this option to wait a certain number of milliseconds after turning the **RTS** signal on and before sending the first byte. **IMPORTANT**: This delay must be used carefully, because it uses 100% of CPU resources while waiting. System's general performance degrades as this value increases
Delay after send	This is the same effect of the **Delay before send** option, but in this case the delay is performed after sending the last byte, before turning the **RTS** signal off

Ethernet Tab

Use this tab to configure parameters of an **Ethernet** Interface. These parameters, except port configurations, must also be configured for use in the **RAS** Interface.



Ethernet tab

Available options on the Ethernet tab

OPTION	DESCRIPTION
Transport	Select the value **TCP/IP** for a TCP socket (*stream*) or select the value **UDP/IP** to use a UDP socket (*connectionless datagram*)
Listen for connections on port	Use this option to wait for new connections in a specific IP port, common in Slave Drivers. If this option remains unselected, a Driver connects to the address and port specified in the **Connect to** option
Share listen port with other processes	Select this option to share the listening port with other Drivers and processes
Interface	Select the local network interface, identified by its IP address, that a Driver uses to establish and receive connections, or select the value **(All Interfaces)** to allow connection in any network interface
Use IPv6	Select this option to force a Driver to use addresses in **IPv6** format on all Ethernet connections. Leave this option deselected to use the **IPv4** format
Enable 'ECHO' suppression	Enable this option to remove the echo from received data. An echo is a copy of sent data, which can be returned before a reply message
IP Filter	List of restricted or allowed IP addresses from where a Driver accepts connections (*Firewall*). Please check the **IO.Ethernet.IPFilter** property for more information
PING before connecting	Enable this option to execute a **ping** command, that is, to check whether a device can be reached on a network, for a device before trying a socket connection. This is a quick way

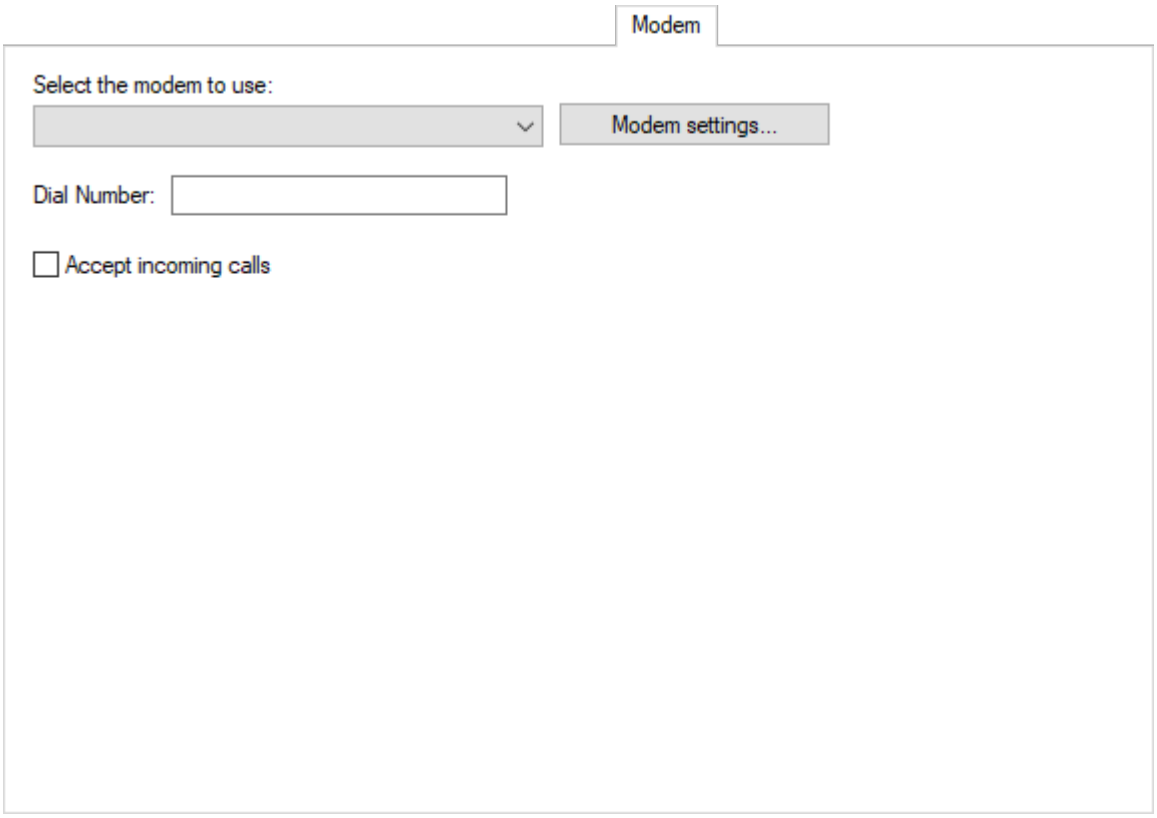
OPTION	DESCRIPTION
	of determining a successful connection before trying to open a socket with a device. The time-out of a connection with a socket can be very high. The available options are: • **Timeout**: Specify the number of milliseconds to wait for a reply from a **ping** command. Users must use a **ping** command to check the normal reply time, configuring this option for a value above that average. Usually this value can be configured between 1000 and 4000 milliseconds, that is, between 1 (one) and 4 (four) seconds • **Retries**: Number of retries of a **ping** command, not counting the first attempt. If all attempts fail, then the socket connection is aborted

Available options on the Connect to group

OPTION	DESCRIPTION
Main IP	Type the IP address of a remote device. Users can use an IP address separated by dots, as well as a URL. In case of a URL, a Driver uses the available DNS service to map that URL to an IP address, such as "192.168.0.13" or "Server1"
Port	Type the IP port of a remote device, between 0 (zero) and 65535
Local port	Select this option to use a fixed local IP port when connecting to a remote device
Backup IP 1, 2, and 3	Indicate the IP address, the IP port, and the fixed local IP port of up to 3 (three) backup addresses of a remote device

Modem Tab

Use this tab to configure parameters of a **Modem** Interface. Some options on the **Serial** tab affect the configuration of a modem, therefore users must also configure the **Serial** Interface.



Modem tab

The **Modem** Interface uses the TAPI modems installed on the computer.

Available options on the Modem tab

OPTION	DESCRIPTION
Select the modem to use	Select a modem on the list of available modems on the computer. If the value **Default modem** is selected, then the first available modem is used. Selecting this option is recommended specially when an application is used on another computer
Modem settings	Click to open the configuration window of the selected modem
Dial Number	Type a default number for dialing. This value can be changed at run time. Users can use the **w** character to represent a pause or a waiting time for a dial tone. For example, "0w33313456" dials the number 0 (zero), waits, and then dials the number "33313456"
Accept incoming calls	Enable this option so that a Driver answers the phone when receiving an external call. To use this option, users must configure the **Connection management** option on the **Setup** tab to the value **Manual**

RAS Tab

Use this tab configure parameters of a **RAS** Interface. Users must also configure the **Ethernet** tab.

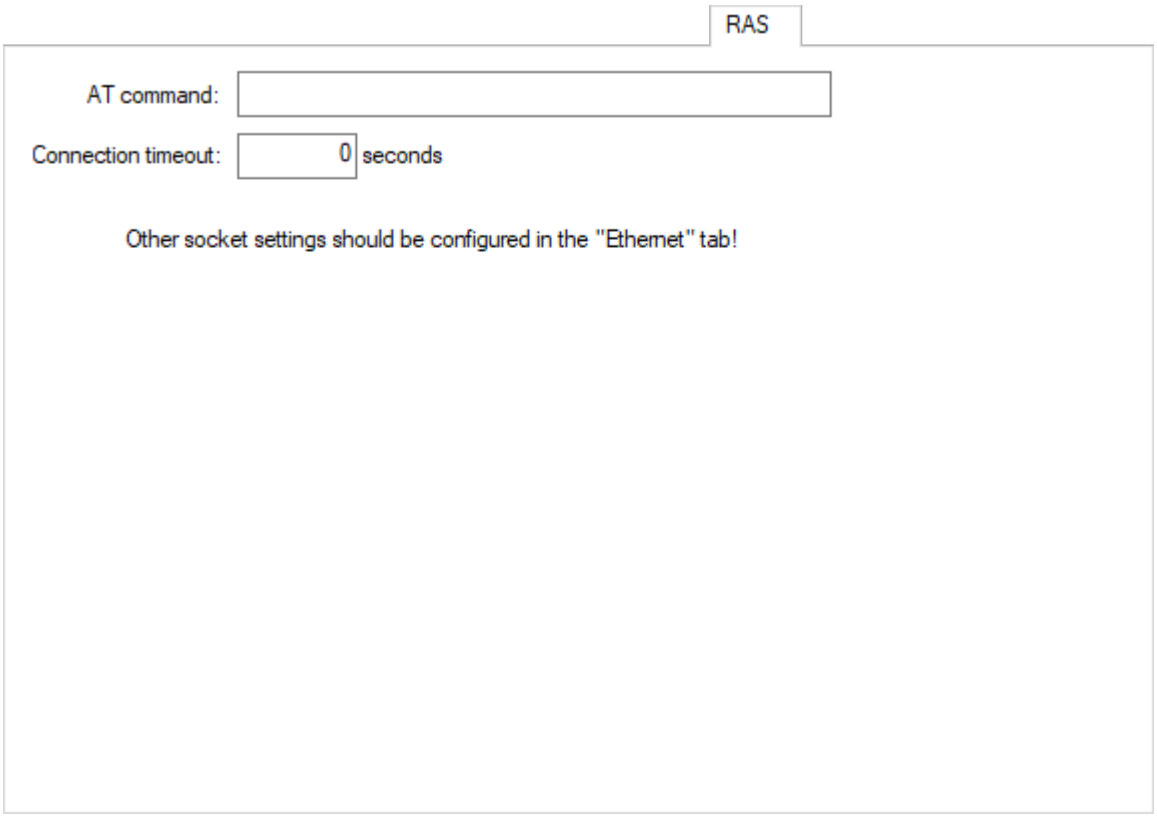
A **RAS** Interface opens a socket connection with a RAS device. A RAS device is a server of modems available through TCP/IP, waiting for socket connections on an IP port. For each connection accepted on this port, users have access to one modem.

When connecting to a RAS device, first the I/O Interface**IOKit** connects to the socket on the IP address and port configured on the **Ethernet** tab. After opening the socket, the following initialization or connection steps are performed:

1. Clears the socket, that is, removes any **TELNET** greeting message received from a RAS device.
2. Sends an **AT** dial message, in **ASCII** format, in the socket.
3. Waits for a **CONNECT** reply.
4. If the time-out expires, the connection is aborted.
5. If the **CONNECT** reply is received within the time-out, the socket is available for communication with a device, that is, the connection was established.

If step 5 (five) is successful, then the socket behaves as a normal socket, with the RAS device working as a router between a Driver and the device. Bytes sent by a Driver are received by the RAS device and sent to the destination device using a modem. Bytes received by the modem's RAS device are sent back to a Driver using the same socket.

After establishing a connection, the **RAS** interface monitors data received by a Driver. If a "NO CARRIER" **String** is found, the socket is closed. If the RAS device does not send a **NO CARRIER** signal, the **RAS** Interface cannot detect when the modem connection between the RAS device and the final I/O device fails. To recover from this failure, users are strongly advised to enable the **Disconnect if non-responsive** option on the **Setup** tab.



RAS tab

Available options on RAS tab

OPTION	DESCRIPTION
AT command	A **String** with the full **AT** command used to dial to a destination device. For example, "ATDT33313456" dials by tone to number "33313456"
Connection timeout	Number of seconds to wait for a modem's **CONNECT** reply, after sending an **AT** command

General Configurations

This section contains information about the configuration of general **I/O Tags** and **Properties** of I/O Interfaces.

I/O Tags

General I/O Interfaces Tags (N2/B2 = 0)

The Tags described next are provided for all supported I/O Interfaces.

IO.CommunicationStatus

Type of Tag	I/O Tag
Type of Access	Reading
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	6 (six)
String Configuration	IO.CommunicationStatus

This Tag informs the communication status of a Driver. It indicates how communication works relative to receiving valid data within a time period arbitrated in the configuration. For more information, please check topic **Setup Tab**. Possible values are **0 - Inactive communication**: The Driver did not receive valid data or stopped receiving data after *n* milliseconds, as configured in the properties window, or **1 - Active communication:** The Driver is receiving valid data.

IO.IOKitEvent

Type of Tag	Block Tag
Type of Access	Read-Only
B1 Parameter	-1 (minus one)
B2 Parameter	0 (zero)
B3 Parameter	0 (zero)
B4 Parameter	1 (one)
Size Property	4 (four)
ParamItem Property	IO.IOKitEvent

This Block returns Driver events generated by several sources in I/O Interfaces. The **TimeStamp** property of this Block represents the moment this event occurred. The Block Elements are the following:

- **Element 0**: Type of event. Possible values are **0**: Information, **1**: Warning, or **2**: Error
- **Element 1**: Source of an event. Possible values are **0**: Driver (specific of a Driver), **-1**: IOKit (generic events of I/O Interfaces), **-2**: **Serial** Interface, **-3**: **Modem** Interface, **-4**: **Ethernet** Interface, or **-5**: **RAS** Interface
- **Element 2**: Error number, specific for each source of event

- **Element 3**: Message of an event, a **String** specific for each event

NOTE

A Driver keeps a maximum number of 100 events internally. If additional events are reported, older events are discarded.

IO.PhysicalLayerStatus

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	2 (two)
String Configuration	IO.PhysicalLayerStatus

This Tag indicates the status of a physical layer. Possible values are the following:

- **0**: Physical layer stopped, that is, a Driver is in **Offline** mode, the physical layer failed when initializing, or exceeded the maximum number of reconnection attempts
- **1**: Physical layer started but not connected, that is, a Driver is in **Online** mode but the physical layer is not connected. If the **Connection management** option is configured with the value **Automatic**, the physical layer can be connecting, disconnecting, or waiting for a reconnection attempt. If the **Connection management** option is configured with the vlaue **Manual**, then the physical layer remains in this status until forced to connect
- **2**: Physical layer connected, that is, the physical layer is ready for use. This **DOES NOT** mean a device is connected, only that the access layer is working

IO.SetConfigurationParameters

Type of Tag	Block Tag
Type of Access	Read-Only
B1 Parameter	-1 (minus one)
B2 Parameter	0 (zero)
B3 Parameter	0 (zero)
B4 Parameter	3 (three)
Size Property	2 (two)
ParamItem Property	IO.SetConfigurationParameters

Use this Tag to change any property of a Driver's configuration dialog box at run time.

This Tag works only while a Driver is in **Offline** mode. To start a Driver in **Offline** mode, select the **Start driver OFFLINE** option on that Driver's configuration dialog box. Users can write to a PLC Tag or to a Block Tag containing the parameters to change. Writing individual Block Elements is not supported, the whole Block must be written at once.

In **Elipse SCADA**, users must use a Block Tag. Every parameter to configure uses two Block Elements. For example, if users want to configure 3 (three) parameters, then the size of the Block must be 6 (six, 3 × 2). The first Element is the property's name, as a **String**, and the second Element is the property's value, according to the next example.

```
// 'Block' must be a Block Tag with automatic reading,
// scan reading, and automatic writings disabled.
// Configure all parameters
Block.element001 = "IO.Type" // Parameter 1
Block.element002 = "Serial"
Block.element003 = "IO.Serial.Port" // Parameter 2
Block.element004 = 1
Block.element005 = "IO.Serial.BaudRate" // Parameter 3
Block.element006 = 19200
// Writes the whole Block
Block.Write()
```

When using **Elipse E3**, the ability to create arrays at run time allows using an I/O Tag as well as a Block Tag. Users can use the **Write** method of a Driver to send the parameters directly to that Driver, without creating a Tag, according to the next example.

```
Dim arr(6)
' Configure all array elements
arr(1) = "IO.Type"
arr(2) = "Serial"
arr(3) = "IO.Serial.Port"
arr(4) = 1
arr(5) = "IO.Serial.BaudRate"
arr(6) = 19200
' There are two methods to send parameters
' Method 1: Using an I/O Tag
tag.WriteEx arr
' Method 2: Without using a Tag
Driver.Write -1, 0, 0, 3, arr
```

A variation of the previous example uses a bidimensional array.

```
Dim arr(10)
' Configure all array elements. Notice the array was resized
' to 10 elements. Empty array elements are ignored by a Driver
arr(1) = Array("IO.Type", "Serial")
arr(2) = Array("IO.Serial.Port", 1)
arr(3) = Array("IO.Serial.BaudRate", 19200)
Driver.Write -1, 0, 0, 3, arr
```

A Driver does not validate parameter names or passed values, therefore be careful when writing parameters and values. The **Write** method fails if the configuration array is incorrectly created. Users can check the log of a Driver or use the *writeStatus* parameter of the **WriteEx** method to find out the exact cause of an error.

```
Dim arr(10), strError
arr(1) = Array("IO.Type", "Serial")
arr(2) = Array("IO.Serial.Port", 1)
arr(3) = Array("IO.Serial.BaudRate", 19200)
If Not Driver.WriteEx -1, 0, 0, 3, arr, , , strError Then
  MsgBox "Failed configuring Driver parameters: " + strError
End If
```

IO.WorkOnline

Type of Tag	I/O Tag
Type of Access	Reading or Writing
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	4 (four)
String Configuration	IO.WorkOnline

This Tag informs the current status of a Driver and allows starting or stopping the physical layer. Possible values are the following:

- **0 - Driver Offline**: Physical layer is closed or stopped. This mode allows a dynamic configuration of a Driver's parameters using the **IO.SetConfigurationParameters** Tag
- **1 - Driver Online**: Physical layer is open or executing. While in **Online** mode, the physical layer can be connected or disconnected and its current status can be checked using the **IO.PhysicalLayerStatus** Tag

In the next example, using **Elipse E3**, a Driver is configured to **Offline** mode, its COM port is changed, and then configured to **Online** mode again.

```
'Configure to Offline mode
Driver.Write -1, 0, 0, 4, 0
'Change port to COM2
Driver.Write -1, 0, 0, 3, Array("IO.Serial.Port", 2)
'Configure to Online mode
Driver.Write -1, 0, 0, 4, 1
```

The **Write** method may fail when configuring a Driver to **Online** mode, that is, writing the value 1 (one). In this case, this Driver remains in **Offline** mode. The cause of failure can be:

- Type of physical layer incorrectly configured, probably an invalid value was configured in the **IO.Type** property
- This Driver may have run out of memory
- Physical layer probably did not create its working thread. Search the log file for a message "Failed to create physical layer thread!"
- Physical layer could not start. The cause of this failure depends on the type of physical layer. It can be an invalid serial port number, a failure when starting Windows Sockets, or a failure when starting TAPI (modem), among others. This cause is recorded on the log file

IMPORTANT

Even if the configuration of a Driver to **Online** mode is successful, this does not necessarily mean the physical layer is ready to use, that is, ready to execute input and output operations with an external device. The **IO.PhysicalLayerStatus** Tag must be checked to ensure the physical layer is connected and ready for communication.

Properties

These are general properties of all supported I/O Interfaces.

IO.ConnectionMode

Controls the management mode of a Connection. Possible values are **0**: Automatic mode, in which a Driver manages the connection or **1**: Manual mode, in which an application manages the connection.

IO.GiveUpEnable

When configured to True, defines a maximum number of reconnection attempts. If all reconnection attempts fail, a Driver enters the **Offline** mode. When configured to False, a Driver tries until a reconnection is successful.

IO.GiveUpTries

Number of reconnection attempts before this one is aborted. For example, if the value of this property is equal to 1 (one), a Driver tries only one reconnection when the connection is lost. If this one fails, this Driver enters the **Offline** mode.

IO.InactivityEnable

Configure to True to enable and to False to disable inactivity detection. The physical layer is disconnected if inactive for a certain period of time. The physical layer is considered inactive only if it is capable of sending data but not capable of receiving it back.

IO.InactivityPeriodSec

Number of seconds to check for inactivity. If the physical layer is inactive for this period of time, it is then disconnected.

IO.RecoverEnable

Configure to True to enable a Driver to recover lost connections and to False to leave a Driver in **Offline** mode when a connection is lost.

IO.RecoverPeriodSec

Delay time between two connection attempts, in seconds.

NOTE

The first reconnection is executed immediately after a connection is lost.

IO.StartOffline

Configure to True to start a Driver in **Offline** mode and to False to start a Driver in **Online** mode.

NOTE

It is pointless to change this property at run time, as it can only be changed when a Driver is already in **Offline** mode. To configure a Driver in **Online** mode at run time, write the value 1 (one) to the **IO.WorkOnline** Tag.

IO.TimeoutMs

Defines a time-out for the physical layer, in milliseconds. One second is equal to 1000 milliseconds.

IO.Type

Defines the type of physical interface used by a Driver. Possible values are the following:

- **N or None**: Does not use a physical interface, that is, a Driver must provide a customized interface
- **S or Serial**: Uses a local serial port (COM*n*)
- **M or Modem**: Uses a local modem, internal or external, accessed via TAPI (*Telephony Application Programming Interface*)
- **E or Ethernet**: Uses a TCP/IP or UDP/IP socket
- **R or RAS**: Uses a **RAS** (*Remote Access Server*) Interface. A Driver connects to a RAS device using the **Ethernet** Interface and then sends an **AT** (*dial*) command

Statistical Configuration

This section contains information about the configuration of **I/O Tags** and **Properties** of I/O Interfaces statistics.

I/O Tags

Tags of I/O Interface Statistics (N2/B2 = 0)

The Tags described next display statistics for all I/O Interfaces.

IO.Stats.Partial.BytesRecv

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1101
Configuration by String	IO.Stats.Partial.BytesRecv

This Tag returns the number of bytes received in the current connection.

IO.Stats.Partial.BytesSent

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1100
Configuration by String	IO.Stats.Partial.BytesSent

This Tag returns the number of bytes sent through the current connection.

IO.Stats.Partial.TimeConnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1102
Configuration by String	IO.Stats.Partial.TimeConnectedSeconds

This Tag returns the number of seconds a Driver is connected in the current connection or 0 (zero) if a Driver is disconnected.

IO.Stats.Partial.TimeDisconnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1103
Configuration by String	IO.Stats.Partial.TimeDisconnectedSeconds

This Tag returns the number of seconds a Driver is disconnected since the last connection ended or 0 (zero) if a Driver is connected.

IO.Stats.Total.BytesRecv

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1001
Configuration by String	IO.Stats.Total.BytesRecv

This Tag returns the number of bytes received since a Driver was loaded.

IO.Stats.Total.BytesSent

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1000
Configuration by String	IO.Stats.Total.BytesSent

This Tag returns the number of bytes sent since a Driver was loaded.

IO.Stats.Total.ConnectionCount

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1004
Configuration by String	IO.Stats.Total.ConnectionCount

This Tag returns the number of connections a Driver already established, successfully, since it was loaded.

IO.Stats.Total.TimeConnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1002
Configuration by String	IO.Stats.Total.TimeConnectedSeconds

This Tag returns the number of seconds a Driver remained connected since it was loaded.

IO.Stats.Total.TimeDisconnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	0 (zero)
N4 Parameter	1003
Configuration by String	IO.Stats.Total.TimeDisconnectedSeconds

This Tag returns the number of seconds a Driver remained disconnected since it was loaded.

Properties

Currently, there are no properties defined specifically to display I/O Interface statistics at run time.

Ethernet Interface Configuration

This section contains information about the configuration of **I/O Tags** and **Properties** of an **Ethernet** Interface.

I/O Tags

Tags of an Ethernet Interface (N2/B2 = 4)

The Tags described next allow controlling and identifying an **Ethernet** Interface at run time and they are also valid when the **RAS** Interface is selected.

IMPORTANT

These Tags are available **ONLY** while a Driver is in **Online** mode.

IO.Ethernet.IPSelect

Type of Tag	I/O Tag
Type of Access	Reading or Writing
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	4 (four)
N4 Parameter	0 (zero)
String Configuration	IO.Ethernet.IPSelect

Indicates the active IP address. Possible values are **0**: The main IP address is selected, **1**: The first alternative or backup IP address is selected, **2**: The second alternative or backup IP address is selected, or **3**: The third alternative or backup IP address is selected.

If the **Ethernet** or **RAS** Interface is connected, this Tag indicates which one of the four configured IP addresses is in use. If the Interface is disconnected, this Tag indicates which IP address is used first on the next attempt to connect.

During the connection process, if the active IP address is not available, the I/O Interface tries to connect using the other IP address. If the connection with the alternative IP address works, it is configured as the active IP address (automatic switchover).

To force a manual switchover, write values from 0 (zero) to 3 (three) to this Tag. This forces a reconnection with the specified IP address (**0**: Main address or **1, 2, 3**: Alternative address) if a Driver is currently connected. If a Driver is disconnected, this Tag configures the active IP address for the next attempt to connect.

IO.Ethernet.IPSwitch

Type of Tag	I/O Tag
Type of Access	Write-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	4 (four)
N4 Parameter	1 (one)
String Configuration	IO.Ethernet.IPSwitch

Any value written to this Tag forces a manual switchover. If the main IP address is active, then the first alternative or backup IP address is activated, and so on for all alternative IP addresses and returning to the main address until a connection is established.

If a Driver is disconnected, this Tag configures the active IP address for the next attempt to connect.

IO.Ethernet.SocketState

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	4 (four)
N4 Parameter	2 (two)
String Configuration	IO.Ethernet.SocketState

The Value property of this Tag corresponds to socket states as a map of bits:

- **Bit 0**: 0 (zero, not listening) or 1 (one, listening)
- **Bit 1**: 0 (zero, disconnected) or 1 (one, connected)

Properties

These properties control the configuration of an **Ethernet** Interface.

NOTE

The **Ethernet** Interface is also used by the **RAS** Interface.

IO.Ethernet.AcceptConnection

Configure to False if a Driver must not accept external connections, that is, if a Driver behaves as a master, or configure to True to enable the reception of connections, that is, if a Driver behaves as a slave.

IO.Ethernet.BackupEnable[2,3]

Configure to True to enable an alternative or backup IP address. If the reconnection attempt with the main IP address fails, a Driver tries to use an alternative IP address. Configure to False to disable its usage.

IO.Ethernet.BackupIP[2,3]

Alternative or backup IP address of a remote device. Users can use a numerical address, as well as a device's host name, such as "192.168.0.7" or "SERVER2".

IO.Ethernet.BackupLocalPort[2,3]

Local port number to be used when connecting to an alternative IP address of a remote device. Used only if **IO.Ethernet.BackupLocalPortEnable** is equal to True.

IO.Ethernet.BackupLocalPortEnable[2,3]

Configure to True to force the use of a specific local port when connecting to an alternative or backup IP address or configure to False to use any available local port.

IO.Ethernet.BackupPort[2,3]

Port number of an alternative or backup IP address of a remote device, used with the **IO.Ethernet.BackupIP** property.

IO.Ethernet.IPFilter

List with a comma-separated IPv4 or IPv6 addresses, which defines from which addresses a Driver accepts or blocks connections. Users can use asterisks, such as "192.168.*.*", or intervals, such as "192.168.0.41-50", in any part of IP addresses. To block an IP address or a range of IP addresses, use the tilde ("~") character at the beginning of the address, according to the next examples:

- **192.168.0.24**: Accepts only connections from IPv4 address 192.168.0.24
- **192.168.0.41-50**: Accepts connections from IPv4 addresses in the interval between 192.168.0.41 and 192.168.0.50
- **192.168.0.***: Accepts connections from IPv4 addresses in the interval between 192.168.0.0 and 192.168.0.255
- **fe80:3bf:877::*:* (expands to fe80:03bf:0877:0000:0000:0000:*:*)**: Accepts connections from IPv6 addresses in the interval between fe80:03bf:0877:0000:0000:0000:0000:0000 and fe80:03bf:0877:0000:0000:0000:ffff:ffff
- **192.168.0.10, 192.168.0.15, 192.168.0.20**: Accepts connections from IPv4 addresses 192.168.0.10, 192.168.0.15, and 192.168.0.20
- **~192.168.0.95, 192.168.0.***: Accepts connections from IPv4 addresses in the interval between 192.168.0.0 and 192.168.0.255, except the IPv4 address 192.168.0.95

When a Driver receives a connection attempt, the list of filters is scanned sequentially from left to right, searching for a specific authorization or block for the IP address where the connection comes from. If no element on the list corresponds to the IP address, the authorization or block are dictated by the last element of that list:

- If the last element on the list is an authorization, such as "192.168.0.24", then all IP addresses not found on the list are blocked
- If the last element on the list is a block, such as "~192.168.0.24", then all IP addresses not found on the list are authorized

If an IP address appears on more than one filter on the list, the leftmost filter has precedence. For example, in case of "~192.168.0.95, 192.168.0.*", the IP address 192.168.0.95 fits both rules, but the rule that wins is the leftmost one, "~192.168.0.95", and therefore this IP address is blocked.

When **IOKit** blocks a connection, it logs a message "Blocked incoming socket connection from {IP}!".

In case of UDP connections in broadcast listening mode, in which a Driver can receive packets from different IP addresses, blocks or permissions are performed at each packet received. If a packet is received from a blocked IP address, it logs a message "Blocked incoming packet from {IP} (discarding {N} bytes)!".

IO.Ethernet.ListenIP

A IP address of the local network interface that a Driver uses to establish and accept connections. Leave this property empty to establish and accepts connections using any local network interface.

IO.Ethernet.ListenPort

9 Number of the IP port used by a Driver to listen to connections.

IO.Ethernet.MainIP

A IP address of a remote device. Users can use a numerical address, as well as a device's host name, such as "192.168.0.7" or "SERVER2".

IO.Ethernet.MainLocalPort

9 Local port number to use when connecting to the main IP address of a remote device. This value is only used if the **IO.Ethernet.MainLocalPortEnable** property is equal to True.

IO.Ethernet.MainLocalPortEnable

◩ Configure to True to force the use of a specific local port when connecting to the main IP address of a remote device or configure to False to use any available local port.

IO.Ethernet.MainPort

9 Number of the IP port of a remote device, used with the **IO.Ethernet.MainIP** property.

IO.Ethernet.PingEnable

◩ Configure to True to enable sending a **ping** command to the IP address of a remote device, before trying to connect to the socket. This socket's connection time-out cannot be controlled, therefore sending a **ping** command before connecting is a fast way to detect if the connection is going to fail. Configure to False to disable a **ping** command.

IO.Ethernet.PingTimeoutMs

9 Delay time to wait for a response from a **ping** command, in milliseconds.

IO.Ethernet.PingTries

9 Maximum number of attempts of a **ping** command. Minimum value is 1 (one), including the first **ping** command.

IO.Ethernet.ShareListenPort

◩ Configure to True to share a listening port with other Drivers and processes or False to open a listening port in exclusive mode. To successfully share a listening port, all Drivers and processes that use that port must open it in shared mode. When a listening port is shared, each incoming connection is distributed to one of the processes listening. This way, if a Slave Driver only supports one connection at a time, users can use several instances of this Driver listening on the same port, therefore simulating a Driver with support for multiple connections.

IO.Ethernet.SupressEcho

◩ Configure to True to eliminate echoes in communication. An echo is the unwanted reception of an exact copy of all data packets a Driver sent to a device.

IO.Ethernet.Transport

A Defines a transport protocol. Possible values are **T or TCP**: Uses the TCP/IP protocol or **U or UDP**: Uses the UDP/IP protocol.

IO.Ethernet.UseIPv6

◩ Configure to True to use IPv6 addresses on all Ethernet connections or configure to False to use IPv4 addresses (default).

Modem Interface Configuration

This section contains information about the configuration of **I/O Tags** and **Properties** of a **Modem** (TAPI) Interface.

I/O Tags

Tags of a Modem Interface (N2/B2 = 3)

The Tags described next allow controlling and diagnosing a **Modem** (TAPI) Interface at run time.

IMPORTANT
These Tags are available **ONLY** while a Driver is in **Online** mode.

IO.TAPI.ConnectionBaudRate

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	3 (three)
N4 Parameter	5 (five)
String Configuration	IO.TAPI.ConnectionBaudRate

Indicates a baud rate value for the current connection. If a modem is not connected, returns the value 0 (zero).

IO.TAPI.Dial

Type of Tag	I/O Tag
Type of Access	Write-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	3 (three)
N4 Parameter	1 (one)
String Configuration	IO.TAPI.Dial

Write any value to this Tag to force a **Modem** Interface to start a call. This is an asynchronous command, only starting the call process. Users can monitor the **IO.TAPI.IsModemConnected** Tag to detect when a call is established.

IO.TAPI.HangUp

Type of Tag	I/O Tag
Type of Access	Write-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	3 (three)
N4 Parameter	4 (four)
String Configuration	IO.TAPI.HangUp

Any value written to this Tag hangs the current call up.

NOTE

Use this command only when managing the physical layer manually or when explicitly trying to force a Driver to restart the communication. If the physical layer is configured for automatic reconnection, a Driver immediately tries to reestablish the connection.

IO.TAPI.IsModemConnected

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	3 (three)
N4 Parameter	3 (three)
String Configuration	IO.TAPI.IsModemConnected

This Tag indicates the status of a modem connection. Possible values are **0**: The modem is not connected, but it may be performing or receiving an external call or **1**: The modem is connected and a Driver completed or received an external call successfully. While it is in this status, the physical layer can send or receive data.

IO.TAPI.IsModemConnecting

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	3 (three)
N4 Parameter	6 (six)
String Configuration	IO.TAPI.IsModemConnecting

This Tag indicates the status of a modem connection, with more details than the **IO.TAPI.IsModemConnected** Tag. Possible values are **0**: Modem is not connected, **1**: Modem is connecting, that is, performing or receiving an external call, **2**: Modem is connected. While in this status, the physical layer can send or receive data, or **3**: Modem is disconnecting the current call.

IO.TAPI.ModemStatus

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	3 (three)
N4 Parameter	2 (two)
String Configuration	IO.TAPI.ModemStatus

Returns a **String** with the current status of a modem. Possible values are the following:

- **"No status!"**: The **Modem** Interface was not open yet or was already closed
- **"Modem initialized OK!"**: The **Modem** Interface was initialized successfully
- **"Modem error at initialization!"**: A Driver could not initialize modem's line. Check that Driver's log file for more details
- **"Modem error at dial!"**: A Driver could not start or accept a call
- **"Connecting..."**: A Driver started a call successfully, and is currently processing that call
- **"Ringing..."**: Indicates that the modem is receiving an external call, but it did not accepted it yet
- **"Connected!"**: A Driver connected successfully, that is, completed or accepted an external call
- **"Disconnecting..."**: A Driver is turning the current call off
- **"Disconnected OK!"**: A Driver turned the current call off
- **"Error: no dial tone!"**: A Driver aborted a call because the available line signal was not detected
- **"Error: busy!"**: A Driver aborted a call because the line was busy
- **"Error: no answer!"**: A Driver aborted a call because no answer was received from the other modem
- **"Error: unknown!"**: Current call was aborted because of an unknown error

IO.TAPI.PhoneNumber

Type of Tag	I/O Tag
Type of Access	Reading or Writing
N1 Parameter	-1 (minus one)
N2 Parameter	0 (zero)
N3 Parameter	3 (three)
N4 Parameter	0 (zero)
String Configuration	IO.TAPI.PhoneNumber

This Tag is a **String** that reads or changes the telephone number used by the **IO.TAPI.Dial** Tag. When changing this Tag, the new value is used only on the next **Dial** command.

Properties

These properties control the configuration of a **Modem** (TAPI) Interface.

IO.TAPI.AcceptIncoming

9 Configure to False if a modem cannot accept external calls, that is, if a Driver behaves as a master, and configure to True to enable receiving calls, that is, if a Driver behaves as a slave.

IO.TAPI.ModemID

9 This is the identification number of a modem. This ID is created by Windows and used internally to identify a modem on a list of devices installed on a computer. This ID may not remain valid if a modem is reinstalled or an application is executed on another computer.

NOTE

It is advisable to configure this property as 0 (zero), indicating that a Driver must use the first available modem.

IO.TAPI.PhoneNumber

A A telephone number used by **Dial** commands, such as "0w01234566", in which the "w" character forces a modem to wait for a call sign.

RAS Interface Configuration

This section contains information about the configuration of **I/O Tags** and **Properties** of a **RAS** Interface.

I/O Tags

Tags of a RAS Interface (N2/B2 = 5)

Currently, there are no Tags defined specifically to manage a **RAS** Interface at run time.

Properties

These properties control the configuration of a **RAS** Interface.

NOTE

A **RAS** Interface uses the **Ethernet** Interface, which therefore must be also configured.

IO.RAS.ATCommand

A An **AT** command to send through a socket to force a RAS device to perform a call using the current RAS channel, such as "ATDT6265545".

IO.RAS.CommandTimeoutSec

Time to wait for a **CONNECT** message in response to an **AT** command, in seconds.

Serial Interface Configuration

This section contains information about the configuration of **I/O Tags** and **Properties** of a **Serial** Interface.

I/O Tags

Tags of a Serial Interface (N2/B2 = 2)

Currently, there are no Tags defined specifically to manage a **Serial** Interface at run time.

Properties

These properties control the configuration of a **Serial** Interface.

IO.Serial.Baudrate

Specifies a baud rate of a serial port, such as 9600.

IO.Serial.CTSTimeoutMs

Time to wait for a **CTS** signal, in milliseconds. After turning the **RTS** signal on, a timer is started to wait for a **CTS** signal. If this timer expires, a Driver aborts sending bytes through the serial port. Available only when the **IO.Serial.RTS** property is configured with the value **Toggle** and the **IO.Serial.WaitCTS** property is configured to True.

IO.Serial.DataBits

Specifies the number of data bits to configure a serial port. Possible values are **5**: Five data bits, **6**: Six data bits, **7**: Seven data bits, or **8**: Eight data bits.

IO.Serial.DelayAfterMs

Number of milliseconds to delay after the last byte is sent through a serial port, but before turning the **RTS** signal off. Available only when the **IO.Serial.RTS** property is configured with the value **Toggle** and the **IO.Serial.WaitCTS** property is configured to False.

IO.Serial.DelayBeforeMs

Number of milliseconds to delay after turning the **RTS** signal on, but before data is sent. Available only when the **IO.Serial.RTS** property is configured with the value **Toggle** and the **IO.Serial.WaitCTS** property is configured to False.

IO.Serial.DTR

Indicates how a Driver deals with the **DTR** signal. Possible values are **OFF**: **DTR** signal is always turned off or **ON**: **DTR** signal is always turned on.

IO.Serial.InterbyteDelayUs

Delay time, in milliseconds (1/1000000 of a second), for each two bytes sent through a **Serial** Interface.

IO.Serial.InterframeDelayMs

Delay time, in milliseconds, before sending a packet after the last packet sent or received.

IO.Serial.Parity

Specifies a parity for the configuration of a serial port. Possible values are **E or Even**: Even parity, **N or None**: No parity, **O or Odd**: Odd parity, **M or Mark**: Mark parity, or **S or Space**: Space parity.

IO.Serial.Port

Number of the local serial port. Possible values are **1**: Uses the COM1 port, **2**: Uses the COM2 port, **3**: Uses the COM3 port, or **n**: Uses the COM*n* port.

IO.Serial.RTS

Indicates how a Driver deals with the **RTS** signal. Possible values are **OFF**: **RTS** signal always off, **ON**: **RTS** signal always on, or **Toggle**: Turns the **RTS** signal on when transmitting data and turns the **RTS** signal off when not transmitting data.

IO.Serial.StopBits

Specifies the number of stop bits for the configuration of a serial port. Possible values are **1**: One stop bit, **2**: One and a half stop bit, or **3**: Two stop bits.

IO.Serial.SupressEcho

Use a value different from 0 (zero) to enable suppressing the echo or 0 (zero) to disable it.

IO.Serial.WaitCTS

Configure to True to force a Driver to wait for the **CTS** signal before sending bytes when the **RTS** signal is turned on. Available only when the **IO.Serial.RTS** property is configured with the value **Toggle**.

Driver Revision History

VERSION	DATE	AUTHOR	COMMENTS
4.0.75	09/02/2025	M. Ludwig	• Driver updated to **IOKit** library version **3.0** and Visual Studio 2022 (*Case 37955*).
4.0.74	05/13/2025	M. Salvador	• 16 and 32-bit analog commands are now interpreted as **int16** and **int32**, respectively (*Case 37456*). • Fixed the behavior of data reception when using the **Unify Objects with/without status** option (*Case 37535*). • Created the **Discard Events Before this date**

VERSION	DATE	AUTHOR	COMMENTS
			(DD/MM/YY) option (*Case 37650*).
4.0.70	12/10/2024	M. Salvador	• Created an option to send Class 2 (two) or 3 (three) events more quickly, without asking for confirmation from an application, assuming that SOE events are reported only in Class 1 (one) (*Case 32987*). • Created an option for individual absolute dead band, in addition to the existing individual percentage option (*Case 33326*). • Now with the **Generate Events on First Write** option configured, an event is not generated in the first writing of a Tag if quality is bad (*Case 33375*). • Created a protection to avoid receiving invalid **Float**-type values (*Case 34437*). • Added support to reading requests with Variation 0 (zero) and Qualifier different from 6 (six) (*Case 36673*). • Created an option with the maximum number of days ahead of the actual time to accept new values. The lowest date limit is fixed at 01/01/1980 and fixed the handling of writing **NULL** values (*Case 33162*). • Now this Driver responds to requests for reading **Strings** with Qualifier 6 (six) and index not specified correctly (integrity of static objects) and responds with an error to requests with other Qualifiers (*Case 34739*). • Now this Driver can send **String**-type objects (110 ou 111) with an index greater than 255 (*Case 34774*).

VERSION	DATE	AUTHOR	COMMENTS
			• Improved performance to handle requests from a Master (*Case 35103*). • Revised UDP communications. Now this Driver responds to the IP address and TCP/IP port used by a Master in each connection (*Case 35375*). • Revised the maximum size of application messages, which is now correctly limited to the maximum user-declared size in the **Max App Size** property (*Case 35646*). • Fixed a problem when reporting events with an offset that accumulates at each additional connection (*Case 36655*).
4.0.50	06/22/2022	M. Salvador	• Now users can inform more than one DNP address in the **My DNP Addresses** option, separated by semicolons, sharing the same connection (*Case 30900*). • Now Object 2005 can be independent of Object 2001 using the **Unify objects with and without Status** option deselected (*Case 31397*). • Implemented Objects 3006 and 3206 (*Double Floating Point*) (*Case 31588*). • Added support to Object 100 Variation 1 (one) (*Case 31780*). • Created a Tag with the *N2* parameter equal to -42 (*force_events*), which allows forcing the generation of events generation with a specific type defined in the *N3* parameter. The *N4* parameter defines how timestamps are handled (*Case 30899*). • Now users can inform which Classes this Driver starts sending, informing in the

VERSION	DATE	AUTHOR	COMMENTS
			Use Unsolicited Msg option a sequence of 1 (one),2 (two) or 3 (three) numbers separated by colons, if needed. This configuration can be changed by a Master when sending **Enable** or **Disable unsolicited** commands (*Case 32331*). • Implemented the Objects 3006, 3206, and 3208 with 64 bits (*Case 32401*). • Created the **Disconnect after unconfirmed unsolicited sequence count** option, which allows informing the maximum number of consecutive unsolicited messages without a response, so that a Slave performs a forced disconnection (*Case 32403*). • Now deferred requests are responded correctly after receiving a confirmation or time-out (*Case 32177*). • Analog events without a timestamp are now sent in the sort order they were generated. In addition, if the **One Analog Event per Index** option is selected, then when a new event happens and a previous one is waiting to be sent, the value is updated but the original sort order is kept. Other events keep being sorted by timestamp, if used, or by index (*Case 32223*). • Now this Driver correctly waits for a time-out or a confirmation to send the next unsolicited message (*Case 32346*). • Improvements in the rules for sending unsolicited messages, to increase transfer rates (*Case 32838*).
4.0.38	05/18/2021	M. Salvador	• Added support to Qualifier 8 (eight) in event requests

VERSION	DATE	AUTHOR	COMMENTS
			from Classes 1 (one), 2 (two) and 3 (three) (*Case 29111*). • Implemented support to *Assign Classes* function (22) (*Case 29138*). • Implemented a response to reading requests of Object 80 variation 1 (one) (*Internal Indications*, indexes from zero to 15) (*Case 29192*). • Created an option to ignore the *Assign Class* function and defined a default *SoeType* without timestamp when receiving an *Assign Class* function and there is no *SoeType* in the Tag (the *N1* parameter) (*Case 29246*). • Created the **Unify objects with and without status** option, which is selected by default to keep compatibility with earlier versions (*Case 29516*). • Created a new SOE option equal to 3 (three), which generates 2 (two) events, with and without timestamp (*Case 29769*). • Implemented an Analog option with a **BitString** type (*Case 30602*). • Fixed functions 20 (*Enable Unsolicited*), 21 (*Disable Unsolicited*), and 24 (*Record Current Time*) (*Case 28900*). • Warm and Cold restart now start immediately after a response to a request (*Case 29206*). • Fixed the usage of individual dead band for **R32**- and **R64**-type Tags (*Case 29223*). • Added support to Variation 0 (zero) with Qualifiers other than 6 (six, unspecified) (*Case 29515*). • A Class 0 response now can be responded in the same application fragment with responses from other

VERSION	DATE	AUTHOR	COMMENTS
			requests in the same message (*Case 29380*). • Added support to requests for event Objects directly, with Variation 0 (zero) or a specific Variation (*Case 29615*). • Fixed a problem during a Class 0 (zero) request if a Slave does not have any Object created (*Case 29768*). • Created new options to publish events in the first connection and subsequent connections, so that a Master can receive before Class 0 (zero) the current Tag values with a timestamp relative to the last change (*Case 30111*).
4.0.18	08/10/2019	M. Salvador	• Added support for UDP/IP operations (*Case 27693*).
		C. Mello	• Driver ported to Visual Studio 2017 (*Case 27509*).
4.0.17	02/14/2019	M. Salvador	• Added support for a **Direct Operate No Ack** command in the Tag with the *N2* parameter equal to 6 (six) (*Case 26212*).
4.0.16	10/16/2018	M. Salvador	• Defined a default value 1 (one) for the number of DNP connections (*Case 25520*).
4.0.15	10/08/2018	M. Salvador	• Implemented an **Item** parameter "DBR: value" to specify a relative dead band (*Case 24431*). • Fixed the response for requests with a qualifier 27h, or 39 in decimal (*Case 24291*). • Added a new Element in the Control Block of commands to identify the Master that sent a command (*Case 22800*). • Added an option to disable start up via **Cold Restart** and **Warm Restart** commands (*Case 22328*).

VERSION	DATE	AUTHOR	COMMENTS
			• Fixed a problem with unsolicited messages after a disconnection (*Case 21757*). • Added support for multiple channels (*Case 21616*). • Fixed a rounding error in object 41 Variation 03 (*Case 21162*). • Implemented improvements on logs of Data Link layer (*Case 20168*). • Added an option to define a dead band individually by Tag (*Case 20161*). • Changed the dead band of analog points and counters to use decimal places (*Case 19337*). • Implemented an event for writing null values (*Case 19117*). • Improvements to the system to control and handle message sequencing (*Case 18783*). • Implemented callback functions for Block Tags (*Case 18782*).
4.0.1	02/20/2015	M. Salvador	• Removed command objects from Class 0 (zero) responses (*Case 18274*). • Added an option to generate events in the first writing of each point (*Case 18273*). • Improvements to command logs (*Case 18176*). • Implemented a Tag with the *N2* parameter equal to -30 to invalidate all existing Tags in a database (*Case 18175*). • Added options to define the stack of events for Classes 1 (one), 2 (two), and 3 (three) (*Case 18085*). • Improvements in the processing of unsolicited messages (*Case 17836*).

VERSION	DATE	AUTHOR	COMMENTS
			• Implemented support for callback functions (*Case 17835*). • Implemented a **String** data type (*Case 16921*). • Driver ported to **IOKit** library version **2.0** (*Case 15635*).
3.2.1	02/22/2013	M. Salvador	• Changed the protection identifier.
3.1.1	05/31/2012	M. Salvador	• **Beta 1, 2**: Changes on Single Block of Events. • **Beta 3**: Fixed the test on dead band. • **Beta 4**: Support for reading Object 50 Variation 1 (one) and fixed Qualifiers 7 (seven) and 8 (eight). • **Beta 6**: Implemented Object 1 (one) Variation 1 (one). • **Beta 11**: Application confirmation cannot be deselected, support for Class messages with Qualifier 7 (seven), support for Objects 1 (one) Variation 1 (one) Qualifier 6 (six) and Object 1 (one) Variation 0 (zero). • **Beta 12, 13**: Corrections in dead band. • **Beta 14, 15**: Sorting function in a response was inverting the order of points without a timestamp. • **Beta 16**: Fixed the **OnOff** Tag. • **Beta 17**: Limited the maximum size of Application layer to 2048 bytes. • **Beta 18**: Fixed the support for double points in Objects 3 (three) and 4 (four). • **Beta 20**: Implemented a **Cancel** command and checking on **Select over Select** command. • **Beta 22**: Fixed the writing of Object 10 Variation 1 (one).

VERSION	DATE	AUTHOR	COMMENTS
			• **Beta 23**: Fixed the propagation of quality when writing values to events. • **Beta 24**: Option to handle analog points when disconnected. • **Beta 25**: Option to wait for a **Reset of Remote Link**, which previously always waited. • **Beta 26**: **Sort Analog by Index** option. • **Beta 27**: The **Respond Link Status** option was using an incorrect code. • **Beta 28**: Data Link's *unacknowledged* messages were not handled. Different Object and Variation events on the same Class were not transmitted at the same time. • **Beta 29**: Option to limit Class messages to a Data Link fragment. • **Beta 30**: Support for event requests with Variation 0 (zero) and Qualifier 6 (six). • **Beta 31**: Support for timestamps converted to **GMT** (*Greenwich Mean Time*) format.
3.0.1	01/20/2010	M. Salvador	• Reimplemented the Application layer (*Case 8270*).
1.1.1	08/31/2004	M. Salvador	• All publications before revision control.

Headquarters
Rua Mostardeiro, 322/Cj. 902, 1001 e 1002
90510-002 — Porto Alegre — RS
Phone: (+55 51) 3346-4699
Fax: (+55 51) 3222-6226
E-mail: elipse-rs@elipse.com.br

Branch in Taiwan
9F., No.12, Beiping 2nd St., Sanmin Dist.
807 — Kaohsiung City — Taiwan
Phone: (+886 7) 323-8468
Fax: (+886 7) 323-9656
E-mail: evan@elipse.com.br

Check our website for information about a representative in your country.

www.elipse.com.br

kb.elipse.com.br
forum.elipse.com.br
www.youtube.com/elipsesoftware
elipse@elipse.com.br

Gartner 2014 Cool Vendor

OPC FOUNDATION MEMBER

Gartner, Cool Vendors in Brazil 2014, April 2014.
Gartner does not endorse any vendor, product or service depicted in its research publications, and does not advise technology users to select only those vendors with the highest ratings. Gartner research publications consist of the opinions of Gartner's research organization and should not be construed as statements of fact. Gartner disclaims all warranties, expressed or implied, with respect to this research, including any warranties of merchantability of fitness for a particular purpose.

Microsoft Partner
Gold Independent Software Vendor (ISV)