Elipse Drivers

IOKit User's Manual

Version 2.0.82 (04/27/2017)

Table of Contents

CHAPTER 1 Introduction

This Manual is a reference for users of Elipse Software's **IOKit**. It describes how to use and configure **IOKit** in **Elipse SCADA** as well as in **E3**, providing technical information about the practical usage of this component.

IOKit is a shared component used by Elipse Software's I/O Drivers, implementing a default access to the physical layer and providing interfaces for:

- Serial ports
- Modem (using TAPI)
- Ethernet (using WinSockets via TCP/IP or UDP/IP)
- RAS (*Remote Access Server*)

All Drivers written using **IOKit** use the resources described in this Manual, such as:

- Physical layer independence
- Generation of logs
- Offline configuration
- Management of connections

IOKit is implemented as a DLL (*Dynamic Link Library*) loaded with a Driver.

IMPORTANT

To avoid conflicts among Drivers, the IOKit.dll file must be present **ONLY** on folder **Windows\System32**. Copies of this DLL on other folders may cause incorrect behavior of a Driver or **IOKit**.

CHAPTER 2 Driver Configuration

IOKit configuration is performed on Driver's configuration dialog box. To access the configuration of this dialog box in **E3** (version 1.0), follow these steps:

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

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

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

Currently, **IOKit** 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 Driver for each serial port.

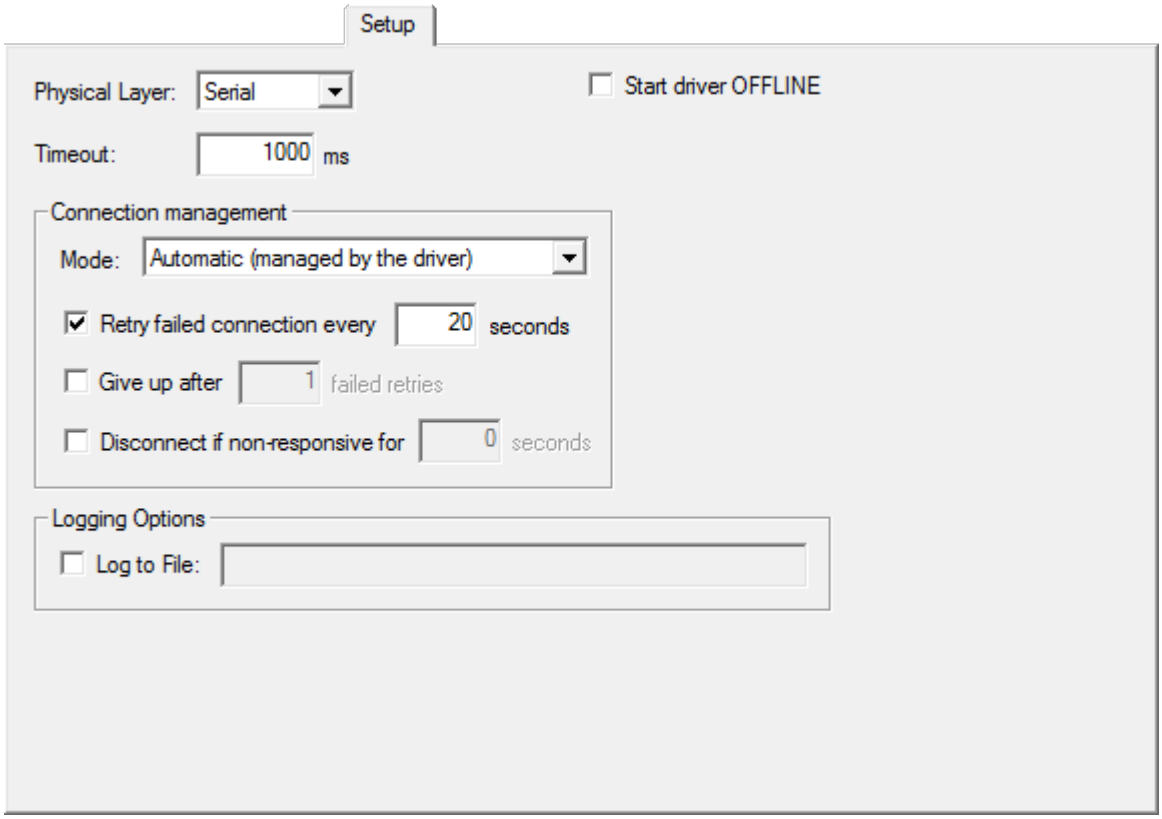
2.1 Configuration Dialog Box

The **IOKit** dialog box 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 each Driver) on the configuration dialog box. Check the Driver's Manual for more information about each specific tab.

2.2 Setup Tab

The **Setup** tab contains Driver's general configurations. This tab is divided into three distinct parts:

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



Setup tab

General options on Setup tab

OPTION	DESCRIPTION
Physical Layer	Select the physical layer on the 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 a byte (any byte from reception's buffer).
Start driver OFFLINE	Select this option so that the Driver starts in **Offline** mode (stopped). This means that the I/O interface is not created until this Driver is configured to **Online** mode (using a Tag in an application). This mode enables a dynamic configuration of an I/O interface at run time. Please check topic **Working Offline** for more details.

Options on Connection management group

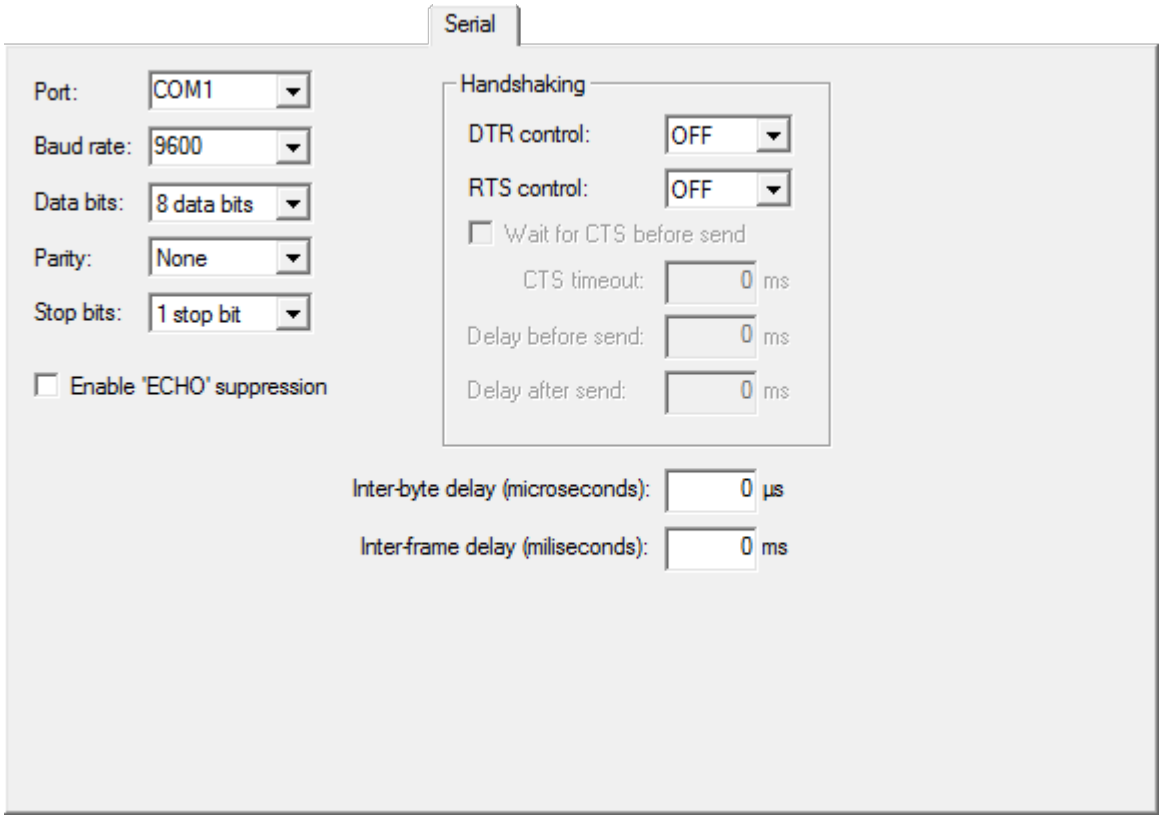
OPTION	DESCRIPTION
Mode	Selects a management mode of a connection. Selecting the **Automatic** option allows a Driver to manage the connection automatically, as specified in the next options. Selecting the **Manual** option allows an application to fully manage the connection. Please check topic **Driver Statuses** for more details.
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, the Driver keeps retrying until the 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, the 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 Logging Options group

OPTION	DESCRIPTION
Log to File	Enable this option and configure the name of the file to write the log. Log files can be large, so use this option for short periods of time, only for test and debugging purposes. If the **%PROCESS%** macro is used in the log file name, it is replaced by the ID of the current process. This option is particularly useful when using several instances of the same Driver in **E3**, thus allowing each instance to generate a separate log file. For example, when configuring this option as **c:\e3logs\drivers\sim_%PROCESS%.log**, a file **c:\e3logs\drivers\sim_00000FDA.log** is generated 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 as **c:\e3logs\drivers\sim_%DATE%.log**, a file **c:\e3logs\drivers\sim_2005_12_31.log** is generated in 12/31/2005 and a file **c:\e3logs\drivers\sim_2006_01_01.log** is generated in 01/01/2006.

2.3 Serial Tab

Use this tab to configure parameters of the **Serial** Interface.



Serial tab

General options on 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*** (for example, "COM15"). When typing a port's name manually, the dialog box only accepts port names 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 (for example, 600).
Data bits	Select 7 or 8 data bits on the list.
Parity	Select a parity on the list (**None**, **Even**, **Odd**, **Mark**, or **List**).
Stop bits	Select the number of stop bits on the list (**1**, **1.5**, or **2** stop bits).
Enable 'ECHO' supression	Enable this option to remove the echo received after **IOKit** sends data via serial port. If this echo is not equal to the bytes just sent, then **IOKit** aborts communication.
Inter-byte delay (microseconds)	Defines a delay between each byte transmitted by **IOKit**, in millionths of a second (1000000 is equal to a second). This option must be used with small delays (less than a millisecond).

OPTION	DESCRIPTION
Inter-frame delay (milliseconds)	Defines a delay between packets sent or received by **IOKit**, in thousandths of a second (1000 is equal to a second). This delay is applied if **IOKit** sends two consecutive packets, or between a received packet and the next sending.

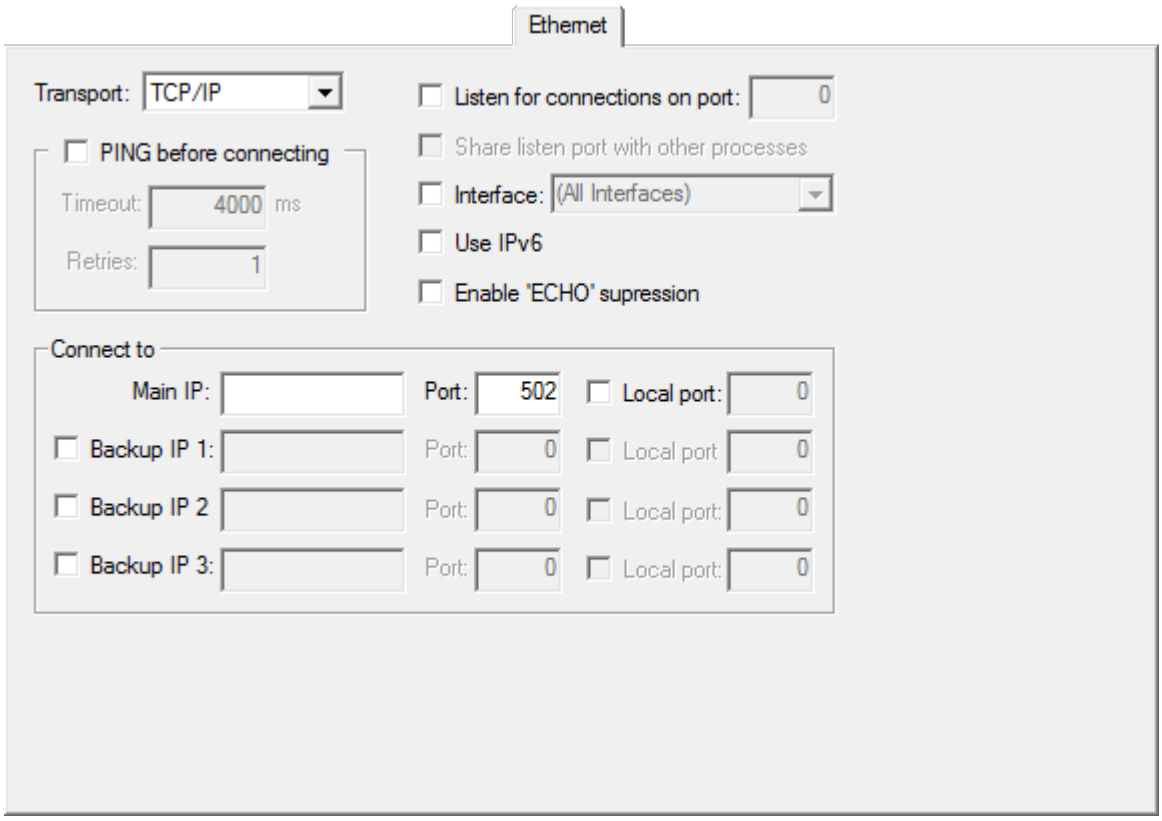
The **Handshaking** group configures the usage of **RTS**, **CTS**, and **DTR** signals in the handshaking process (controlling when data can be sent or received via serial line). Most of the time, configuring the **DTR control** option to **ON** and the **RTS control** option to **Toggle** works with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

Available options on Handshaking group

OPTION	DESCRIPTION
DTR control	Select **ON** to keep the **DTR** signal always on while the serial port is open. Select **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 **ON** to keep the **RTS** signal always on while the serial port is open. Select **OFF** to turn the **RTS** signal off while the serial port is open. Select **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 to **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, the Driver then fails the current communication and returns an error.
Delay before send	Some serial port hardware 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.

2.4 Ethernet Tab

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



Ethernet tab

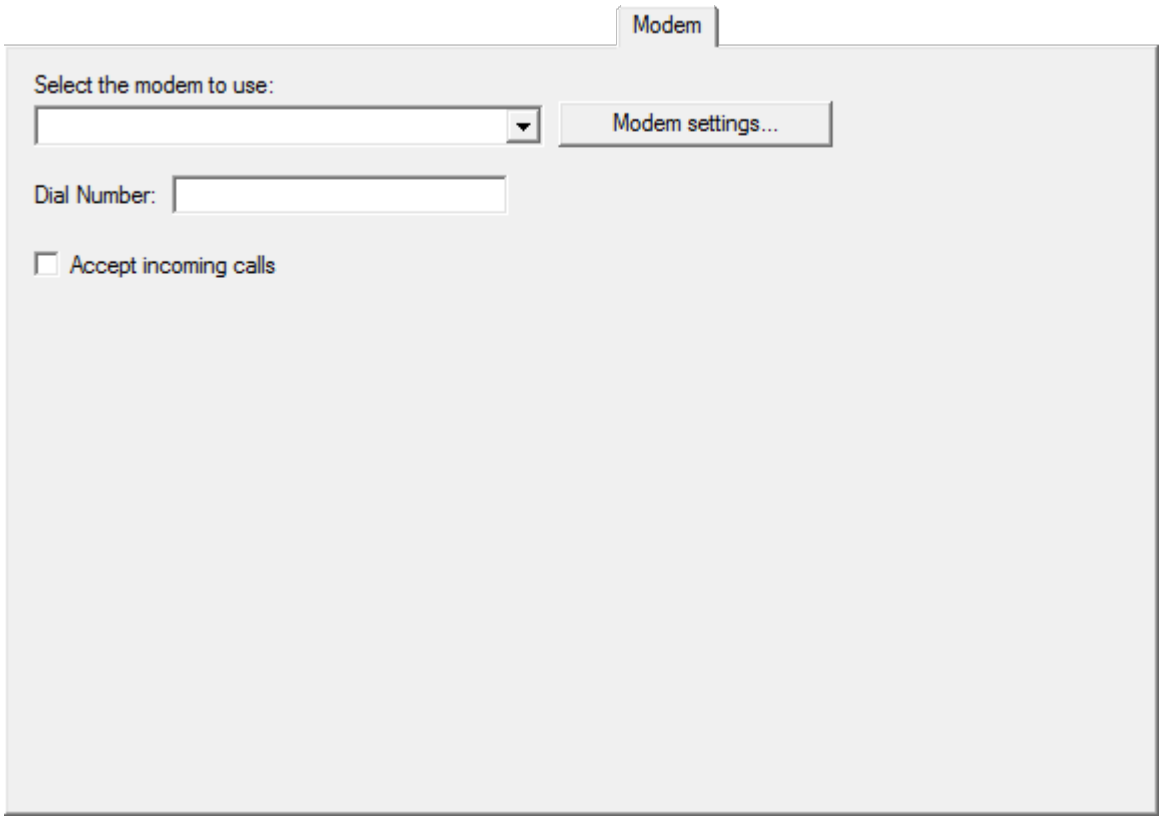
Available options on Ethernet tab

OPTION	DESCRIPTION
Transport	Select **TCP/IP** for a TCP socket (stream). Select **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, the Driver connects to the address and port specified in the **Connect to** option.
Connect to	These options configure an IP address and port of a remote device. • **IP**: Type an IP address of a remote device. It can be an IP address separated by dots as well as a URL (for a URL, the Driver uses the available DNS service to map that URL to an IP address). For example, "192.168.0.13" or "Server1" • **Port**: Type an IP port of a remote device (from 0 to 65535) • **Specify local port**: Select this option to use a fixed local port when connecting to the main address

OPTION	DESCRIPTION
Backup address	Enables a backup address if a device provides an alternative IP address (if the first address fails): • **IP**: Type an alternative IP address of a remote device. This can be an IP address separated by dots or a URL • **Port**: Type an alternative IP port of a remote device (from 0 to 65535) • **Specify local port**: Select this option to use a fixed local port when connecting to the backup address
PING before connecting	Enable this option to execute a **ping** command (check if a device can be reached on a network) for a device before trying a socket connection. This is a quick way 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): • **Timeout**: Specify the number of milliseconds to wait for a reply from the **ping** command. Users must use the **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 (between one and 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.
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.

2.5 Modem Tab

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



Modem tab

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

Available options on Modem tab

OPTION	DESCRIPTION
Select the modem to use	Select a modem on the list of modems available on the computer. If the **Default modem** option is selected, then the first available modem is used. Selecting this option is recommended specially when the 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 (waiting for the dial tone). Por exemplo, "0w33313456" (disca o número zero, espera e então disca o número "33313456").
Accept incoming calls	Enable this option so that the Driver answers the phone when receiving an external call. To use this option, users must configure the **Connection management** option on **Setup** tab to **Manual**.

2.6 RAS Tab

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

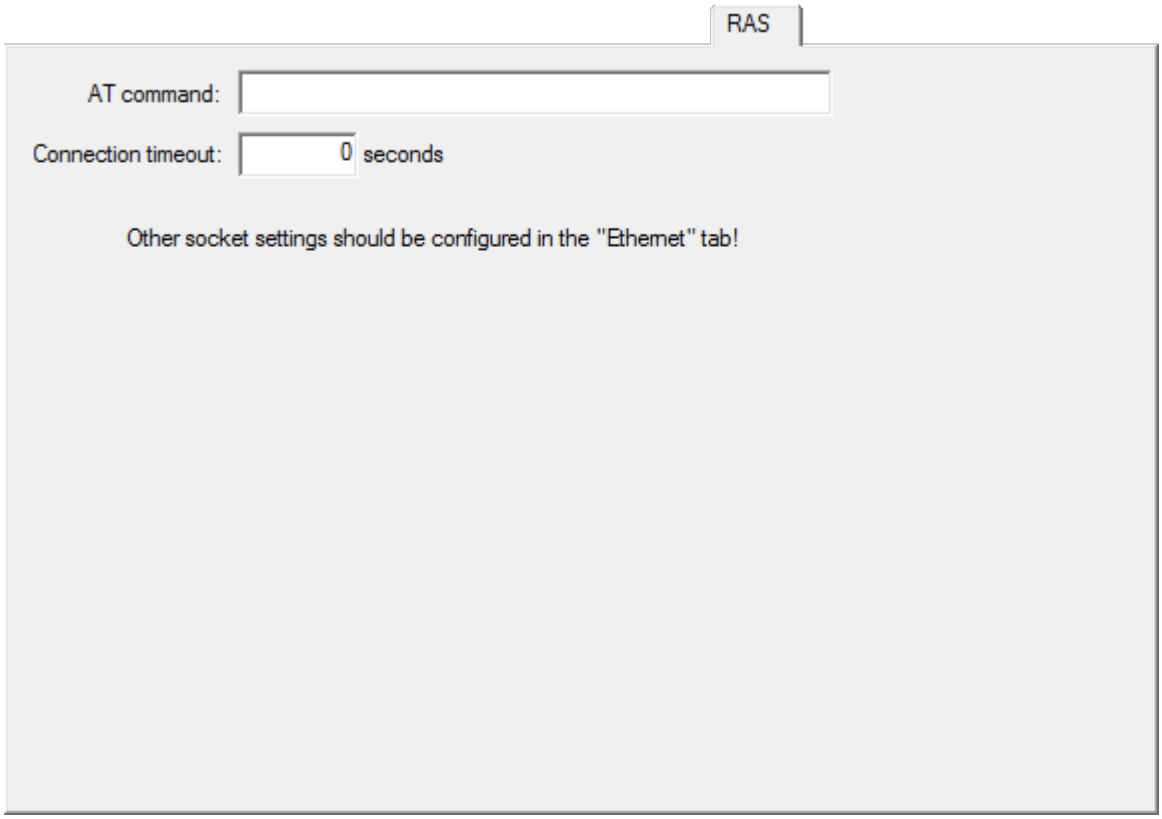
The **RAS** Interface opens a socket connection with a RAS device. A RAS device is a server of modems available through TCP/IP, waiting for socket connections on an IP port. For each connection accepted on this port, users have access to one modem.

When connecting to a RAS device, first **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. Clear the socket (remove any TELNET greeting message received from the RAS device).
2. Send an **AT** dial message (in ASCII) in the socket.
3. Wait 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 the device (connection was established).

If step 5 is successful, then the socket behaves as a normal socket, with the RAS device working as a router between the Driver and the device. Bytes sent by the 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 the Driver using the same socket.

After establishing the connection, the **RAS** interface monitors data received by the Driver. If a **String** "NO CARRIER" 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 **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" (tone dialing to number "33313456").
Connection timeout	Number of seconds to wait for a modem's **CONNECT** reply, after sending an **AT** command.

CHAPTER 3 General Configurations

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

3.1 I/O Tags

General IOKit Tags (N2/B2 = 0)

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

3.1.1 IO.IOKitEvent

Type of Tag	Block Tag
Type of Access	Read-Only
B1 Parameter	-1
B2 Parameter	0
B3 Parameter	0
B4 Parameter	1
Size Property	4
ParamItem Property	IO.IOKitEvent

This Block returns Driver events generated by several sources in **IOKit** (please check topic **IOKit Events** for a list with all events generated by **IOKit**). The **TimeStamp** property of this Block represents the moment this event occurred. The Block Elements are the following:

- **Element 0**: Type of event
  - **0**: Information
  - **1**: Warning
  - **2**: Error
- **Element 1**: Source of event
  - **0**: Driver (specific of a Driver)
  - **-1**: IOKit (generic events of **IOKit**)
  - **-2**: **Serial** Interface
  - **-3**: **Modem** Interface
  - **-4**: **Ethernet** Interface
  - **-5**: **RAS** Interface
- **Element 2**: Error number (specific for each source of event)
- **Element 3**: Event message (**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.

3.1.2 IO.PhysicalLayerStatus

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	0
N4 Parameter	2
String Configuration	IO.PhysicalLayerStatus

This Tag indicates the status of the physical layer. Its possible values are the following:

- **0**: Physical layer stopped (the 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 (the Driver is in **Online** mode, but the physical layer is not connected. If the **Connection management** option is configured as **Automatic**, the physical layer can be connecting, disconnecting, or waiting for a reconnection attempt. If the **Connection management** option is configured as **Manual**, then the physical layer remains in this status until forced to connect)
- **2**: Physical layer connected (the physical layer is ready for use). This **DOES NOT** mean the device is connected, only the access layer is working

3.1.3 IO.SetConfigurationParameters

Type of Tag	Block Tag
Type of Access	Read-Only
B1 Parameter	-1
B2 Parameter	0
B3 Parameter	0
B4 Parameter	3
Size Property	2
ParamItem Property	IO.SetConfigurationParameters

Use this Tag to change any property of Driver's configuration dialog box at run time (the complete list of properties can be found on the specific topic of each Interface).

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 Driver's configuration dialog box. Users can write to a PLC Tag or to a Block Tag containing the parameters to change (writings of individual Block Elements are 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 three parameters, then the size of the Block must be 6 (3 * 2). The first Element is the property's name (as a **String**) and the second Element is the property's value. Check this script in **Elipse SCADA**:

```
// 'Block' must be a Block Tag with automatic reading,
// scan reading, and automatic writing 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 **E3**, the ability to create arrays at run time allows using an I/O Tag as well as a Block Tag. Users can use Driver's **Write** method to send all parameters to the Driver, without creating a Tag. Check these examples:

```
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 elements of the array are ignored by the 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 Driver's log or use the *writeStatus* parameter of the **WriteEx** method to find out the exact cause of the 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 "Failure when configuring Driver parameters: " + strError
End If
```

3.1.4 IO.WorkOnline

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

This Tag informs Driver's current status and allows starting or stopping the physical layer.

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

In the next example (using **E3**), the Driver is configured to **Offline** mode, its COM port is changed, and then configured to **Online** mode again:

```
' Configure to Driver 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 Driver to Online mode
Driver.Write -1, 0, 0, 4, 1
```

The **Write** method can fail when configuring the Driver to **Online** mode (writing the value one). In this case, the 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)
- Driver may have run out of memory
- Physical layer probably did not create its working thread (search the log file for the message "Failed to create physical layer thread!")
- Physical layer could not start. The cause of failure depends on the type of physical layer. It can be an invalid serial port number, failure when starting Windows Sockets, failure when starting TAPI (modem), etc. 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 (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.

3.2 Properties

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

3.2.1 IO.ConnectionMode

9 Controls the management mode of the Connection:

- **0**: Automatic mode (the Driver manages the connection)
- **1**: Manual mode (the application manages the connection)

3.2.2 IO.GiveUpEnable

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

3.2.3 IO.GiveUpTries

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

3.2.4 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.

3.2.5 IO.InactivityPeriodSec

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

3.2.6 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.

3.2.7 IO.RecoverPeriodSec

9 Delay time between two connection attempts, in seconds.

NOTE

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

3.2.8 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.

3.2.9 IO.TimeoutMs

9 Defines a time-out for the physical layer, in milliseconds (one second is equal to 1000 milliseconds).

3.2.10 IO.Type

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

- **N or None**: Does not use a physical interface (the 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. The Driver connects to a RAS device using the **Ethernet** Interface and then sends an **AT** (*dial*) command

CHAPTER 4

Statistical Configuration

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

4.1 I/O Tags

Tags of IOKit statistics (N2/B2 = 0)

The Tags described next display statistics for **IOKit**.

4.1.1 IO.Stats.Partial.BytesRecv

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

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

4.1.2 IO.Stats.Partial.BytesSent

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

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

4.1.3 IO.Stats.Partial.TimeConnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	0
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.

4.1.4 IO.Stats.Partial.TimeDisconnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	0
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.

4.1.5 IO.Stats.Total.BytesRecv

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

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

4.1.6 IO.Stats.Total.BytesSent

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

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

4.1.7 IO.Stats.Total.ConnectionCount

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	0
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.

4.1.8 IO.Stats.Total.TimeConnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	0
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.

4.1.9 IO.Stats.Total.TimeDisconnectedSeconds

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	0
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.

4.2 Properties

Currently, there are no properties defined specifically to display **IOKit** statistics at run time.

CHAPTER 5 Serial Interface Configuration

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

5.1 I/O Tags

Tags of Serial Interface (N2/B2 = 2)

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

5.2 Properties

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

5.2.1 IO.Serial.Baudrate

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

5.2.2 IO.Serial.CTSTimeoutMs

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

5.2.3 IO.Serial.DataBits

9 Specifies the number of data bits to configure the serial port. Possible values are the following:

- **5**: Five data bits
- **6**: Six data bits
- **7**: Seven data bits
- **8**: Eight data bits

5.2.4 IO.Serial.DelayAfterMs

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

5.2.5 IO.Serial.DelayBeforeMs

9 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 to **Toggle** and the **IO.Serial.WaitCTS** property is configured to False.

5.2.6 IO.Serial.DTR

A Indicates how a Driver deals with the **DTR** signal:

- **OFF**: **DTR** signal is always turned off
- **ON**: **DTR** signal is always turned on

5.2.7 IO.Serial.InterbyteDelayUs

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

5.2.8 IO.Serial.InterframeDelayMs

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

5.2.9 IO.Serial.Parity

A Specifies a parity for the configuration of the serial port. Possible values are the following:

- **E or Even**: Even parity
- **N or None**: No parity
- **O or Odd**: Odd parity
- **M or Mark**: Mark parity
- **S or Space**: Space parity

5.2.10 IO.Serial.Port

9 Number of the local serial port:

- **1**: Uses the COM1 port
- **2**: Uses the COM2 port
- **3**: Uses the COM3 port
- **n**: Uses the COM*n* port

5.2.11 IO.Serial.RTS

A Indicates how a Driver deals with the **RTS** signal:

- **OFF**: **RTS** signal always off
- **ON**: **RTS** signal always on
- **Toggle**: Turns the **RTS** signal on when transmitting data and turns the **RTS** signal off when not transmitting data

5.2.12 IO.Serial.StopBits

9 Specifies the number of stop bits for the configuration of the serial port. Possible values are the following:

- **1**: One stop bit
- **2**: One and a half stop bit
- **3**: Two stop bits

5.2.13 IO.Serial.SupressEcho

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

5.2.14 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 to **Toggle**.

CHAPTER 6 Ethernet Interface Configuration

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

6.1 I/O Tags

Tags of Ethernet Interface (N2/B2 = 4)

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

IMPORTANT

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

6.1.1 IO.Ethernet.IPSelect

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

Indicates the active IP address. Possible values are the following:

- **0**: The main IP address is selected
- **1**: The alternative (backup) IP address is selected

If the **Ethernet** (or **RAS**) Interface is connected, this Tag indicates which of the two IP addresses configured is in use. If the Interface is disconnected, this Tag indicates which IP address is used first on the next connection attempt.

During the connection process, if the active IP address is not available, **IOKit** tries to connect using the other IP address. If the connection with the alternative IP address works, this one is configured as the active IP address (automatic switchover).

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

6.1.2 IO.Ethernet.IPSwitch

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

Any value written to this Tag forces a manual switchover. If the main IP address is active, then the alternative (backup) IP address is activated, and vice versa. This forces a reconnection with the specified IP address if the Driver is currently connected. If the Driver is disconnected, this Tag configures the active IP address for the next connection attempt.

6.2 Properties

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

NOTE

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

6.2.1 IO.Ethernet.AcceptConnection

◩ Configure to False if the Driver must not accept external connections (the Driver behaves as a master) or configure to True to enable the reception of connections (the Driver behaves as a slave).

6.2.2 IO.Ethernet.BackupEnable

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

6.2.3 IO.Ethernet.BackupIP

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

6.2.4 IO.Ethernet.BackupPort

9 Port number of the alternative IP address of the destination device (used with the **IO.Ethernet.BackupIP** property).

6.2.5 IO.Ethernet.ListenPort

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

6.2.6 IO.Ethernet.MainIP

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

6.2.7 IO.Ethernet.MainPort

9 Number of the IP port on the destination device (used with the **IO.Ethernet.MainIP** property).

6.2.8 IO.Ethernet.PingEnable

Configure to True to enable sending a **ping** command to the IP address of the destination 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.

6.2.9 IO.Ethernet.PingTimeoutMs

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

6.2.10 IO.Ethernet.PingTries

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

6.2.11 IO.Ethernet.Transport

Defines a transport protocol. Possible values are the following:

- **T or TCP**: Uses the TCP/IP protocol
- **U or UDP**: Uses the UDP/IP protocol

CHAPTER 7

Modem Interface Configuration

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

7.1 I/O Tags

Tags of Modem Interface (N2/B2 = 3)

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

IMPORTANT

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

7.1.1 IO.TAPI.ConnectionBaudRate

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	3
N4 Parameter	5
String Configuration	IO.TAPI.ConnectionBaudRate

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

7.1.2 IO.TAPI.Dial

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

Write any value to this Tag to force the **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.

7.1.3 IO.TAPI.HangUp

Type of Tag	I/O Tag
Type of Access	Write-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	3
N4 Parameter	4
String Configuration	IO.TAPI.HangUp

Any value written to this Tag turns the current call off.

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, the Driver immediately tries to reestablish the connection.

7.1.4 IO.TAPI.IsModemConnected

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

This Tag indicates modem's connection status. Possible values are the following:

- **0**: The modem is not connected, but it may be performing or receiving an external call
- **1**: The modem is connected and the Driver completed or received an external call successfully. While it is in this status, the physical layer can send or receive data

7.1.5 IO.TAPI.IsModemConnecting

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	3
N4 Parameter	6
String Configuration	IO.TAPI.IsModemConnecting

This Tag indicates the connection status of a modem, with more details than the **IO.TAPI.IsModemConnected** Tag. Possible values are the following:

- **0**: Modem is not connected
- **1**: Modem is connecting (performing or receiving an external call)
- **2**: Modem is connected. While in this status, the physical layer can send or receive data
- **3**: Modem is disconnecting the current call

7.1.6 IO.TAPI.ModemStatus

Type of Tag	I/O Tag
Type of Access	Read-Only
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	3
N4 Parameter	2
String Configuration	IO.TAPI.ModemStatus

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

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

7.1.7 IO.TAPI.PhoneNumber

Type of Tag	I/O Tag
Type of Access	Reading or Writing
N1 Parameter	-1
N2 Parameter	0
N3 Parameter	3
N4 Parameter	0
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.

7.2 Properties

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

7.2.1 IO.TAPI.AcceptIncoming

9 Configure to False if the modem cannot accept external calls (the Driver behaves as a master) and configure to True to enable receiving calls (the Driver behaves as a slave).

7.2.2 IO.TAPI.ModemID

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

NOTE

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

7.2.3 IO.TAPI.PhoneNumber

A The telephone number used by **Dial** commands. For example, "0w01234566" (the "w" character forces the modem to wait for a call signal).

CHAPTER 8 RAS Interface Configuration

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

8.1 I/O Tags

Tags of RAS Interface (N2/B2 = 5)

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

8.2 Properties

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

NOTE

The **RAS** Interface uses the **Ethernet** Interface, which for this reason must be also configured.

8.2.1 IO.RAS.ATCommand

A **AT** command to send through a socket to force a RAS device to perform a call using the current RAS channel. Example: "ATDT6265545".

8.2.2 IO.RAS.CommandTimeoutSec

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

CHAPTER 9 IOKit Events

This Appendix lists all events currently reported by **IOKit**. To access these events, users must declare the **IO.IOKitEvent** Block Tag (*B1* = -1, *B2* = 0, *B3* = 0, *B4* = -1, and *Size* = 4).

Events reported by IOKit

TYPE	SOURCE	CODE	MESSAGE
Error (2)	Serial (-2)	0	Error opening serial port: %s • Tried to open port COM%u twice • Failed to open port %s (Windows error %u) • Failed to configure port %s (Windows error %u)
Error (2)	Modem (-3)	0	Error opening modem line: %s • Tried to open line twice • Failed to initialize line (TAPI error %s) • Failed to negotiate API version (TAPI error %s) • Failed to open line (TAPI error %s) • Failed to configure status messages (TAPI error %s) • Failed to get line address status (TAPI error %s) • Out of memory getting line address status • Failed to get device capabilities (TAPI error %s) • Device ID = %u not found • Out of memory getting device capabilities
Info (0)	Modem (-3)	2	Beginning of the list of available modems
Info (0)	Modem (-3)	3	%u:%s • %u is the modem ID (decimal) • %s is the modem description
Info (0)	Modem (-3)	4	End of the list of available modems

TYPE	SOURCE	CODE	MESSAGE
Error (2)	Ethernet (-4)	0	Error connecting to %s on port %u: %s Error connecting to %s on port %u (backup IP): %s • Tried to connect socket twice • Null address • Socket error %s(%d) calling %s()
Error (2)	RAS (-5)	1	Time-out waiting for CONNECT
Info (0)	RAS (-5)	3	RAS response: '%s'
Error (2)	RAS (-5)	4	RAS connection error: NO CARRIER
Info (0)	IOKit (-1)	1	Initializing physical layer...
Error (2)	IOKit (-1)	2	Physical layer initialization failed!
Info (0)	IOKit (-1)	3	Physical layer initialized!
Info (0)	IOKit (-1)	4	Connecting physical layer...
Error (2)	IOKit (-1)	5	Failed to connect physical layer!
Info (0)	IOKit (-1)	6	Physical layer connected!
Error (2)	IOKit (-1)	7	Physical layer connection lost!
Info (0)	IOKit (-1)	8	Reconnecting physical layer...
Error (2)	IOKit (-1)	9	Failed to reconnect physical layer!
Info (0)	IOKit (-1)	10	Physical layer reconnected!
Info (0)	IOKit (-1)	11	Physical layer connected automatically!
Info (0)	IOKit (-1)	12	Reconnecting physical layer (retry #%u)...
Error (2)	IOKit (-1)	13	Failed to reconnect physical layer (retry #%u)!
Info (0)	IOKit (-1)	14	Physical layer reconnected (on retry #%u)!
Info (0)	IOKit (-1)	15	Terminating physical layer...
Info (0)	IOKit (-1)	16	Physical layer terminated! (%u bytes sent, %u bytes received)
Info (0)	IOKit (-1)	17	Disconnecting physical layer...
Error (2)	IOKit (-1)	18	Physical layer aborted (exhausted all retries)!
Error (2)	IOKit (-1)	19	Physical layer non-responsive for %u seconds, disconnecting...

CHAPTER 10 Advanced Topics

This chapter contains in-depth information about the internal functionality of **IOKit**.

10.1 Driver Statuses

There are four distinct Driver statuses:

- **Stopped**
- **Offline** (started)
- **Disconnected** (started, online)
- **Connected** (started, online)

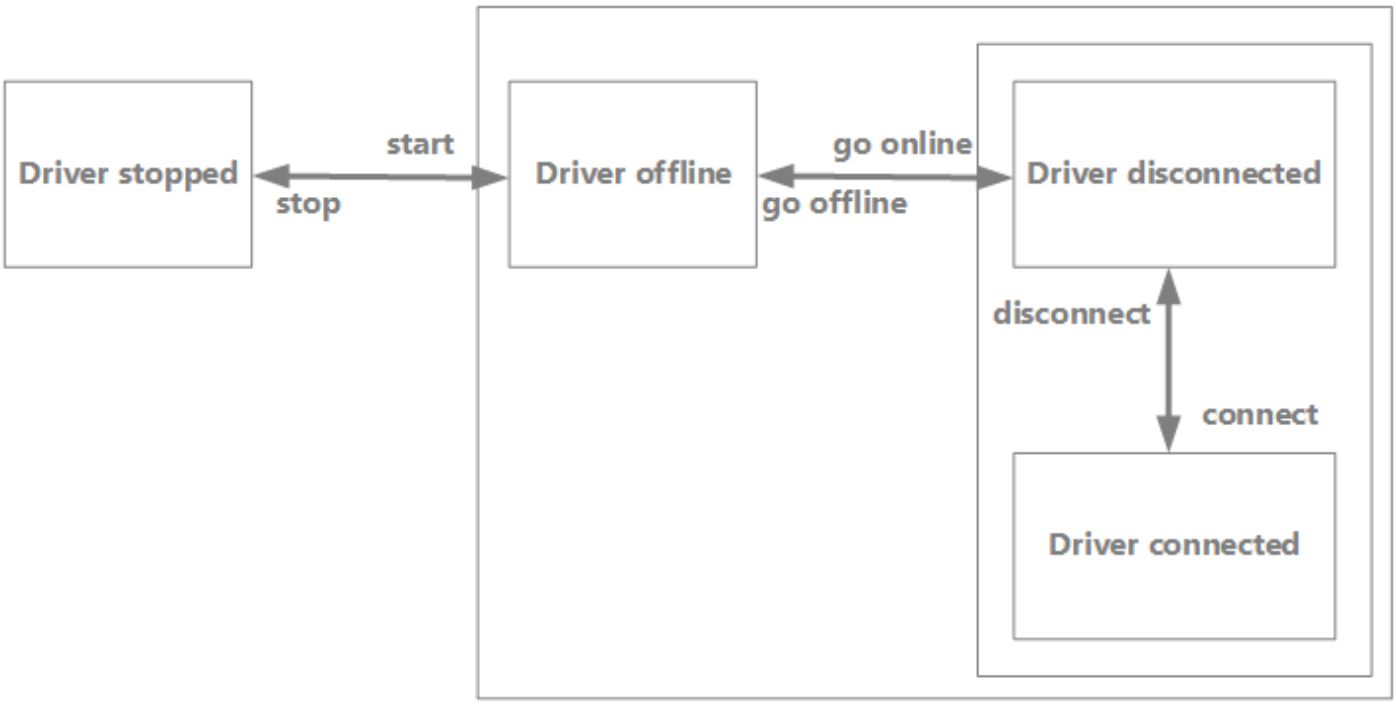
The **Stopped** status happens when a Driver is not executing or loaded, usually during application configuration.

As soon as the application is executed, the Driver enters the Offline status. In this status, the minimal conditions for execution are detected (basically, Driver's DLL is loaded), and the Driver waits for more parameter configuration or for a command to enter the **Online** mode.

The Online status is presented in two different ways, **Connected** or **Disconnected**. A Driver is **Disconnected** when the physical layer still cannot transmit or receive data. A Driver is **Connected** when the physical layer is ready to use.

NOTE

All statuses defined in this topic apply only to the physical layer. When the physical layer is connected, the Driver can access an I/O device (PLC), but that PLC may still not respond. Connecting the physical layer is a good starting point, but always check specific Driver Tags that report the status of a device.



Statuses of an IOKit Driver

10.2 Working Offline

The **Offline** mode was designed specially for applications with Drivers that must be configured at run time. Prefer this mode when the type of connection is ignored, or when not all connection parameters are known before executing the application.

While in **Offline** mode, all Driver's I/O Tags fail, in all readings and writings. The only allowed Tags are **IOKit General Tags** (**N1/B1 = -1** and **N2/B2 = 0**).

Although users can change statuses **Online** and **Offline** at any time, usually these procedures are followed:

1. Configure the **Start driver OFFLINE** option on Driver's Extras dialog box. This option allows a Driver to start in **Offline** mode.

2. Configure Driver's parameters in the application script. This script can be executed automatically or as a reply to a user's interface command.

3. Configure the Driver to the **Online** mode.

10.3 Connection Management

When a Driver is configured in **Online** mode, it enters a status where it tries to connect to the physical layer. This connection can be performed in three different ways:

- **Automatic Mode**: The Driver manages the connection according to the configuration of Driver's dialog box
- **Manual Mode**: The Driver remains disconnected and depends on application commands (such as a dial command, for example) to connect to a physical layer
- **Listen Mode**: The Driver remains disconnected and behaves as a slave, accepting connections requested by other devices on the physical line

This section also covers **Inactivity Detection**, which automatically disconnects the physical layer if it does not receive any byte within a specified period of time.

10.3.1 Automatic Mode

The **Automatic** mode can be selected in two ways:

- On Driver's configuration dialog box, by selecting the **Automatic (managed by the driver)** option as the connection mode on **Setup** tab
- Configuring the **ConnectionMode** property as 0 (zero, automatic)

The algorithm for automatic connection starts when the Driver is configured in **Online** mode. At this time, the Driver establishes an initial connection. From this time on, the Driver starts to behavior according to the options of connection management.

If connection recovery is enabled (the **Retry failed connection every ... seconds** option on **Setup** tab), the Driver automatically tries to reconnect to the physical layer if the connection is lost. If connection recovery is disabled, the Driver returns to the **Offline** mode.

Users can also define a maximum number of connection retries (the **Give up after ... failed connections** option on **Setup** tab). This limits the number of reconnections to the physical layer. If the maximum number of retries is reached, the Driver is stopped, returning to the **Offline** mode again. The connection counter is restarted (configured to zero) when the connection is successful.

10.3.2 Manual Mode

When a Driver is configured to work in **Manual** mode, it remains in the disconnected status after configured in **Online** mode. After that configuration, users can force it to connect to the physical layer using Tags for connection management.

For example, when using the **Modem** physical layer, there is a Screen where users can edit the telephone number (using a SetPoint) and a button to start the call. If the call completes, the Driver automatically changes to the **Connected** status. Users can disconnect the Driver again using the **IO.TAPI.HangUp** Tag.

10.3.3 Listen Mode

The **Listen** mode is almost identical to the **Manual** mode. The only difference is that the physical layer is programmed to accept external connections. For the **Ethernet** layer, check the **Listen for connections on port** option. For the **Modem** layer, enable the **Accept incoming calls** option.

For the **Listen** mode to work, users must configure the Driver to work in **Manual** mode. It remains in the disconnected status until the physical layer indicates that a connection is available. Users can check when a connection is established by monitoring the **IO.PhysicalLayerStatus** Tag.

10.3.4 Detecting Inactivity

A Driver can be programmed to disconnect automatically from the physical layer if it remains inactive for a certain period of time. A Driver considers the physical layer inactive if it is constantly requesting data (trying to receive characters) without retrieving any data. If any character is received (even if it is not a valid protocol character), inactivity time is restarted.

The inactivity timer is started whenever the time to receive characters of the physical layer expires. The timer is restarted (stopped) when any byte is received by the physical layer.

The inactivity time-out must be greater than the physical layer time-out, otherwise the physical layer can be considered inactive during a normal Driver operation. It is recommended to configure this value to at least 10 seconds.

10.4 Driver Threads

Both **E3** and **Elipse SCADA** benefit from multithreading in their data acquisition structures. With multithreading, several Drivers can communicate at the same time, without blocking each other. This section contains an explanation about **E3** and **Elipse SCADA** communication threads.

10.4.1 Elipse SCADA Threads

When using **Elipse SCADA** (Elipse32.exe), the following communication threads are used (only threads regarding I/O operations are listed here):

- **Main Thread**: This thread executes the user interface and all application scripts. All Driver requests (readings and writings triggered by scripts or by the user interface, and also asynchronous writings) are generated in this thread. The **Main Thread** blocks these synchronous requests while they are manipulated by the Driver

- **Driver Thread**: This thread is where the Driver is executed. There is a Driver Thread for each Driver declared in the application. This thread handles Tag's background readings (polling), as well as asynchronous requests generated by the **Main Thread**. If the Driver is configured for the **16-bit Compatibility** mode, then this thread does not exist and all requests (synchronous and background readings) are manipulated by the **Main Thread**

- **Physical Layer Thread**: This thread is started when a Driver enters the **Online** mode. This thread handles communication requests sent by the **Driver Thread**, and is also responsible for managing the connection (connecting, disconnecting, trying a reconnection, etc.). **Driver Thread** requests can only be manipulated by the **Physical Layer Thread** when the Driver is disconnected

10.4.2 Elipse E3 Threads

When using **E3**, each Driver has its own process (IOServer.exe). All Driver processes are managed by **E3Run** (E3Run.exe). This section only covers **IOServer** threads:

- **Main Thread**: This thread receives **E3Run** requests. These requests include starting or stopping a Driver, writing commands, and starting or stopping Tag scanning
- **Callback Thread**: This thread is responsible for sending values back to **E3Run**
- **Driver Thread**: This thread, similar to **Elipse SCADA's Driver Thread**, is where a Driver is effectively executed
- **Physical Layer Thread**: This thread, similar to **Elipse SCADA's Physical Layer Thread**, handles communication requests sent by the **Driver Thread**, and also manages the physical connection. This thread is started only when the Driver is in **Online** mode

CHAPTER 11 Knowledge Base

This section contains articles with tips about using **IOKit**.

11.1 Enumerating a List of Modems

The **IO.TAPI.ModemID** property allows users to select a new modem at run time. The problem is that this ID changes from computer to computer, and there is no simple way to determine a valid ID for a computer.

Starting with version 1.14, **IOKit** generates **events** whenever it cannot connect to a modem. These events list valid values for the modem ID of the current computer.

To enumerate a list of modems, users must follow these steps:

- **Start the Driver with an invalid ID**

    1. Configure the Driver to the **Offline** mode, configuring it to always start offline (select the **Start Driver OFFLINE** option on **Setup** tab), or write the value 0 (zero) to the **IO.WorkOnline** Tag while the Driver is executing.
    2. Write an invalid modem ID (a value of minus one is usually an invalid ID) to the **IO.TAPI.ModemID** property, by using the **IO.SetConfigurationParameters** Tag.
    3. Now the Driver enters the **Offline** mode, and with a configured invalid modem ID.
    4. Configure the Driver to the **Online** mode again (write the value one to the **IO.WorkOnline** Tag). This activates the TAPI Interface with an invalid modem ID, which generates events enumerating all valid IDs.

NOTE
These events are generated asynchronously. Therefore, when performing these steps in a script, finish it right at this point.

- **Capture enumeration events of valid modem IDs**

    1. To capture these events, users must declare a **IO.IOKitEvent** Tag in the application. On this Tag's **OnRead** event, users must check the events that report all available modem IDs.Users must search for events with type equal to 0 (zero, **Information**) and source equal to -3 (minus three, **Modem**). This enumeration always starts with an event with code equal to 2 (two), followed by an event with code equal to 3 (three) for each modem ID, and ends with an event with code equal to 4 (four).

The next table lists all events returned when two fictitious modems are found, with IDs 318 and 99383.

Enumeration events

TYPE	SOURCE	CODE	MESSAGE
0 (Information)	-3 (Modem)	2	Beginning of the list of available modems
0 (Information)	-3 (Modem)	3	318: Modem on serial 3
0 (Information)	-3 (Modem)	4	99383: Built-in modem
0 (Information)	-3 (Modem)	5	End of the list of available modems

The **OnRead** event can extract modem IDs from a message for events with code equal to 3 (three) and store them for later use.

- **Reconfigure IOKit with a valid modem ID**

    1. After enumerating all valid modem IDs, users can proceed with Driver configuration.
    2. First, configure the Driver again to the **Offline** mode (write the value zero to **IO.WorkOnline** Tag).
    3. Write one of the valid modem IDs enumerated in the **IO.TAPI.ModemID** property using the **IO.SetConfigurationParameters** Tag.
    4. Finally, configure the Driver again to the **Online** mode (write the value one to **IO.WorkOnline** Tag).

CHAPTER 12 Revision History

VERSION	DATE	AUTHOR	COMMENTS
			• Fixed a problem where the **IPSwitch** Tag was not working when triggered right after a successful reconnection (*Case 8012*). • Added the **ECHO Suppression** option to the **Ethernet** and **RAS** Interfaces (*Case 8383*). • The **Retry failed connection every ... seconds** option is now enabled by default (*Case 9140*). • Fixed the text of the **Give up after ... failed retries** option on **IOKit**'s **Setup** tab (*Case 9289*). • Added the **Specify local port** option to the **Ethernet** Interface (*Case 9311*).
1.15	24/08/2009	F. Englert	• Fixed a problem where **IOKit** may lock if a Driver is stopped while communicating with another thread (*Case 9423*). • **IOKit** now logs a message when the name of the log file changes, such as "Log redirected to newlog.log" (*Case 10175*). The next features and corrections are available only for Drivers compiled with **IOKitLib** v1.15: • Added support for readings by callback (*Case 8222*). • Fixed a problem when reading configuration Tags, such as **ParamItem** = "IO.Ethernet.MainIP", which caused a memory leak (*Case 8949*).

VERSION	DATE	AUTHOR	COMMENTS
			• **IOKit** nows stores in the log the Driver configuration and the Driver and **IOKit** version in the following situations: When the physical layer is connected, when the Driver starts in **Offline** mode, and when the log is activated or the log file name changes at run time (*Case 10175*). • Now users can change the name of the log file at run time, without configuring the Driver to the **Offline** mode, by writing to the **Enable** and **Filename** Tags (*Case 10175*).
1.14	15/01/2007	F. Englert	• Fixed a problem where **IOKit** may lock when finishing a serial Driver (*Case 5313*). • Improved the answering time for serial slave Drivers (*Case 7280*). • **IOKit** now enumerates all available modems, with their respective IDs, when a Driver is activated with an invalid modem ID (*Case 7474*). • Fixed a problem where the first communication operations, readings or writings, could fail while **IOKit** was still connecting to the target device (*Case 7614*). • Fixed a problem where **IOKit** could close when a very large **Float** or **Double** value was written or read, such as -4585887366944162000000000000.0 (*Case 7806*).
1.13	10/08/2006	F. Englert	• Fixed the timestamps of read or written values, which may be incorrect by one millisecond (*Case 7261*).

VERSION	DATE	AUTHOR	COMMENTS
1.12	03/07/2006	F. Englert	• Fixed a memory leak when a Driver returned an **HVALUE** sized as an array of **HVALUEs** (*Case 7092*). • Fixed a problem where a Driver with a separate I/O thread could lock or close when **IOKit** is configured in **Offline** mode (*Case 7099*).
1.11	11/05/2006	F. Englert	• Now users can check the **IOKitLib** version a Driver was compiled with. A function whose name is **IOKitLib_v1.11** is exported from Driver's DLL, allowing to easily identify the **IOKitLib** version (*Case 6968*). • Fixed a problem where the time-out is not obeyed when data was sent using the **Ethernet** Interface. This problem may lock the Driver if the paired device is not reading data from a connected socket (*Case 6935*). • Fixed a problem where the **Disconnect if inactive** option was not working if the Driver was in the **Listen** mode (*Case 6933*). • Fixed a problem where there were no reconnection retries if the **Serial** Interface could not be opened (*Case 6865*).
1.10	20/12/2005	F. Englert	• **IOKit** now generates a log file per day if the file name contains the **%DATE%** macro (*Case 2047*). • Fixed **IOKit** log, which mixed bytes sent to or received from different threads (*Case 6530*).

VERSION	DATE	AUTHOR	COMMENTS
1.09	07/12/2005	F. Englert	• The **Modem** Interface does not list or open a modem if the system contains other invalid modems (*Case 3613*). • A message box is now displayed when users enable the **Start driver OFFLINE** option, asking their confirmation for this option (*Case 4584*). • Configuration parameters of **IOKit** are now listed on the log file when a Driver is started (*Case 4584*). • Fixed a problem when delays in the configuration of the **RTS** signal do not work on **Windows NT** (*Case 5525*). • Added the **Inter-frame delay (milliseconds)** option to the **Serial** Interface, also available to the **Modem** Interface (*Case 5525*). • The logged message when a remote partner closes the socket, "recv3() returned error (unknown) (0)", changed to "Socket gracefully closed by the remote partner" (*Case 5599*). • The **Ethernet** Interface now uses WinSock v2.2 if available, while previous versions used WinSock v1.1. Some PLCs require that version (*Case 5617*). • Added the **Enable 'ECHO' suppression** option to the **Serial** Interface, also available to the **Modem** Interface (*Case 5647*). • Added the **Modem settings** option to the **Modem** Interface, allowing users to configure a modem (*Case 5656*). • Added support for browsing **IOKit** internal Tags, used by Tag Browser's dialog box in **E3** (*Case 6016*).

VERSION	DATE	AUTHOR	COMMENTS
			• Added three options to the selection list of baud rates of the **Serial** Interface: **38700**, **57600**, and **115200** (*Case 6076*). • **IOKit** now can receive and identify a source of UDP transmissions. This feature is now available to Drivers that must receive UDP transmissions (*Case 6085*). • **IOKit** does not try to reconnect immediately if the connection recently performed was lost (*Case 6294*). • Added the **None** option to the list of available interfaces, allowing users to disable the physical layer. This configuration is useful for Drivers that implement customized physical layers (*Case 6316*).

VERSION	DATE	AUTHOR	COMMENTS
1.08		F. Englert	• Added the **Ringing** status and the **Is Modem Connecting** Tag to the **Modem** Interface (*Case 4368*). • Implemented the **Backup IP**, **Ping**, and **UDP** options to the **Ethernet** Interface (*Cases 3014, 3015, and 3017*). • Added a protection against buffer overrun when generating logs (*Case 4365*). • Added the **Interbyte delay** option to the **Serial** and **Modem** Interfaces (*Case 4343*). • **IOKit** ported to **Linux** and **Windows CE** (*Case 4280*). • Implemented services to activate or deactivate logs used by the Driver Manager dialog box on **E3Run** (*Case 4513*). • Extra logs were added to help debugging and diagnosing a Driver: disconnected communication indicated in **sputs** and **sgets** (*Case 4851*). • **IOKit** version now is displayed on Driver's properties window ("Driver nonono (IOKit v1.08)") (*Case 4778*). • The previous status of the serial port is now restored when that port is closed. This avoid problems with programs that count on the serial port as configured on Windows Control Panel (*Case 4484*).
1.07	05/04/2004	A. Corrêa	• Text revision.
1.07	01/26/2004	F. Englert	• The **Serial** Interface sometimes takes one second to handle input bytes. This only affects slave Drivers (*Case 3279*).

VERSION	DATE	AUTHOR	COMMENTS
1.06	12/18/2003	F. Englert	• **IOKit** now checks the Driver version to ensure that the Driver was compiled with a compatible version of **IOKit** (*Case 3018*).
1.05	11/27/2003	F. Englert	• Implemented the **Listen for connections on port** option on the **Ethernet** Interface. Slave Drivers via Ethernet are now supported (*Case 3018*).
1.04	10/27/2003	F. Englert	• The **RAS** Interface now correctly disconnects the socket when receiving a **String** "NO CARRIER" after establishing the connection (*Case 2643*).
1.03	09/10/2003	F. Englert	• The **Ethernet** tab does not allow editing IP port parameters when the **RAS** Interface is selected (*Case 2675*). • Allows inserting the % PROCESS% macro on log file names, allowing a process ID to be inserted on the file name. For example, **c:\driver_% PROCESS%.log** creates the log file **c:\driver_0000.log**, where **0000** is the process ID (*Case 2676*). • The **Enable** and **Filename** properties do not work, and they must be immediately applied when modified using the **Set Configuration Parameters** Tag (*Case 2678*).
1.02	07/10/2003	F. Englert	• The **RAS** Interface now connects to a single IP port (the port specified on the **Ethernet** tab). The RAS device checks whether there is a modem available and sets the modem to the socket or else denies the connection to the socket (*Case 2656*).

VERSION	DATE	AUTHOR	COMMENTS
1.01	10/06/2003	F. Englert	• The **RAS** Interface now disconnects the socket when receiving a **String** "NO CARRIER" after establishing a connection (*Case 2643*). **NOTE: This feature is not working correctly, use version 1.04**.
1.00	08/11/2003	F. Englert	• First version. The Ethernet **Ping**, **Backup address**, **UDP**, **Use new socket**, and **Accept connection** options are still not implemented.

Elipse Drivers

Headquarters
Rua 24 de Outubro, 353 - 10º andar
90510-002 Porto Alegre
Phone: (+55 51) 3346-4699
Fax: (+55 51) 3222-6226
E-mail: elipse-rs@elipse.com.br

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 more information about a representative in your city or country.
www.elipse.com.br
kb.elipse.com.br
forum.elipse.com.br
www.youtube.com/elipsesoftware
elipse@elipse.com.br

Gartner 2014 CoolVendor

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)