

# WAN HDLC/4 USER MANUAL

## For Windows NT/2000/XP

INTELLIGENT COM PORTS WITH MULTIPROTOCOL SUPPORT

<p><b>ACKSYS</b> COMMUNICATIONS &amp; SYSTEMS 10, rue des Entrepreneurs ZA Val Joyeux 78450 VILLEPREUX FRANCE</p>	 <p>Tel : +33 (0)1 30 56 46 46 Fax : +33 (0)1 30 56 12 95 Web : <a href="http://www.acksys.fr">www.acksys.fr</a> Hotline : <a href="mailto:support@acksys.fr">support@acksys.fr</a> Sales : <a href="mailto:sales@acksys.fr">sales@acksys.fr</a></p>
---	--

Important : The ACKSYS Windows Device Driver is common to all Windows Operation systems NT, 2000 and XP.

## TABLE OF CONTENTS

I	PRESENTATION .....	4
II	OPERATIONAL RESTRICTIONS.....	5
III	SUPPORTED PROTOCOLS : AN OVERVIEW .....	5
III.1	Asynchronous mode .....	5
III.2	Synchronised asynchronous mode.....	5
III.3	MONOSYNC and BISYNC modes.....	5
III.4	Raw HDLC mode .....	6
III.5	HDLC ABM or LAPB mode.....	6
IV	PHYSICAL BOARD INSTALLATION .....	7
IV.1	SW1, SW2 and SW3 switch settings .....	7
IV.2	Starting the board.....	8
IV.3	Diagnostic LEDs on the main board .....	8
IV.4	LEDs installed on the mezzanine board.....	10
V	WIRING.....	11
V.1	SUBD62 connector pin assignment.....	11
V.2	Pin assignment for the 4P570M25 cable .....	12
V.3	The electrical interfaces.....	14
VI	NT/2K DRIVER .....	15
VI.1	Reboot .....	16
VI.2	Installation and configuration .....	17
VI.3	Configuring installed boards.....	19
VI.4	Board properties .....	20
VII	CHECKING THE INSTALLATION .....	22
VIII	DÉVELOPPEMENT TOOLS AND EXAMPLES .....	23
 <b>COM COMPATIBILITY MODES .....</b>		 <b>24</b>
I	APPLICATION PROGRAMMING INTERFACE (API) .....	24
I.1	Programming asynchronous communications.....	25
I.2	Programming synchronous communications .....	27
I.3	Programming the LAPB (or HDLC-ABM) protocol.....	29
I.4	Programming driver-specific services. ....	31
I.5	Standard Windows NT utilities. ....	32
I.6	Other utilities. ....	33
II	DÉTAILÉD RÉFÉRENCE MANUAL.....	34
II.1	Extract from the mcc_mcx.h file .....	34
II.2	SET/GET SYNC STATE functions .....	37
II.3	Example of SET_SYNC_STATE .....	40
II.4	CMD and CMD_AUTO functions .....	41
II.5	Samples of CMD and CMD_AUTO .....	44
III	APPENDIX : SPECIFIC ERROR CODES.....	47
IV	APPENDIX : LIMITATIONS AND DIFFÉRENCES WITH THE COM PORTS. ....	48

## ANNEXES

**I PRESENTATION**

This manual describes the technical characteristics of the boards in the ACKSYS WANHDLC range and explains how they are integrated and used with the Windows NT and Windows 2000 operating systems.

The boards in the WAN HDLC range extend the communication capabilities of a PC (PCI or CPCI 6U) by adding serial communication interfaces that are multiprotocol (asynchronous, synchronised asynchronous, Monosync, Bisync, Raw HDLC & HDLC ABM) and multi-interface (RS232/V28, RS422/V11, RS485 & V35).

Each board has 4 fully independent channels of this type.

A typical configuration might be :

- Channel 1 : Raw HDLC RS422
- Channel 2 : ASYNC synchronised RS232
- Channel 3 : HDLC ABM V35
- Channel 4 : MONOSYNC RS485

These boards offload communication line management tasks such as acquisition, transmission and protocol from the PC, releasing the bandwidth of the PCI bus for the exchange of useful information between the PC and the board.

Running on Windows NT 4.0 and Windows 2000 systems, an ACKSYS driver can make use of the board's 4 intelligent serial ports in compliance with the specifications of the Windows Win32 API (Files, COMM & Ioctl), the market standard for traditional PC serial ports.

## II OPERATIONAL RESTRICTIONS

Number of boards .....	limited to the number of available slots
Operating system.....	Windows NT 4.0 from SP3 upwards, and Windows 2000. Monoprocessor only.
Firmware MCX Multiprotocol.....	Version 3.1 or later
Buses supported .....	PCI & CPCI 5V
Max. bitrate in ASYNC mode .....	250 Kbps
Max. bitrate in ASYNC sync mode .....	250 Kbps
Max. bitrate in MONOSYNC mode .....	4 Mbps
Max. bitrate in BISYNC mode .....	4 Mbps
Max. bitrate in raw HDLC mode .....	4 Mbps
Max. bitrate in HDLC ABM mode .....	4 Mbps

## III SUPPORTED PROTOCOLS : AN OVERVIEW

This is a quick overview of the protocols supported by the board. For full information on these protocols, consult the documentation on the MCX Multiprotocol firmware.

The synchronous protocols support the FM0, FM1, NRZi & NRZ encoding modes (see the VINIT command in the MCX MULTIPROTOCOL firmware).

The transmit/receive clocks can be configured (internally, externally or in data).

### III.1 Asynchronous mode

This is the mode used by traditional PC COM ports.

### III.2 Synchronised asynchronous mode

This mode is identical to asynchronous mode (above) except that the transmit or receive clock can be external to the board.

### III.3 MONOSYNC and BISYNC modes

xSYNC is a mode in which the board only manages a frame envelope.

In transmission mode, the board encapsulates the data for transmission (buffer used by the WriteFile function) in a synchronous frame which looks like this :

“S1/S2” “OUTGOING DATA” where S1 is the synchronisation byte in MONOSYNC mode or the first synchronisation byte in BISYNC mode, and S2 is the second synchronisation byte in BISYNC mode.

The PC sends only the outgoing data block to the board, which handles frame transmission.

In reception mode, if the board receives a frame of the following type :

“S1/S2” “FIXED LENGTH INCOMING DATA”.

The board handles the acquisition of the frame and stops data reception after n bytes (n is customisable) and sends only the incoming data block to the PC (buffer used by the ReadFile function).

In Monosync mode, the board uses an 8-bit synchronisation called S1.

In Bisync mode, the board uses a 16-bit synchronisation called S1/S2.

### III.4 Raw HDLC mode

RAW HDLC is a mode in which the board only manages the envelope of the HDLC frame. In transmission mode, the board encapsulates the outgoing data (buffer used by the WriteFile function) in an HDLC frame :

“Flag 0x7E” “OUTGOING DATA” “CRC CCITT PRESET1” “Flag 0x7E”

The PC sends only the outgoing data block to the board which handles the calculation and generation of the CRC, the management of the stuffing<sup>1</sup> bit and the generation of the frame start and end flags.

In reception mode, if the board receives a frame of the following type :

“Flag 0x7E” “INCOMING DATA” “CRC CCITT PRESET1” “Flag 0x7E”

The board takes full control of the frame (CRC, ABORT, OVERRUN, FRAMING ERROR), handles the stuffing bit and only sends the incoming data block to the PC (buffer used by the ReadFile function).

### III.5 HDLC ABM or LAPB mode

HDLC ABM is a mode in which the board manages level 2 of the HDLC procedure in ABM mode. The distinctive characteristics of this protocol are :

- The general frame format is that described above for RAW HDLC mode.
- Unnumbered frames SABM UA DM
- Supervision frames RR, REJ, RNR
- Information frame I
- T1 timeout
- N2 retry count
- Max. frame length : N1
- Width of the K sliding window : modulo 8
- P/F bit

The PC only manages the buffers for data carried in the I information frames and the connection/disconnection of the communication link.

---

<sup>1</sup> The stuffing bit is a mechanism that is specific to the HDLC protocol that prevents the possible occurrence of a flag between a frame's start and end flags.

## **IV PHYSICAL BOARD INSTALLATION**

### **IV.1 SW1, SW2 and SW3 switch settings**

SW1-1 and SW1-2 are set to OFF (Mode Built in firmware)

SW1-3 set to ON (Watchdog enable)

SW1-4 set to ON (Reserved)

SW2-1 set to ON (Battery Enable)

SW2-2 set to OFF (Reserved)

SW3-1 and SW3-2 set to OFF (Reserved)

## IV.2 Starting the board

Each time the board is reinitialised (powered up or reset), all the channels are reset and initialised in high impedance mode. An on-board self-test program is run automatically and displays a code identifying the current test on a group of 8 diagnostic LEDs (numbered DL0, DL1 ...to DL7). If a test returns an error, the program stops, leaving the code of the current test displayed on the LEDs.

When all the tests have been successfully executed (this takes around 15 seconds), the self-test program lights up and switches off each of the LEDs in rotation indicating that the board is now awaiting an instruction from the PC driver before continuing initialisation.

This provides a visual indication that there is no major operational defect on the board.

When the PC driver has been successfully loaded, the board stops the rotating LED display and flashes the DL0 diagnostic LED on and off once per second.

## IV.3 Diagnostic LEDs on the main board

This group of eight LEDs is numbered from DL0, DL1 ... thru DL7. Generally DL0 thru DL6 are yellow and DL7 is red. The purpose of these LEDs is to check that the board is operating correctly and to monitor reception/transmission operations on the serial ports.

To interpret the status of these LEDs, 3 distinct phases should be identified:

Phase 1 : while the self-test program is being run, i.e. immediately after the board has been powered up or reset.

The board will display a code on the LEDs in the event of an error :

LED	DL0	DL1	DL2	DL3	DL4	DL5	DL6	DL7
Error code bit	C <sub>0</sub>	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>

(C0 refers to the low-order bit, DL0 which you can see printed on the circuitry)

The codes are as follows :

Code 01h .....	CPU flag register error.
Code 02h .....	CPU register error.
Code 03h .....	BIOS memory checksum error.
Code 04h .....	DMA controllers error.
Code 05h .....	System TIMERS error.
Code 06h .....	Error in the address test of the first 64 K or bad memory configuration.
Code 07h .....	Error in the address test of the first 64 K of RAM.
Code 08h .....	Error on the INT controller.
Code 09h .....	Unexpected INT detection.
Code 0Ah.....	No TIMER interrupt.
Code 0Bh .....	CPU already in protected mode.
Code 0Ch .....	Error in DMA page register.
Code 0Dh .....	No memory refresh.
Code 0Eh .....	Error on keyboard controller.
Code 0Fh .....	Cannot enter protected mode.
Code 10h .....	Error on GDT or IDT registers.
Code 11h .....	Error on LDT register.

---

Code 12h .....	Error in task register.
Code 13h .....	Error on LSL instruction.
Code 14h .....	Error on LAR instruction.
Code 15h .....	Error on VERR / VERW.
Code 16h .....	Error on A20 address line.
Code 17h .....	Unexpected exception.
Code 18h .....	Shutdown during memory test.
Code 19h .....	Copyright checksum error.
Code 1Ah .....	Parameter checksum error.
Code C0h .....	Error in memory test.
Code C1h .....	Error on IO/CHECK signal.
Code C2h .....	“Watchdog time out”.
Code C4h .....	“Bus time out”.
Code 81h .....	Error on UART, SCC or SCA.
Code 82h .....	Dual access memory error.
Code 83h .....	Unexpected TRAP error.
Code 84h .....	Buffer error.
Code 85h .....	Firmware checksum error.
Code 86h .....	Lithium battery error.
Code 87h .....	MCXPCI to PC interrupt error.
Code 88h .....	“Watchdog” error.
Code 89h .....	FIFO, flags or Int error at MCXPCI end. Or MCXPCI error when reading FIFO.
Code 8Bh .....	SCC error during high-speed DMA test.
Code 8Ch .....	General protection fault.
Code 8Dh .....	Memory size error.
Code 8Eh .....	NMI interrupt received.
Code 8Fh .....	Error on 100 us counter.

Phase N°2 : before the PC driver is loaded

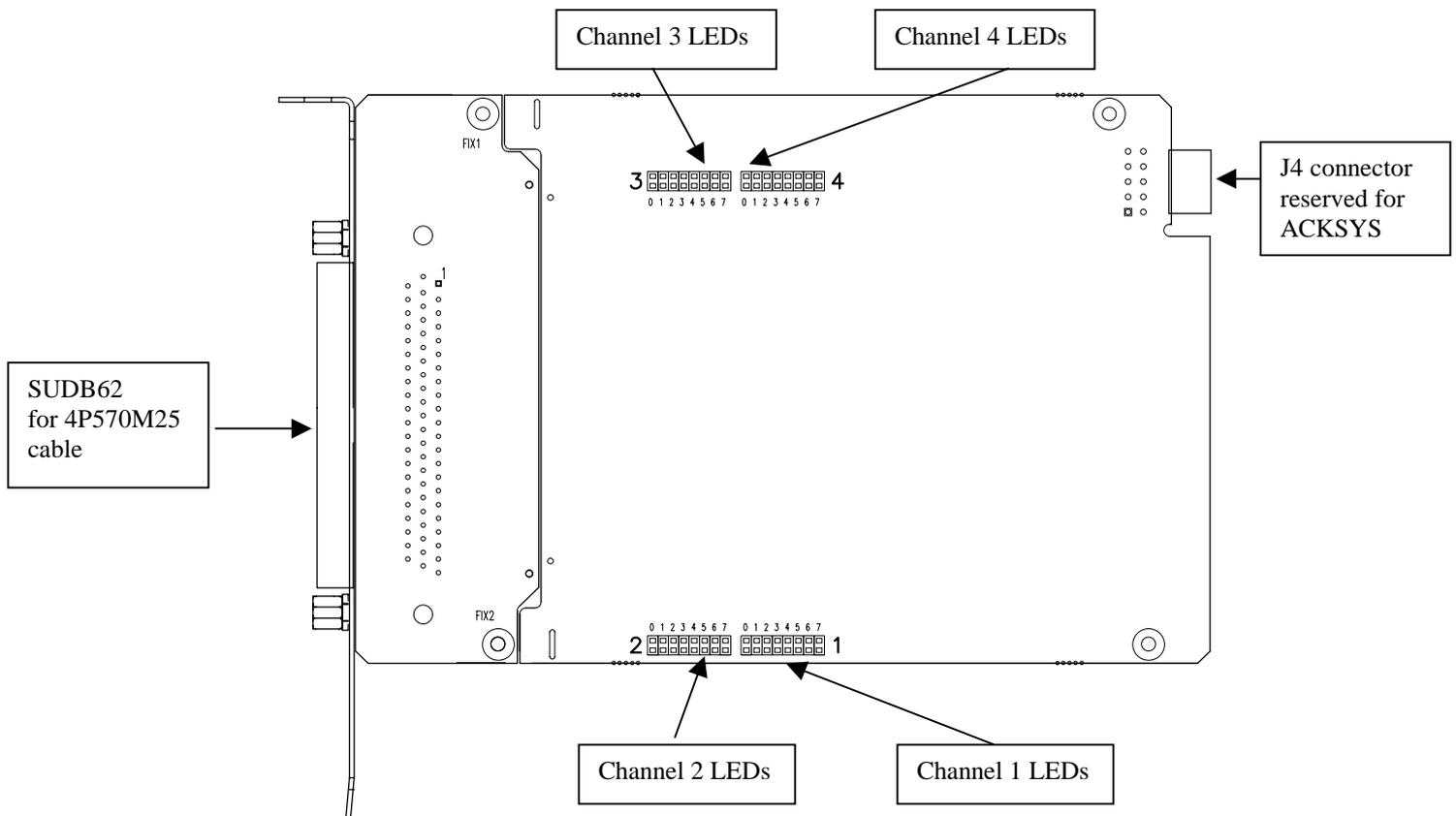
When the self-test phase has been successfully executed, the DL0 thru DL7 LEDs are lit up in rotation.

Phase N°3: after the PC driver has loaded

- DL0 : flashes once per second and indicates that the board is active
- DL1 : the board executes a command
- DL2 : the board has sent an interrupt to the PC that has not been acknowledged
- DL3 : Reception in asynchronous mode on one or more of the 4 channels
- DL4 : Transmission in asynchronous mode on one or more of the 4 channels
- DL5 : Reception in synchronous mode on one or more of the 4 channels
- DL6 : Transmission in synchronous mode on one or more of the 4 channels
- DL7 : Reception error on one or more of the 4 channels

#### IV.4 LEDs installed on the mezzanine board

Each channel has a group of 8 LEDs that indicate whether the interface has been programmed for RS232, V35, RS422, RS485 or High impedance (also called HiZ).



	Led <sub>0</sub>	Led <sub>1</sub>	Led <sub>2</sub>	Led <sub>3</sub>	Led <sub>4</sub>	Led <sub>5</sub>	Led <sub>6</sub>	Led <sub>7</sub>
HiZ	0	0	0	0	0	0	0	0
RS-232	0	1	0	0	0	1	0	0
V35	0	1	1	1	0	1	1	1
RS-422	0	0	1	0	0	0	1	0
RS-485	1	0	1	0	0	0	0	0

For value 1, the LED is on, and for value 0 the LED is off.

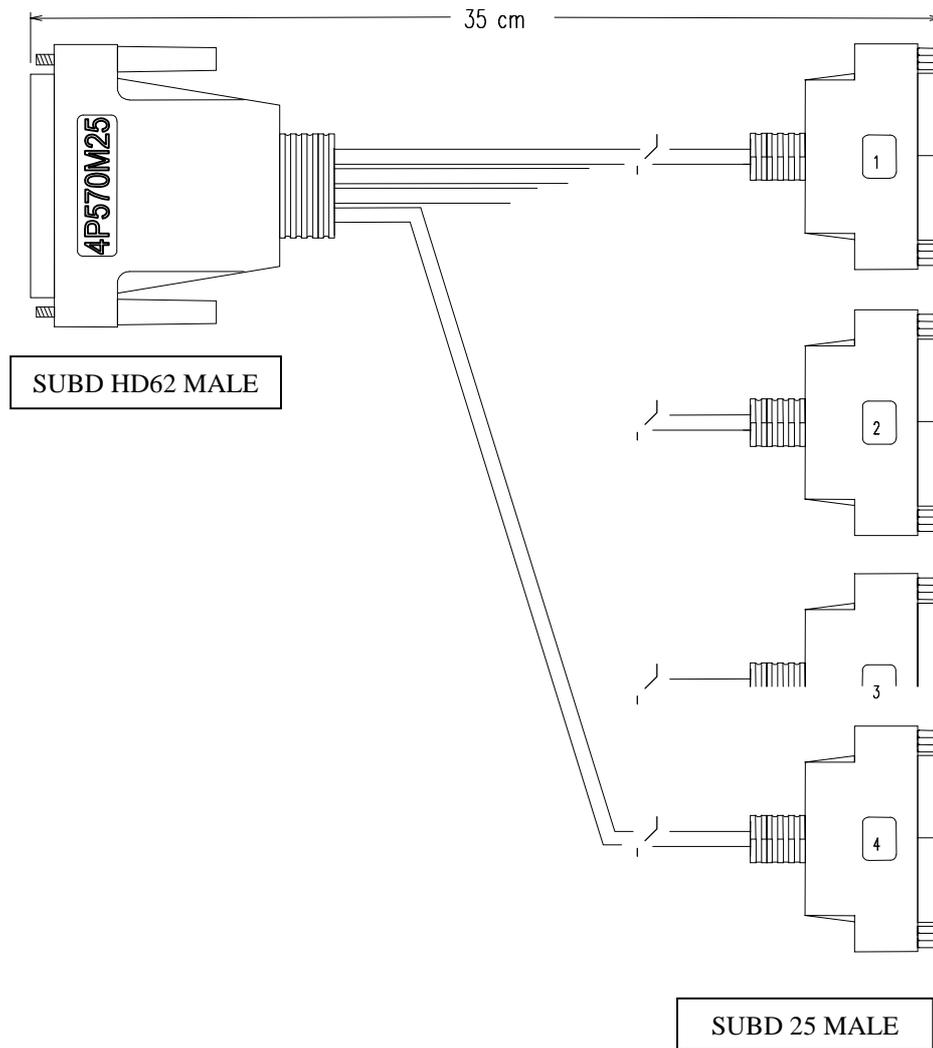
Note: the channels stay in the HiZ position (LEDs off) until the ACKSYS driver has started.

## V WIRING

### V.1 SUBD62 connector pin assignment

Signal	Mnemonic	Direction	Pin N° Channel 1	Pin N° Channel 2	Pin N° Channel 3	Pin N° Channel 4
TxD	Transmit Data	Output (A)	37	21	27	32
TxD	Transmit Data	Output (B)	57	42	5	52
RxD	Receive Data	Input (A)	15	62	26	10
RxD	Receive Data	Input (B)	36	20	46	31
TxCkOut	Transmit Clock	Output (A)	56	41	4	51
TxCkOut	Transmit Clock	Output (B)	14	61	25	9
TxCkIn	Transmit Clock	Input (A)	35	19	45	30
TxCkIn	Transmit Clock	Input (B)	55	40	3	50
RxCkIn	Receive Clock	Input (A)	13	60	24	8
RxCkIn	Receive Clock	Input (B)	34	18	44	29
RTS	Request To Send	Output (A)	54	39	2	49
RTS	Request To Send	Output (B)	12	17	23	7
CTS	Clear To Send	Input (A)	33	38	22	28
CTS	Clear To Send	Input (B)	11	16	1	6
GND			47, 58			
+5V (useful in RS485 mode for specifying the polarisation)			43, 48, 53, 59			

## V.2 Pin assignment for the 4P570M25 cable



**Important before connecting the 4P570M25 cable to the WAN-HDLC/4 board**  
**Switch off the PC.**

The signals associated with the serial channels 1, 2, 3 and 4 are assigned respectively to the connectors numbered 1, 2, 3 and 4 on the 4P570M25 cable.

**PIN ASSIGNMENT OF THE 4 25-PIN MALE CONNECTORS(DB25 FORMAT SIZE C)  
ON THE 4P570M25 CABLE**

<b>Pin N°</b>	<b>Signal</b>	<b>Mnemonic</b>	<b>Direction</b>
1	P.G	Protective Ground	
2	TxD	Transmit Data	Output (A)
14	TxD	Transmit Data	Output (B)
3	RxD	Receive Data	Input (A)
16	RxD	Receive Data	Input (B)
24	TxClkOut	Transmit Clock	Output (A)
11	TxClkOut	Transmit Clock	Output (B)
15	TxClkIn	Transmit Clock	Input (A)
12	TxClkIn	Transmit Clock	Input (B)
17	RxClkIn	Receive Clock	Input (A)
9	RxClkIn	Receive Clock	Input (B)
20	N.C.	Not Connected	
23	N.C.	Not Connected	
4	RTS	Request To Send	Output (A)
19	RTS	Request To Send	Output (B)
5	CTS	Clear To Send	Input (A)
13	CTS	Clear To Send	Input (B)
8	N.C.	Not Connected	
10	N.C.	Not Connected	
6	N.C.	Not Connected	
18	N.C.	Not Connected	
22	N.C.	Not Connected	
21	N.C.	Not Connected	
7	GND		
25	+5V (useful in RS485 mode for specifying the polarisation)		

The +5V output (pin n° 25) on each connector is protected by a resettable thermal fuse. If an accidental short-circuit occurs on the output connectors, it is advisable to stop the system and identify the cause of the short-circuit. The system can only be powered up again after 20 seconds have elapsed.

### V.3 The electrical interfaces

	RS232	V35	RS422	RS485
TxD(A)	V28	V35	V11	RS485
TxD(B)	HiZ	V35	V11	RS485
RTS(A)	V28	V28	V11	RS485
RTS(B)	HiZ	HiZ	V11	RS485
TxCkOut(A)	V28	V35	V11	RS485
TxCkOut(B)	HiZ	V35	V11	RS485
RxD(A)	V28	V35	V11	RS485
RxD(B)	>12K $\Omega$	V35	V11	RS485
RxCkIn(A)	V28	V35	V11	RS485
RxCkIn(B)	>12K $\Omega$	V35	V11	RS485
CTS(A)	V28	V28	V11	RS485
CTS(B)	>12K $\Omega$	>12K $\Omega$	V11	RS485
TxCkIn(A)	V28	V35	V11	RS485
TxCkIn(B)	>12K $\Omega$	V35	V11	RS485

HiZ : High impedance  
>12K $\Omega$  : relative to the ground

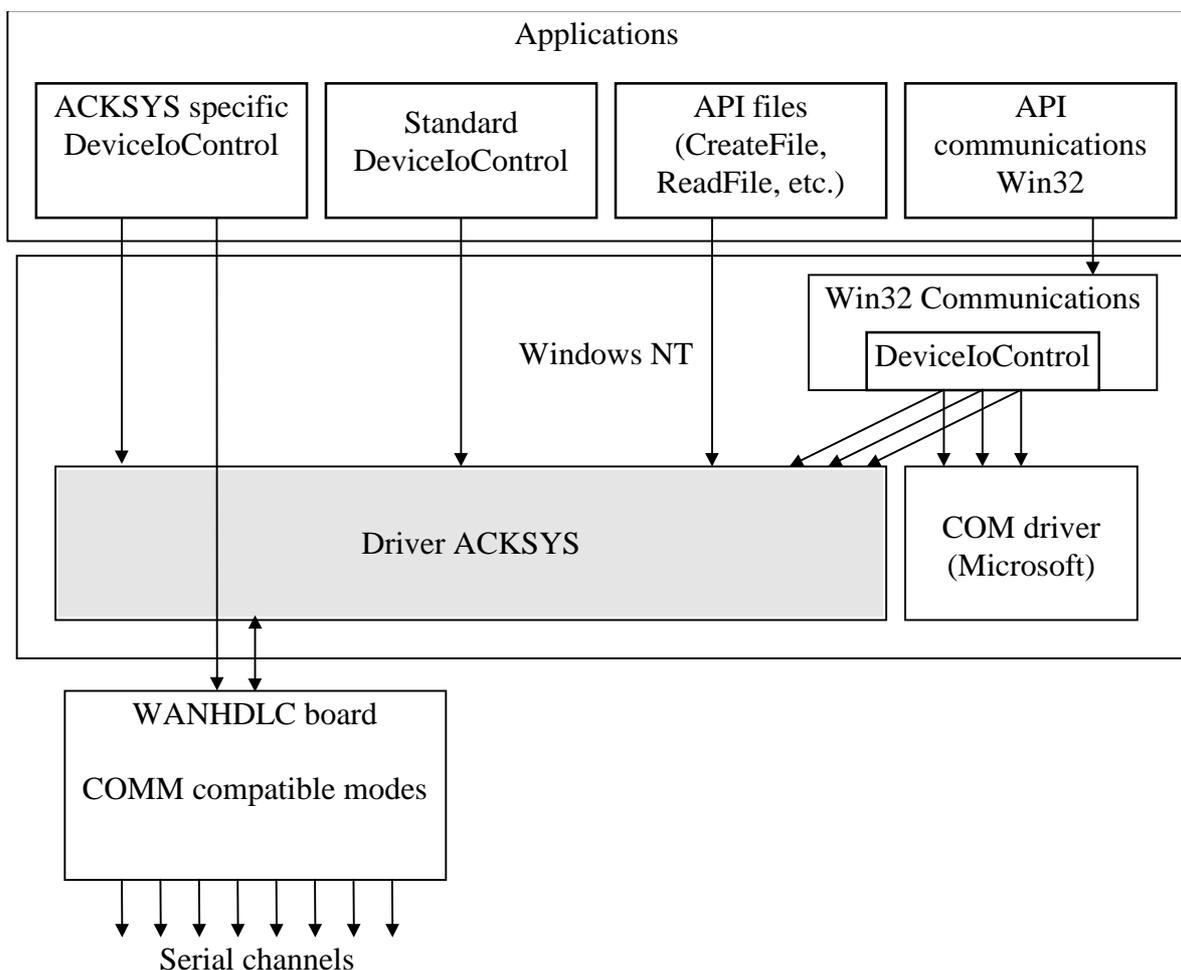
The signal pairs RxD(A,B), RxCkIn(A,B) and TxCkIn(A,B) programmed in differential mode (V11 or RS485) are arranged so that the A and B signals are connected to a 120 Ohm terminal resistor.

The polarisation resistors should be built into the connection cable.

To polarise a pair (A,B), connect a 470 $\Omega$  (to a 1K $\Omega$  (1/4W 5%) resistor between signal A and the signal ground (GND pin 7 or 26) and between signal B and the +5V power supply (pin n $^{\circ}$  25)

## VI NT/2K DRIVER

The following diagram shows how the driver is installed in the Windows architecture :



The diagram shows that the driver uses the same API as the COM ports. If a specific application needs to use board features without accessing them via the Win32 API, it can access the driver or even the board directly via the DeviceIoControl commands described in the section entitled COM COMPATIBILITY MODES.

Typical examples of this are switching to RS422 or HDLC mode.

## VI.1 Reboot

You must install the board in the computer before starting the installation program. Otherwise the program will not be able to recognise the board and allocate resources automatically.

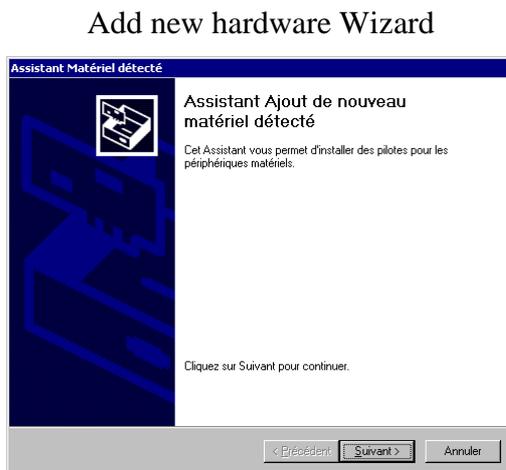
Users of Windows NT 4 need simply to check that the Service Pack installed on the system is later than or identical to the SP3 and then go on to the next section.

When Windows 2000 users install the board and then reboot their computer, a new, unknown device is detected automatically :



Board installation is not integrated in the Plug-and-play environment. This means that before using the installation program supplied by ACKSYS (see next section) you will have to disable the plug-and-play capability for this device.

To do this, here is the dialogue that you should follow with Windows 2000 :



Select “**Next>**”



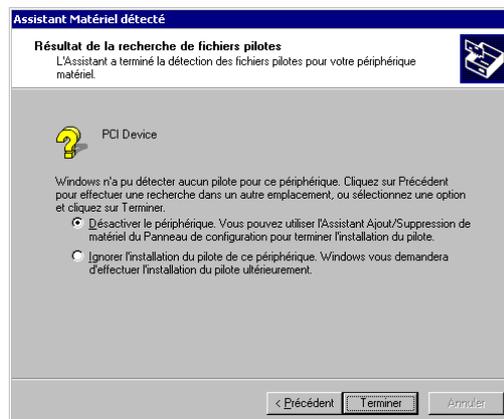
Select “**Next>**”

### Looking for driver files



Select “**Next>**”

### Driver files found



Select “**End**”

Next time you reboot the computer, Windows will no longer ask these questions if you have selected “Disable device” in the last step.

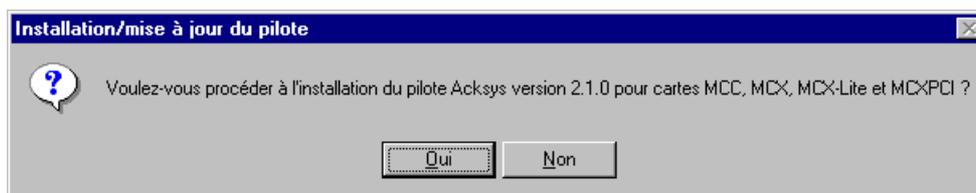
When you have completed this dialogue, proceed with driver installation.

## VI.2 Installation and configuration

When you have installed the board in the computer and restarted the operating system, place the “MCC, MCX, MCX-Lite Windows NT Driver & utilities” diskette in drive A: and execute the command :

**A: \SETUP**

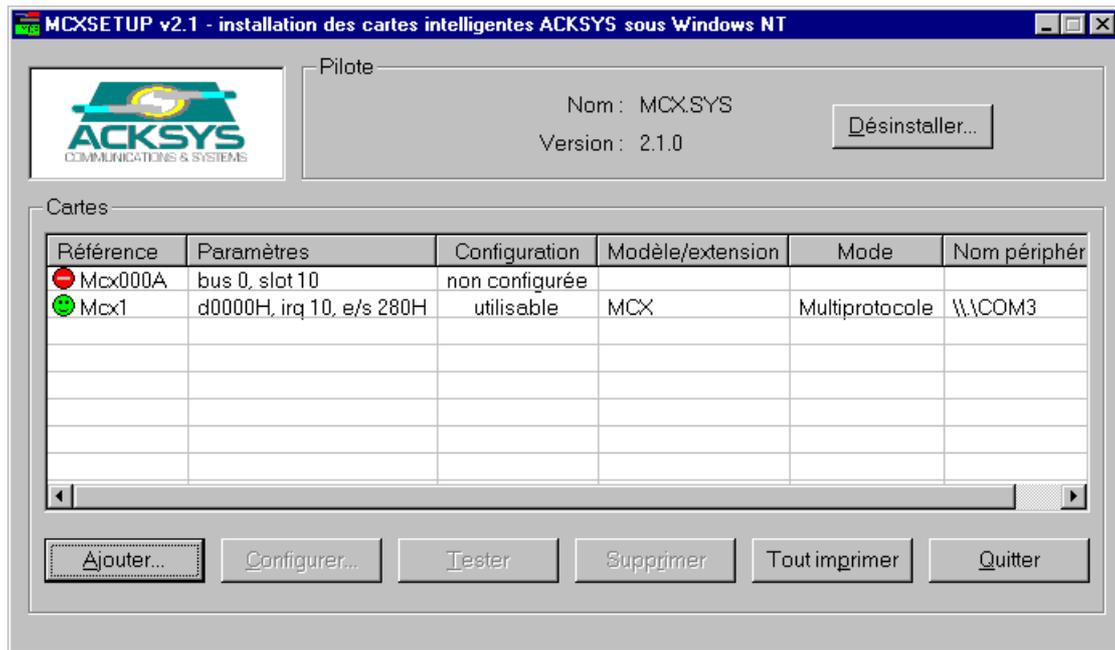
This script starts the installation and configuration program called “MCXSETUP”. The first time you execute the program, it will detect that the MCX.SYS driver has not yet been installed and ask you to confirm the installation :



If a previous version of the program is already installed on the disk, MCXSETUP checks that the version you want to install is more recent and that updating is possible, before proceeding with installation.

It then installs the “MCX.SYS” device driver and a number of utilities on the hard disk. You will then be able to run these utilities directly from the command line prompt, or from the menu bar (**start** → **execute** → **mcxsetup** → **OK**).

The first window displayed by MCXSETUP is split into two halves. The first half lets you manage the driver installation process and the bottom half manages the installation and initialisation of the boards :



This is what the MCXSETUP v2.1 main window looks like

- ◆ **Version** indicates the version of the MCX.SYS driver file that is installed on your hard disk.
- ◆ **Deinstall** lets you delete MCX.SYS, MCXSETUP and the disk utilities, and deletes the driver information from the Register.

---

### VI.3 Configuring installed boards

The list in the lower half of the window indicates the boards that have already been detected or installed and their main characteristics.

A red icon indicates that boards have been detected but that they are not yet configured and cannot be used. You must configure them before they can be used.

A yellow icon indicates that boards are inhibited. In other words these have been deliberately placed “out of bounds” (you have checked the “inhibit” box in the board properties). You can re-enable this board before it can be used.

A green icon indicates that enabled PCI boards have been recognised. Boards with this status can be used.

You can access a board by selecting it then using either of the buttons at the bottom of the window or clicking with the right-hand mouse button. Double-clicking opens the properties window (see below).

The **Add...** button lets you define an ISA board and is therefore not discussed in this manual. The **Test** button lets you run some basic checks on the operation of a selected board, and displays a report after a few seconds. The **Quit** button checks that, after a board has been modified, the same resources are not being used by several boards, offers some installation advice, then ends the program.

The **Print all** button, accessible if no board has been selected, prints a summary of the configuration of each board. This button changes to **Print** when a board has been selected.

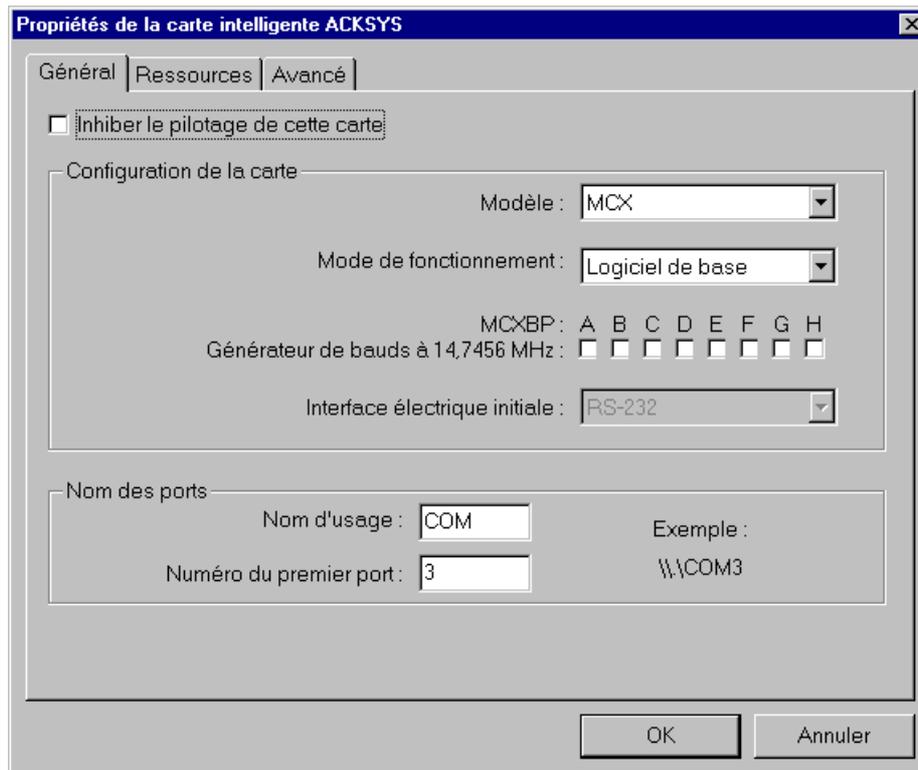
**The overall configuration procedure is consequently as follows :**

- 1) Find the line that corresponds to the board you want to configure. If several boards have been installed, they can be identified by their bus and slot numbers.
- 2) Double-click on the board to open the properties window.
- 3) Select the model from the list of boards.
- 4) Select “multiprotocol” mode
- 5) Select the other parameters according to your needs, and click on OK to close the properties window.
- 6) Reselect the board in the list and click on **Test**.

## VI.4 Board properties

These parameters modify the operation of the device driver. They are only analysed each time the driver is restarted. The driver can be restarted either when the system is rebooted, or manually from the “Devices” icon in the Windows Control Panel, or at the command line prompt by entering the “net stop mcx” command followed by “net start mcx”, or by MCXSETUP with user confirmation.

Use the **General** tab to specify the characteristics of the board you want to install.



General properties of an MCX, MCXPCI or MCX-Lite board

### ◆ Inhibit driver for this board

Checking this box will prevent the driver from accessing the board described. You can use the box to temporarily inhibit an installed board, or temporarily delete a board from the computer without generating messages in the Event Viewer.

### ◆ Model

Selects the board model. Only models that correspond to the bus being used will be displayed in this list. The model must be specified accurately, as other options depend on your choice.

### ◆ Operating mode

This mode corresponds to the settings of the SW1-1 and SW1-2 switches. For the WANHDLC board, select “multiprotocol”.

### ◆ 14,7456 MHz baud generator

Does not apply to WAN HDLC boards.

### ◆ Initial electrical interface

Specifies the electrical interface that the driver will use to initialise on

---

all the channels. If you want to program the channels with different interfaces, leave the HiZ setting here, and assign the interfaces at application program level.

◆ **Usal name**

Prefixes the name that will be used to identify the board's channels. "COM" can be specified to use standard tools like HyperTerminal. By specifying another name, you can use channel names with a fixed format (see later the description of the CreateFile).

◆ **First port number**

This number will be assigned to the name of the board's first channel. The other channels will be numbered sequentially starting with this number.

◆ **Example**

This displays the name of the board's fist channel as it will be recognised by the driver. The channel names are formed by concatenating the fixed character string "\\ . \\", the usual name and the number in the sequence starting with "first port number".

The **Resources** tab is only available for an ISA bus.

The **Advanced** tab gives access to the options and compatibility with earlier versions of the driver (1.6.4 thru 1.8.3).

This does not apply to WAN HDLC boards that are only fully supported as from version 2.1.0 of the MCX driver.

## VII CHECKING THE INSTALLATION

You can check that the driver has been properly started by consulting the Event Viewer, which displays two messages about the driver :

- a) a message indicating that the driver has been loaded and its version,
- b) for each board installed, a message indicates the version of the EPROM, the number of channels recognised by the board and other items of useful information.

### **In Windows 2000**

You can check that the driver has been started correctly by consulting the Device Manager, accessible notably by clicking with the right-hand mouse button on the Workstation icon.

**WARNING :** when you have installed a new driver, the “General” tab will tell you that there is a problem with a device. Ignore this message which should disappear the next time you reboot the system. This message has no impact on operation: the driver will function correctly as soon as it has been installed.

The options in the “Driver” tab let you stop, restart or inhibit the driver.

### **In Windows NT 4**

You can check that the driver has been started by consulting the “Devices” icon in the Control Panel.

---

## VIII DEVELOPPEMENT TOOLS AND EXAMPLES

You can communicate with the driver via your C language source programs, either through the standard functions of Win32 (e.g. CreateFile, etc.) or with the commands and structures that are specific to your driver.

The following files will be useful for developing your own applications :

A:\WINNT\SDK\INCLUDE\MCC\_MCX.H

A:\WINNT\SDK\INCLUDE\MCXPROTO.H

If necessary, copy these files on to your hard disk so that they can be accessed by your C language source programs.

You will find sample programs in A:\WINNT\SDK. More specifically, the SDK\MSVC\LIB directory contains a library of functions that are useful for getting to know the driver; these functions use the “header” SDK\INCLUDE\ACK\_W32.H and are compiled with the MultiThread option in the SDK\LIB\MCC\_MCX.LIB. library.

---

## COM COMPATIBILITY MODES

---

The following sections describe how the driver operates with the MCX MULTIPROTOCOL on-board firmware.

### I APPLICATION PROGRAMMING INTERFACE (API)

Each serial channel can be programmed independently, whether as regards the electrical interface, the signal format, the frame format, the protocol or the flow control, etc. Each channel can be driven by a different application if necessary.

The file name used to access the channels is defined when the board is configured. If the “usual name” is not “COM”, the channels will be numbered 01, 02, etc. up to the number of channels installed on the board. If the “usual name” is “COM”, the channels will be numbered without the leading zero, as from “first port number”. For example :

Usual name	First port number	Channel identification
COM	3	\\.\COM3, \\.\COM4, etc.
COM	9	\\.\COM9, \\.\COM10, etc.
Mcx1	ignored	\\.\Mcx101, \\.\Mcx102, etc.
McxB	ignored	\\.\McxB01, etc.

The available APIs are based on the Win32 documentation and fall into three groups : **file services** (CreateFile, ReadFile, etc.), **communication services** (SetCommState, etc.) and the **DeviceIoControl** file service which is used by all the driver-specific functions that are not provided in Win32 (change of protocol or electrical interface, etc.).

In the following pages, the APIs are presented according to whether they are used in :

- ☞ asynchronous communications,
- ☞ synchronous communications,
- ☞ LAPB communications,
- ☞ driver-specific services,
- ☞ utilities.

## I.1 Programming asynchronous communications

For general information, consult Microsoft's Win32 documentation. The details below simply concern :

- ☞ cases in which the driver differs from Win32 serial communications specifications,
- ☞ cases in which the Win32 documentation is vague,
- ☞ cases where the specification is not easily understood.

### File services

Service	Comments
CreateFile	after <b>CreateFile</b> , the transmit and receive buffers are empty.
WriteFile	the number of outgoing characters must be less than or equal to 8,192 bytes.  <b>WriteFile</b> ends when the outgoing characters are in the board buffer, and not when they are actually transmitted.
CloseHandle	<b>CloseHandle</b> ends by deleting the transmit buffer and opening circuits 105 and 108 (RTS and DTR). It is advisable to use <b>FlushFileBuffers</b> before <b>CloseHandle</b> to ensure that all the data in the buffer has been transmitted.

### Serial communication services.

Service	Comments
BuildCommDCB	<i>unrestricted.</i>
BuildCommDCBAndTimeouts	Same comments as for SetCommTimeout.
ClearCommBreak	No effect, the BREAK ends automatically after one second.
ClearCommError	Only the CE_BREAK, CE_FRAME, CE_OVERRUN, CE_RXPARITY bits in <i>lpdwErrors</i> are supported. In the COMSTAT structure, only the <i>cbInQue</i> and <i>cbOutQue</i> fields are supported.
EscapeCommFunction	CLRDTR, CLRRTS, SETDTR, SETRTS, SETBREAK are supported. CLRBREAK : see ClearCommBreak. SETXON, SETXOFF are not supported.
GetCommMask	<i>unrestricted.</i>
GetCommModemStatus	see "Appendix : limitations and différences with the COM ports.", page 48.

---

GetCommProperties	<p><i>dwMaxBaud</i> : BAUD_USER, because any speed supported to within 1% is accepted.</p> <p><i>dwProvSubType</i> : RS232, even though these boards also support the RS422, RS485, current loop types if they have the necessary options.</p> <p><i>dwMaxTxQueue</i>, <i>dwMaxRxQueue</i> : total number of bytes effectively allocated to the channel buffers.</p> <p>Support for the other fields is standard.</p>												
GetCommState	Returns the values specified for SetCommState (except <i>fParity</i> ).												
SetCommState	<p>The value RTS_CONTROL_TOGGLE for <i>fRtsControl</i>, and the <i>fBinary</i> and <i>EofChar</i> fields are not supported. With the Multiprotocol options, the other fields are supported normally. With the basic software, the driver also ignores the fields below and uses the fixed value indicated :</p> <table border="0" style="margin-left: 40px;"> <tr> <td><i>fDsrSensitivity</i></td> <td>FALSE</td> </tr> <tr> <td><i>fTXContinuesOnXoff</i></td> <td>TRUE</td> </tr> <tr> <td><i>fErrorChar</i></td> <td>FALSE</td> </tr> <tr> <td><i>fNull</i></td> <td>FALSE</td> </tr> <tr> <td><i>XonLim</i>, <i>XoffLim</i></td> <td>128, 20</td> </tr> <tr> <td><i>fInX</i>, <i>fOutX</i>, <i>fOutxDsrFlow</i>, <i>fOutxCtsFlow</i>, <i>fDtrControl</i>, <i>fRtsControl</i></td> <td>limited support : see “Appendix : flow control.”, page 46</td> </tr> </table>	<i>fDsrSensitivity</i>	FALSE	<i>fTXContinuesOnXoff</i>	TRUE	<i>fErrorChar</i>	FALSE	<i>fNull</i>	FALSE	<i>XonLim</i> , <i>XoffLim</i>	128, 20	<i>fInX</i> , <i>fOutX</i> , <i>fOutxDsrFlow</i> , <i>fOutxCtsFlow</i> , <i>fDtrControl</i> , <i>fRtsControl</i>	limited support : see “Appendix : flow control.”, page 46
<i>fDsrSensitivity</i>	FALSE												
<i>fTXContinuesOnXoff</i>	TRUE												
<i>fErrorChar</i>	FALSE												
<i>fNull</i>	FALSE												
<i>XonLim</i> , <i>XoffLim</i>	128, 20												
<i>fInX</i> , <i>fOutX</i> , <i>fOutxDsrFlow</i> , <i>fOutxCtsFlow</i> , <i>fDtrControl</i> , <i>fRtsControl</i>	limited support : see “Appendix : flow control.”, page 46												
GetCommTimeouts	<i>unrestricted</i> .												
PurgeComm	<i>unrestricted</i> .												
SetCommBreak	The duration of the BREAK is predefined (set to 1 second).												
SetCommMask	For EV_RING and EV_DSR, see “Appendix : limitations and differences with the COM”, page 48. EV_RX80FULL does not produce the expected results and EV_RXCHAR can be 100 ms late relative to the event.												
SetCommTimeouts	<i>ReadIntervalTimeout</i> is supported with an error of 25% + 100 ms.												
SetupComm	Non-standard buffer sizes are ignored. An error may be indicated in some cases (see the section on the configuration of advanced properties, page 15). The standard sizes are <b>8,192</b> characters per channel for <b>transmit buffers</b> , and <b>512</b> characters per channel for <b>receive buffers</b> .												
TransmitCommChar	Not supported.												
WaitCommEvent	<i>unrestricted</i> .												

## I.2 Programming synchronous communications

For general information, consult Microsoft™'s Win32 communications documentation. . The information below simply discusses :

- ☞ cases where the Win32 serial communication API is inappropriate,
- ☞ cases where the driver differs from the Win32 specifications,
- ☞ cases in which the Win32 documentation is vague.

### File services

Service	Comments
CreateFile	after <b>CreateFile</b> , the transmit and receive buffers are empty.
ReadFile	<b>ReadFile</b> always returns a maximum of one frame. If the frame exceeds the requested length, only the first part will be returned and the rest lost. If the frame is shorter than the requested length it is returned in full and the requested buffer is not completely filled (even if another frame is already present in the board buffer : it will be provided with the next <b>ReadFile</b> ). If the frame contains an error (ABORT, CRC, etc.) <b>ReadFile</b> returns an error (see also the compatibility options in the section on installation).
WriteFile	each <b>WriteFile</b> creates a frame whose size is limited to the value specified by the IOCTL_SERIAL_SET_SYNC_STATE command. <b>WriteFile</b> ends as soon as the outgoing characters are no longer in the board's buffer and not when they are effectively transmitted.
CloseHandle	<b>CloseHandle</b> ends by deleting the transmit buffer and opening circuits 105 and 108 (RTS and DTR). It is advisable to use <b>FlushFileBuffers</b> before <b>CloseHandle</b> to ensure that all the data in the buffer has been transmitted.
FlushFileBuffers	<b>FlushFileBuffers</b> ends with the transmission of the flag of the last transmitted frame (by the immediately preceding <b>WriteFile</b> ). This guarantees that the board's buffers are empty.

### Serial communication services.

Service	Comments
BuildCommDCB	<i>unrestricted.</i>
BuildCommDCBAndTimeouts	See the comments for SetCommTimeout.
ClearCommBreak	No effect. The BREAK ends automatically after one second.
ClearCommError	Only the CE_BREAK, CE_FRAME, CE_OVERRUN, and RXPARITY bits in <i>lpdwErrors</i> are supported. In the COMSTAT structure, only the <i>cbInQue</i> and <i>cbOutQue</i> fields are supported; they specify a number of frames and not a number of characters.

EscapeCommFunction	CLRDTR, CLRRTS, SETDTR, SETRTS, SETBREAK are supported. CLRBREAK : see ClearCommBreak(). SETXON and SETXOFF are not supported.
GetCommMask	<i>unrestricted.</i>
GetCommModemStatus	see “Appendix : limitations and différences with the COM ports.”, page 48.
GetCommProperties	<i>dwMaxBaud</i> : BAUD_USER, because any speed supported to within 1% is accepted.  <i>DwProvSubType</i> : RS232, even though these boards also support the RS422, RS485, current loop types if they have the necessary options.  <i>dwMaxTxQueue</i> , <i>dwMaxRxQueue</i> : total number of frames effectively allocated to the channel buffers.  Support for the other fields is standard.
GetCommState	Returns the values specified for SetCommState (except <i>fParity</i> ).
SetCommState	The only fields supported are <i>BaudRate</i> , <i>fOutxCtsFlow</i> , <i>fOutxDsrFlow</i> , <i>fDsrSensitivity</i> , <i>fDtrControl</i> , <i>fRtsControl</i> , <i>ByteSize</i> and <i>Parity</i> (the last two only in some protocols).
GetCommTimeouts	<i>unrestricted.</i>
PurgeComm	<i>unrestricted.</i>
SetCommBreak	The duration of the BREAK is predefined (set to 1 second).
SetCommMask	The EV_RXFLAG event is not supported. For EV_RING and EV_DSR, see “Appendix : limitations and différences with the COM ports.”, page 48.
SetCommTimeouts	<i>ReadIntervalTimeout</i> is ignored, except the value MAXDWORD. Otherwise, it should be initialised to zero to prevent any future incompatibility.
SetupComm	No effect.
TransmitCommChar	Not supported.
WaitCommEvent	<i>unrestricted.</i> EV_RXCHAR signals the arrival a complete incoming frame.

### Driver-specific services.

Service	Comments
DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)	Sets the protocol (synchronous, asynchronous, synchronised asynchronous) and the associated options. For a full description, see page 37, and for an example of how to use the command see the program source "a:\winnt\-sdk\msvc\utils\mcxmode.c".

### I.3 Programming the LAPB (or HDLC-ABM) protocol.

In addition to the specific details provided below on the programming of a LAPB connection, see the previous section, describing synchronous connections in general.

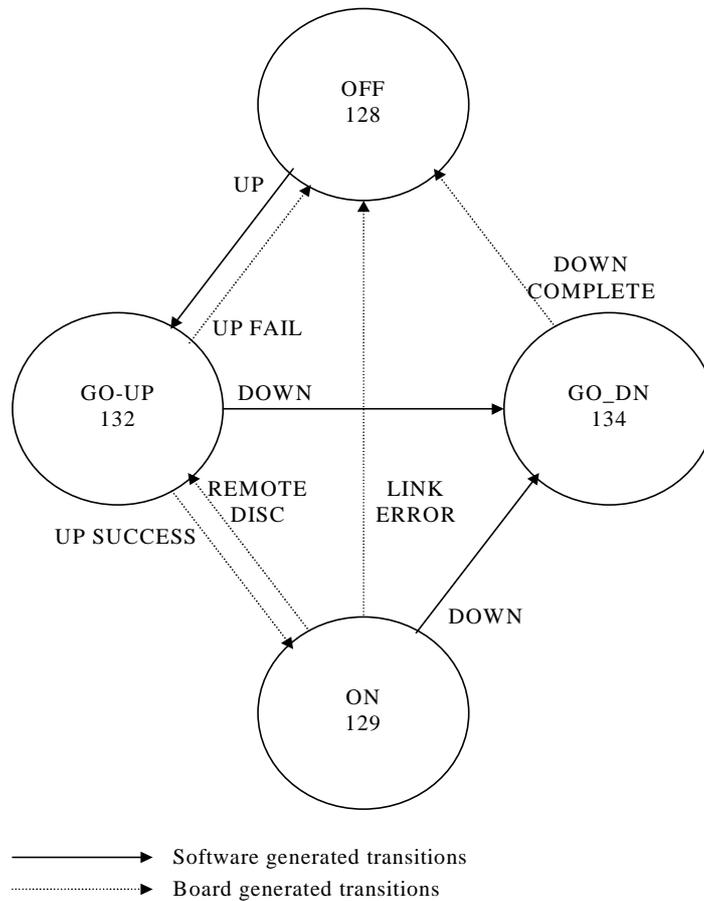
The informations below are necessary to program a LAPB link. Note that the **MCXMODE** utility has some limited capabilities for managing the Data Link layer. These can be used, in particular, in the test phase. See the directory **WINNT\SDK\MSVC\LIB** for programming samples.

#### Primitives linked to the use of LAPB.

Service	Comments
CreateFile	Keeps the current protocol, the electrical interface and the transmission speed. Reactivates the signal management mode. Purges the receive buffer. <b>CreateFile</b> does not attempt to create the connection.
CloseHandle	Purges the transmit buffer. In LAPB mode, the receiver is not inhibited. Protocol frames can thus be received and answered. <b>CloseHandle</b> does not attempt to break the connection..
ReadFile	Reads a frame in the board buffer or awaits the arrival of an information frame. Returns <b>ERROR_HANDLE_EOF</b> if the link is disconnected or broken during the wait. <b>ReadFile</b> is subject to the timeouts set by <b>SetCommTimeouts</b> .
WriteFile	Places a frame in the transmit buffer and waits if the buffer is full. Returns <b>ERROR_NOT_READY</b> if the link is disconnected or broken during the wait. <b>WriteFile</b> is subject to the timeouts set by <b>SetCommTimeouts</b> .
SetCommMask/GetCommMask/WaitCommEvent	The <b>EV_MCXLAPB</b> event (the equivalent of <b>EV_EVENT1</b> in Win32) can be processed. It is triggered each time an interrupt of the LAPB state is received (see list of states below).
DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)	Sets the protocol and the associated options. See the “Reference Manual” section.
DeviceIoControl(...,IOCTL_SERIAL_CMD_AUTO,...)	Sends a command not directly supported by the Win32 interface. . In particular, the <b>PRCTL</b> command enables the state of the protocol to be checked with the <b>ABMLINKUP</b> , <b>ABMLINKDN</b> , and <b>ABMSTATE</b> options. See the sample files and the detailed reference in section 41.

### Driver states

A LAPB channel can be in one of the 4 states shown in the diagram below :



New state	Triggered by
MCX_LINK_OFF (down, disconnected)	<ul style="list-style-type: none"> <li>• DeviceIoControl(SET_SYNC_STATE) switching to LAPB protocol</li> <li>• DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKDN command (if the disconnection occurs after the command has finished)</li> <li>• State interrupt, failure of an attempted connection by PRCTL ABMLINKUP</li> <li>• State interrupt, disconnection following a fatal transmission error</li> <li>• State interrupt, successful disconnection request by PRCTL ABMLINKDN</li> </ul>
MCX_LINK_GO_UP (connection in progress)	<ul style="list-style-type: none"> <li>• DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKUP command</li> <li>• State interrupt, temporary disruption following a recoverable error</li> </ul>
MCX_LINK_ON (up, connected)	<ul style="list-style-type: none"> <li>• DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKUP command (if the connection occurs before the end of the command)</li> <li>• State interrupt, reception of a request or acknowledgement of a connection</li> </ul>
MCX_LINK_GO_DN (break in progress)	<ul style="list-style-type: none"> <li>• DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKDN command</li> </ul>

### Effect of state changes

No action is taken when the state change is caused by the application. The table below summarises the action taken when a state change is interrupted.

New state	Action
MCX_LINK_OFF	EV_MCXLAPB signalled if specified by an earlier SetCommMask Any ReadFile will return 0 characters and the ERROR_HANDLE_EOF error Any WriteFile will return the ERROR_NOT_READY error.
MCX_LINK_ON	EV_MCXLAPB signalled if specified by an earlier SetCommMask

### Current status information

There are three ways to obtain information on the state of the connection.

- The first, imprecise, consists of assuming that the PRCTL/ABMLINKUP command always switches to the MCX\_LINK\_ON state, that PRCTL/ABMLINKDN always switches to the MCX\_LINK\_OFF state, and detecting any disconnections that may occur by analysing the errors returned by ReadFile and WriteFile. With this method a LAPB channel can be used as a filter with the system's standard commands, by establishing and breaking the connection with the MCXMODE utility.
- The second method consists of using the PRCTL/ABMSTATE to query the channel.
- The third method consists of using WaitCommEvent to instantly detect state changes that have been caused by the board or by the other end of the connection. As a rapid sequence of events can result in just one wake-up of WaitCommEvent, it is advisable to systematically query the channel state after a wake-up, and to consider that some steps in the sequence of states can be ignored. Note that the events are memorised as soon as they are selected by SetCommMask, and that, subsequently, one or more events occurring between two consecutive WaitCommEvent services cannot be lost (but may be combined into one event).

All three sources of information should be used in a complete process.

## I.4 Programming driver-specific services.

The Win32 **DeviceIoControl()** function has been extended to dialogue directly with the software installed on the board. This makes it possible to send the commands manually, i.e. bypassing the driver. For details of these commands, see the "MCX board basic software user manual" and the "MCX board multiprotocol software user manual". For a full description, see page 41, and for an example of how to use the command, see the program sources "a:\winnt\sdks\msvc\utils\mccioctl.c" and "a:\winnt\sdks\msvc\special\addfunc.c".

## I.5 Standard Windows NT utilities.

If you just want to use the standard Windows NT utilities with your MCX boards, it is advisable to select the name “COM” in the installation procedure.

### Control panel

Do not use the “ports” icon to create the COM channels for MCX boards because the driver creates these names automatically. However, you can use this dialogue to change the transmission parameters (this is useful if a channel is used by the Print Manager). Moreover, you should not try to use this dialogue to define addresses or interrupts.

### HyperTerminal.exe

This Windows NT 4 accessory works normally with this driver.

### Terminal.exe

This Windows NT 3.x accessory only supports the COM1 thru COM9 ports.

### Mode.exe

This DOS utility only supports the names that begin with COM (and, in Windows NT 3.1, only COM1 thru COM9). It can only be used to modify asynchronous transmission modes. Our **mcxmode.exe** utility can be used to work around these restrictions (see the description of **mcxmode.exe**).

### Remote Access Services

This communication system has been successfully tested from version 1.6.3 of the driver upwards, in single-channel server mode on Windows NT 3.51 Workstation. The **Windows NT 4.0** dialling interface **does not work** with this driver.

### Serial printers

Because of a fault in the Windows NT 3.51 Print Manager, you should proceed as follows to create a printer :

- Use the “Ports” icon in the “Control Panel” to modify or at least display the parameters for the particular port. You must use the <OK> button to validate.
- Use the “Services” icon in the “Control Panel”: stop the service called “Spooler” and restart it immediately. This will recognise the new ports and its parameters.
- “Printers” icon in the “Control Panel”: in the Print Manager, go to the “Printer” menu and select <Create a printer...>. Fill in the form; the port should appear in the <Print to:> list. The installation procedure ends here.

### Command.exe (command line prompt in the DOS window)

The names COM1 thru COM9 can be used directly here. Otherwise, use the standard names `\\.\name-assigned-on-installation`, for example, to redirect console output :

```
C:> DIR > COM9           } Either format may
C:> DIR > \\.\COM9       } be used
C:> DIR > \\.\COM10      This is the only format authorised if the installed
                        name is COM
C:> DIR > \\.\MCX101     This is the only format authorised if the name is
                        MCX1
```

## I.6 Other utilities.

The **MCXMODE** utility can be used to customise the transmission characteristics in synchronous or asynchronous mode. The **MCXMODE** command, used without parameters, displays on-line help.

The **MCCIOCTL** utility can be used to send the command as an argument directly to the board. The **MCCIOCTL** command, entered without parameters, displays on-line help.

The **DOSDEV** utility manages the links between the names of Windows NT objects and DOS peripherals. It remains effective until the system is shut down. The syntax is as follows :

```
dosdev                                lists the aliases.
dosdev -l COMn \Device\McxCNN         creates the COMn alias for the McxCNN device.
dosdev -r COMn                         deletes the COMn alias.
```

(where *n* is the number of the COM port, *C* is the number of the MCX board and *NN* is the number of the channel on the board).

The **SETMCX** utility can be used to display or modify a board's parameters from the command line or in a "batch" process :

```
SETMCX                                on-line help
SETMCX n                               displays all the parameters of board n
SETMCX n param                          displays the value of the param parameter for board n
SETMCX n param val                       assigns the value val to the param parameter for board n
SETMCX n Compatibility +code -code...    adds/removes the code bit to/from the configuration options
```

## II DETAILED REFERENCE MANUAL.

The driver-specific functions can be accessed via **DeviceIoControl()**. They use definitions and structures described in “**mcc\_mcx.h**” or in “**mcxproto.h**” for functions that are specific to Multiprotocol mode.

### II.1 Extract from the **mcc\_mcx.h** file

```
#include "mcc_mcx.h"
```

```
/* serial IOCTL codes for Windows NT */
#if defined(CTL_CODE) && defined(FILE_DEVICE_SERIAL_PORT)

#define MCX_IOCTL(code) \
        CTL_CODE(FILE_DEVICE_SERIAL_PORT,code, \
                METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_SERIAL_GET_SYNC_STATE    MCX_IOCTL(0x901)
#define IOCTL_SERIAL_SET_SYNC_STATE    MCX_IOCTL(0x902)
#define IOCTL_SERIAL_CMD                MCX_IOCTL(0x903)
#define IOCTL_SERIAL_CMD_AUTO           MCX_IOCTL(0x904)
#define IOCTL_SERIAL_ACCESS_AREA        MCX_IOCTL(0x90A)
#define IOCTL_SERIAL_MCX_OPTIONS        MCX_IOCTL(0x90B)

#endif /* Windows NT */

-----

/* macros and structs for CMD & CMD_AUTO */
typedef struct mcc_cmd {
    unsigned char opcode;
    unsigned char status;
    unsigned char par[76];
    unsigned char ichan;
    unsigned char icond;
    unsigned char ipar1;
    unsigned char ipar2;
    unsigned char ipar3;
    unsigned char padding1;
    unsigned char *data;
    unsigned char *kdata;
    unsigned short length;
    unsigned short padding2;
}mcc_cmd;

typedef struct _MCC_CMD {    /* buffer for ioctls CMD... */
    mcc_cmd Cb;
    unchar Data[1];
} MCC_CMD, *PMCC_CMD;

/* size assigned to contain the struct _MCC_CMD */
#define MCX_DIRECT_IO_BUFFER_SIZE(datalen)
    ((datalen)+sizeof(mcc_cmd))
-----
```

```

-----
typedef struct _MCX_SYNCHRONIZATION_PARAMETERS {
    unchar SynchronousMode;          /* protocol */
#define MCX_SYNC_CHAR          0    /* NOT synchronous */
#define MCX_SYNC_BISYNC       2
#define MCX_SYNC_HDLC         4
#define MCX_SYNC_LAPB         5    /* LAPB, HDLC/ABM */
    unchar Duplex;                   /* flag version+2 bits duplex */
#define MCX_WAY_VERSION        0x80 /* validity bit Version field */
#define MCX_WAY_FULLDUPLEX    0x80 /* full duplex + version */
#define MCX_WAY_HALFDUPLEX    0x81 /* low RTS in each frame */
/*----- horloges ----- */
/*      ETTD      ETCD NULL-MODEM */
    unchar TransmitClockSource;      /* TXCI      BRG      BRG      */
    unchar ReceiveClockSource;       /* RXC      BRG      RXC      */
    unchar TxClockPinSource;         /* TRXC_HIGH BRG      BRG      */
#define MCX_CLOCK_RXC          0    /* modes for TransmitClockSource... */
#define MCX_CLOCK_TXCI        1    /* ... and ReceiveClockSource
    */
#define MCX_CLOCK_TRXC_HIGH    0    /* TxClockPinSource = always high      */
#define MCX_CLOCK_TXCLOCK      1    /*      = copy of TransmitClockSource    */
#define MCX_CLOCK_BRG          2    /* common modes                          */
#define MCX_CLOCK_DPLL         3
    unchar MonosyncChar;
    unchar BisyncChar;
    unchar Version;                  /* valid if Duplex = MCX_WAY... */
    McxUnshort Options;              /* default: 0 */
#define MCX_HDLC_USERDTR       1    /* slows HDLC but enables DTR          */
#define MCX_HDLC_SPECS         2    /* Use the Protocol.HdLC struct below   */
#define MCX_BISYNC_SPECS       2    /* Use the Protocol.Bisync struct below  */
    McxUnshort DataLength;           /* max. frame size (LAPB N1)          */
    union{
        struct{
            unchar RxFrames;         /* default: 14 */
            unchar TxFrames;         /* default: 4 */
            McxUnshort Spare1;       /* reserved, set to zero */
            McxUnshort Spare2;       /* reserved, set to zero */
            McxUnshort Spare3;       /* reserved, set to zero */
            McxUnshort Spare4;       /* reserved, set to zero */
            McxUnshort Spare5;       /* reserved, set to zero */
        }HdLC, Bisync;
#define MCX_HDLC_DEFAULT        0    /* default for theses fields */
        struct{
            unchar Role;              /* default: CLIENT */
#define MCX_ROLE_CLIENT        1
#define MCX_ROLE_NETWORK      3
            unchar K;                 /* default: 7 */
            McxUnshort N2;           /* default: 10 essais */
            McxUnshort T1;           /* default: 2550 ms */
            McxUnshort T2;           /* default: 0 */
            McxUnshort T3;           /* default: infinite */
            McxUnshort Spare;        /* reserved, set to zero */
#define MCX_LAPB_DEFAULT        0    /* default for theses fields */
        }Lapb;
    }Protocol;
} MCX_SYNCHRONIZATION_PARAMETERS, *PMCX_SYNCHRONIZATION_PARAMETERS;
-----

```

```

/* structure for IOCTL_SERIAL_ACCESS_AREA */
typedef struct _MCX_AREA_DESCRIPTOR {
    long Operation;           /* combin.of the following flags */
#define MCX_AREA_GET          0 /* board to application */
#define MCX_AREA_SET          1 /* application to board */
#define MCX_AREA_MEMORY      0 /* acces to mailbox */
    long StartAddress;       /* starting address of access */
                                /* relative to the base port or */
                                /* the start of the mailbox */
    long Length;             /* length to transfer */
    unchar Buffer[1];        /* values to write if MCX_AREA_SET */
} MCX_AREA_DESCRIPTOR, *PMCX_AREA_DESCRIPTOR;
#define MCX_AREA_DESCRIPTOR_SIZE(dlen) \
    ((dlen)+sizeof(MCX_AREA_DESCRIPTOR)-1)

-----

/* structure for IOCTL_SERIAL_SET_OPTIONS */
typedef struct _MCX_OPTION {
    long Option;             /* option code */
#define MCX_OPTION_GET_CHANNEL      0x20002 /* get channel n° */
#define MCX_OPTION_SET_DSR_RI_INVERSION 0x30100 /* exchange DSR/RING */
#define MCX_OPTION_GET_DSR_RI_INVERSION 0x40001 /* get DSR/RING state */
#define MCX_OPTION_GET_COMPATIBILITY 0x50004 /* get current Compatibility */
    union{
        long Long[1];
        short Short[1];
        unchar Char[1];
    }Value;                 /* parameters used by option */
} MCX_OPTION, *PMCX_OPTION;
#define MCX_OPTION_SIZE(dlen) \
    ((dlen)+sizeof(MCX_OPTION)-sizeof(long))

```

## II.2 SET/GET SYNC STATE functions

Two functions have been added to customise the format of synchronous frames.

```
#include "windows.h"
#include "winiocctl.h"
#include "mcc_mcx.h"
```

```
DeviceIoControl(hDevice, IOCTL_SERIAL_SET_SYNC_STATE,
               frameFormatBuffer, sizeof(MCX_SYNCHRONIZATION_PARAMETERS),
               NULL, 0, lpcbBytesReturned, lpoOverlapped )
```

```
DeviceIoControl(hDevice, IOCTL_SERIAL_GET_SYNC_STATE, NULL, 0,
               frameFormatBuffer, sizeof(MCX_SYNCHRONIZATION_PARAMETERS),
               lpcbBytesReturned, lpoOverlapped )
```

```
HANDLE hDevice; /* Handle of the device */
PMCX_SYNCHRONIZATION_PARAMETERS frameFormatBuffer; /* pointer to parameters */
LPDWORD lpcbBytesReturned; /* size of returned params */
LPOVERLAPPED lpoOverlapped; /* overlapped struct. addr */
```

The SET function can be used to select the frame format : HDLC, etc. The integer pointed to by *lpcbBytesReturned* always takes the value 0. **WARNING** : in driver versions earlier than 1.8.3, this function returns an error if the state of the port set previously by SetCommState contains options not supported by the board (e.g. *ByteSize=7* in HDLC mode). In the later versions, the port is forced to a “reasonable” state.

The GET function consults the current frame parameters. The integer designated by *lpcbBytesReturned* always takes the value `sizeof(MCX_SYNCHRONIZATION_PARAMETERS)`.

The `MCX_SYNCHRONIZATION_PARAMETERS` structure is made up of the following items :

```
UCHAR SynchronousMode; Protocol: MCX_SYNC_HDLC, MCX_SYNC_BISYNC,
MCX_SYNC_LAPB or MCX_SYNC_CHAR. The
MCX_SYNC_CHAR mode corresponds to asynchronous
transmissions. LAPB is also known as HDLC/ABM.
```

```
UCHAR Duplex; Simultaneous transmission : MCX_WAY_FULLDUPLEX
(simultaneous transmission and reception), or
MCX_WAY_HALFDUPLEX (alternating transmission and
reception ; see the details of this mode in the Multiprotocol
Manual [DT003]).
```

**Important** : most of the functionalities used in half-duplex mode can be activated by correctly configuring the full-duplex mode. Half-duplex mode should only be used when it is really necessary (it is the only way to prevent transmission when reception is in progress).

Half-duplex mode forces *fOutxCtsFlow* = TRUE, *fRtsControl* = RTS\_CONTROL\_TOGGLE, and ignores data received without DCD. It prohibits transmission when DCD is active and also when DSR is active with *fDsrSensitivity* = TRUE.

**UCHAR** *TransmitClockSource*; Transmit clock source (see below).

**UCHAR** *ReceiveClockSource*; Receive clock source (see below).

Constant	clock source
<b>MCX_CLOCK_RXC</b>	pin 17 (RxClock)
<b>MCX_CLOCK_TXCI</b>	pin 15 (TxClock)
<b>MCX_CLOCK_BRG</b>	internal bauds generator
<b>MCX_CLOCK_DPLL</b>	decoded in data (only with FM or Manchester coding)

**UCHAR** *TxClockPinSource*; Clock source available on pin 24. Warning : this pin is disabled if **MCX\_CLOCK\_TXCI** is used (by *TransmitClockSource* on *ReceiveClockSource*).

Constant	clock source
<b>MCX_CLOCK_TRXC_HIGH</b>	none (pin set to MARK state)
<b>MCX_CLOCK_TXCLOCK</b>	like <i>TransmitClockSource</i>
<b>MCX_CLOCK_BRG</b>	internal bauds generator
<b>MCX_CLOCK_DPLL</b>	decoded in data (only with FM or Manchester coding)

**UCHAR** *MonosyncChar*; First sync character in BISYNC mode. The only sync character in MONOSYNC mode. Ignored in the other modes.

**UCHAR** *BisyncChar*; Second sync character in BISYNC mode. Ignored in the other modes.

**UCHAR** *Version*; Version of the structure. Should always = 1.

**USHORT** *Options*; Protocol options. Each option is a bit that must be added if the option is to be used.

- **MCX\_HDLC\_USERDTR** enables the use of the CCITT 108 (DTR) circuit in HDLC, LAPB, and X25 modes on channels 1, 2, and 3 of the MCX and MCX-Lite/S boards. However, this will limit the performance levels<sup>2</sup>.
- **MCX\_HDLC\_SPECS** forces the use of the elements of the *Protocol.Hdlc* structure, which are ignored otherwise.
- **MCX\_BISYNC\_SPECS** forces the use of the elements of the *Protocol.Bisync* structure, which are ignored otherwise.

**USHORT** *DataLength*; Maximum frame length. If the value is set to 0, the default length will be used (see the PROTO command in the Multiprotocol documentation [DT003]).

**union** {...} *Protocol*; The sub-structures specified here can be used to specify the parameters for a specific protocol.

<sup>2</sup> See the PROTO documentation in the "Multiprotocol Software User Manual" [DT003]

---

The following should be specified for LAPB only :

**UCHAR Protocol.Lapb.Role;** **MCX\_ROLE\_CLIENT** if the application acts as a client (ETTD), **MCX\_ROLE** if the application serves the network.

**UCHAR Protocol.Lapb.K,N2,T1,T2,T3**

Standardised LAPB parameters. **MCX\_LAPB\_DEFAULT** will invoke the default value<sup>3</sup>.

**USHORT Protocol.Lapb.Spare;** Zone reserved for LAPB.

The following elements should be specified for HDLC only :

**UCHAR Protocol.Hdlc.RxFrames;**

Number of frames acceptable in reception mode without risk of loss if the PC does not read them immediately from the board (number of frame receive buffers on the board). The value 0 resolves to the default value<sup>3</sup>.

**UCHAR Protocol.Hdlc.TxFrames;**

Number of frames that the board can memorise as awaiting transmission (number of frame transmit buffers on the board). The **WriteFile()** function will never halt processing if a frame buffer is available on the board at the time of the call. The value 0 resolves to the default value<sup>3</sup>.

**USHORT Protocol.Hdlc.Spare1; à Spare5;**

Zones reserved for HDLC.

The following elements should be specified for BISYNC only :

**UCHAR Protocol.Bisync.RxFrames ;**

Number of frames acceptable in reception mode without risk of loss if the PC does not read them immediately from the board (number of frame receive buffers on the board). The value 0 resolves to the default value<sup>3</sup>.

**UCHAR Protocol.Bisync.TxFrames ;**

Number of frames that the board can memorise as awaiting transmission (number of frame transmit buffers on the board). The **WriteFile()** function will never halt processing if a frame buffer is available on the board at the time of the call. The value 0 resolves to the default value<sup>3</sup>.

**USHORT Protocol.Bisync.Spare1 à Spare5 ;**

Zones reserved for BISYNC.

---

<sup>3</sup> See the PROTO documentation in the "Multiprotocol Software User Manual" [DT003]

### II.3 Example of SET\_SYNC\_STATE

```
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

Proto(HANDLE chan, int fonc)
{
    MCX_SYNCHRONIZATION_PARAMETERS sp;
    DWORD count;
    DWORD speed;

    //
    // switch to HDLC mode with the appropriate clocks
    // for a NULL-MODEM cable (internal clocking)
    //
    sp.SynchronousMode = MCX_SYNC_HDLC;
    sp.Version = 1;
    sp.Duplex = MCX_WAY_FULLDUPLEX;
    sp.Options = 0;
    sp.DataLength = 0; /* default = 1,024 bytes */
    sp.TransmitClockSource = MCX_CLOCK_BRG;
    sp.ReceiveClockSource = MCX_CLOCK_RXC;
    sp.TxClockPinSource = MCX_CLOCK_BRG;

    if(!DeviceIoControl(
        chan, IOCTL_SERIAL_SET_SYNC_STATE,
        &sp, sizeof(sp), NULL, 0, &count, NULL)) {
        printf("SET_SYNC_STATE Ioctl: error %d\n",
            GetLastError());
        exit(1);
    }
}
```

## II.4 CMD and CMD\_AUTO functions

Two communication functions have been added to enable manual dialogue with the on-board interpreter. For a description of the commands, their parameters and data zone, consult the appropriate manual for the “firmware” you are using (basic software [DT002] or multiprotocol software [DT003]).

```
#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"
```

```
DeviceIoControl( hDevice, IOCTL_SERIAL_CMD,
                paramsFromAppToBoard, paramsToSize,
                paramsFromBoardToApp, paramsFromSize, lpcbBytesReturned,
                lpoOverlapped
                )
```

```
DeviceIoControl( hDevice, IOCTL_SERIAL_CMD_AUTO,
                paramsFromAppToBoard, paramsAppSize,
                paramsFromBoardToApp, paramsBoardSize, lpcbBytesReturned,
                lpoOverlapped
                )
```

<b>HANDLE</b> <i>hDevice</i> ;	/* Handle of the device */
<b>PMCC_CMD</b> <i>paramsFromAppToBoard</i> ;	/* pointer to sent parameters */
<b>DWORD</b> <i>paramsAppSize</i> ,	/* size of sent params */
<b>PMCC_CMD</b> <i>paramsFromBoardToApp</i> ;	/* pointer to returned parameters */
<b>DWORD</b> <i>paramsBoardSize</i> ,	/* size of space for returned params */
<b>LPDWORD</b> <i>lpcbBytesReturned</i> ;	/* size of returned params */
<b>LPOVERLAPPED</b> <i>lpoOverlapped</i> ;	/* overlapped struct. addr */

The **IOCTL\_SERIAL\_CMD** function can make the board execute any command.

The **IOCTL\_SERIAL\_CMD\_AUTO** function can make the board execute any command with parameter 1 being initialised by the driver, and the channel number corresponding to *hDevice*.

The **MCC\_CMD** type is a structure that matches that of the board’s mailbox. This enables the application to transmit and receive parameters and data.

When a command is sent to the board by these functions, the driver executes one of the following actions :

- 1) If it exists, the *paramsFromAppToBoard→Data* table is copied into the DATA zone of the board’s mailbox,
- 2) the *paramsFromAppToBoard→Cb.par[]* table is copied into the mailbox’s PARAMETERS zone,
- 3) *paramsFromAppToBoard→Cb.opcode* is copied into the mailbox’s OPCODE zone,
- 4) the binary value 0000 0001 is written into the mailbox’s VALIDATION byte. This causes the board to execute the command. The board then issues an end of command interrupt that enables the driver to continue processing,
- 5) the mailbox’s STATUS zone is copied into *paramsFromBoardToApp→Cb.status*

- 6) the PARAMETERS zone in the mailbox is copied into *paramsFromBoardToApp*→*Cb.par*[],
- 7) if the *paramsFromAppToBoard*→*Data* exists, table exists, the mailbox's DATA zone is copied into *paramsFromBoardToApp*→*Data*,
- 8) the application is woken up or alerted depending on the *lpoOverlapped* value.

The length of the **MCC\_CMD** *Data* field is variable. For example, the structure can be created by dynamically allocating (sizeof(**MCC\_CMD**)+*data\_length*) bytes; the **MCX\_DIRECT\_IO\_BUFFER\_SIZE**(*datalen*) macro can be used to calculate the required number of bytes. The driver recognises the existence of the *Data* field if the *Cb.length* field is non-null. Otherwise, it assumes that the command that is to be executed does not use the mailbox's DATA zone.

*paramsAppSize* must be equal to the sum of sizeof(**MCC\_CMD**) and the length of the data zone.

If the command has been executed correctly, the integer designated by *lpcbBytesReturned* will still equal *paramsBoardSize*.

The **\_MCC\_CMD** structure contains the following elements :

<b>unsigned char</b> <i>Cb.opcode</i> ;	Code of the command that is to be executed. <b>mcc_mcx.h</b> defines the symbolic names for these codes.
<b>unsigned char</b> <i>Cb.status</i> ;	The returned result supplied by the board in the STATUS zone.
<b>unsigned char</b> <i>Cb.par</i> [76];	The command parameters; returned parameters for some commands.
<b>unsigned char</b> <i>Cb.ichan</i> ;	}
<b>unsigned char</b> <i>Cb.icond</i> ;	} Returns a copy of the mailbox INTERRUPT
<b>unsigned char</b> <i>Cb.ipar1</i> ;	} zone. Theoretically, these five elements are not
<b>unsigned char</b> <i>Cb.ipar2</i> ;	} used <sup>4</sup> .
<b>unsigned char</b> <i>Cb.ipar3</i> ;	}
<b>unsigned char</b> * <i>Cb.data</i> ;	Unused in Windows NT <sup>5</sup> .
<b>unsigned char</b> * <i>Cb.kdata</i> ;	Field used temporarily by the driver during command execution.
<b>unsigned short</b> <i>Cb.length</i> ;	Working length of the data zone in bytes.
<b>unsigned char</b> <i>Data</i> [0...];	Data zone that will be exchanged with the board's mailbox; because this zone must immediately follow the <i>Cb</i> structure, the <b>MCC_CMD</b> variable length structure is used <sup>6</sup> .

<sup>4</sup> except for the LDIAL command used by the MCC board.

<sup>5</sup> In a UNIX environment, pointer to the data zone that will be exchanged with the board's mailbox.

<sup>6</sup> This zone is not used in the UNIX driver.

## Notes

- ☞ To avoid errors, it is advisable to include the same pointer in *paramsFromAppToBoard* and *paramsFromBoardToApp*, and the same length in *paramsAppSize* and *paramsBoardSize*.
- ☞ To ensure compatibility with the UNIX driver, we suggest initialising *paramsFromAppToBoard→Cb.data = paramsFromAppToBoard→Data*.
- ☞ To avoid confusion between *Cb.par[]* which starts at *Cb.par[0]*, and the command description which starts at PARAMETER 1, it is advisable to reference PARAMETER N by :  
*paramsFromAppToBoard→Cb.par[N-1]* to quote the PARAMETRE N.

## Interactions between these functions and normal driver operation

The description below is up-to-date for version 1.8.3 of the driver. These interactions may change in future versions.

### ALLOC :

The driver detects this command and, if necessary, adjusts the buffer size information. It can consequently replace *SetupComm()* which is ignored.

### CHDEF :

A CHDEF is executed by the driver during *SetCommState*, if an attempt is made to change *DCB.EvtChar*. In this case, only *EvtChar* is included in the CHDEF command.

### MINTR :

- This command is executed during *CreateFile*, *CloseHandle*, *DeviceIoControl* (*IOCTL\_SERIAL\_SET\_SYNC\_STATE*), *SetCommMask* (if the *EV\_RXFLAG* is activated or deactivated) and in some cases during *ReadFile*.
- The activated sources are: IT1, IT2 (bit *Mde=1*), IT3 (if *EV\_RXFLAG* is active), IT5, IT6, IT7.

### PROTO :

- The driver detects this command and adjusts its own protocol information. It can consequently replace *DeviceIoControl SET\_SYNC\_STATE* to enable the use of non-standardised parameters ; this does not disrupt driver operation.
- The *VINIT*, *RXENB*, *MINTR* and, in some cases *VMODE*, *EscapeCommFunction()* commands must then be executed to position *RTS* and *DTR*, and *PurgeComm()* to purge the buffers.

### RSMDE :

- No undesirable interaction.

## II.5 Samples of CMD and CMD\_AUTO

Set422() switches an MCX serial channel to RS422 by sending the RSMDE command to the board's firmware.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>
...
BOOLEAN Set422(HANDLE hfd)
{
    MCC_CMD cmd;
    long return_bytes;
    int lasterror;

    //
    // init opcode and parameters, see RSMDE command in
    // basic software or multiprotocol documentation
    //
    cmd.Cb.opcode = RSMDE;
    cmd.Cb.par[1-1] = 0; // Will be replaced by channel n°
    cmd.Cb.par[2-1] = 1; // Mode RS422 active
    cmd.Cb.length = 0; // No data zone required

    if( !DeviceIoControl( hfd, IOCTL_SERIAL_CMD_AUTO,
        &cmd, sizeof(cmd),
        &cmd, sizeof(cmd),
        &return_bytes, NULL) ) {
        printf("Set422: error, code %d\n", GetLastError());
        return FALSE;
    }
    return TRUE;
}
```

Relrp() runs the RELRP command on the board. This command is not linked to a particular channel. The command sends information on the board type and capabilities.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

BOOL Relrp()
{
    PMCC_CMD command; /* space for RELRP and its parameters */
    DWORD retLen;     /* length returned by DeviceIoControl */
    int cmdLen;       /* length of command structure */
    int dataLen;      /* Length of the RELRP Data area */
    HANDLE hDevice;

    // one of the channels must be used (any one will do)
    hDevice = CreateFile("\\\\.\\COM3",
                        GENERIC_WRITE|GENERIC_READ, 0, NULL,
                        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    //
    // init opcode and data, see RELRP command in
    // basic software or multiprotocol documentation
    //
    dataLen = 18;
    cmdLen = MCX_DIRECT_IO_BUFFER_SIZE(dataLen);
    command = malloc(cmdLen);
    command->Cb.opcode = RELRP;
    command->Cb.length = dataLen;
    // the following assignment distinguishes the MCC boards
    // which do not modify data item 18.
    command->Data[18-1] = MCX_TYPE_MCC;

    if(!DeviceIoControl(hDevice, IOCTL_SERIAL_CMD,
                        command, cmdLen, command, cmdLen, &retLen, NULL )){
        printf("RELRP: Win32 error %d\n", GetLastError());
        return FALSE;
    }
    if(command->Cb.status != 0){
        printf("RELRP: failed, status %d\n", command->Cb.status);
        return FALSE;
    }
    printf("RELRP: carte type %d à %d MHz, %d canaux\n",
          command->Data[18-1], command->Data[11-1],
          command->Data[9-1]);

    free(command);
    CloseHandle(hDevice);
    return TRUE;
}
```

---

Appendix : flow control.

The boards in the MCX range, equipped with the MCX-MULTIPROTOCOL option, and used with a version of the driver later than 1.7.0, **support all the flow controls offered by the Win32 API**, and even some additional controls (accessible by executing the VMODE command directly).

The restrictions affecting the operation of flow control are described below :

Methods supported

The following flow control options are supported :

- none
- XON/XOFF configurable
- DTR/CTS
- RTS/CTS

In all cases, both transmission directions are controlled. The two directions cannot be configured independently.

Configuring the control method

The *SetCommState* service and the interpretation of the fields in the *DCB* structure (see page 26) have been adapted to address these constraints. The use of the Multiprotocol option and the board configuration options (i.e. the Register *Synchronous* and *Compatibility* values) also affect operation. Flow control is consequently supported as follows :

- a) if the *Synchronous* indicator is set to 1 (with an old driver and/or old firmware), flow control will not be supported because the board does not support the VMODE command,
- b) otherwise, if *fInX* or *fOutX* is TRUE, flow control is XON/XOFF with the *XonChar* and *XoffChar* characters,
- c) otherwise, if *fRtsControl* is set to *RTS\_CONTROL\_HANDSHAKE*, flow control is implemented by hardware with the RTS and CTS signals,
- d) otherwise, if *fOutxCtsFlow* is TRUE or *fOutxDsrFlow* is TRUE or if *fDtrControl* is set to *DTR\_CONTROL\_HANDSHAKE*, flow control is implemented by hardware with the DTR and CTS signals,
- e) otherwise there is no flow control.

Compatibility with earlier versions

Earlier versions of the boards and software did not support flow control by RTS/CTS. In any of the following cases :

- the driver version is 1.6.2 or earlier,
- the “EPROM 1.8/3.8” configuration option has been validated,
- the version of the board firmware is earlier than 1.8,

Rules c) and d) must be combined into a single rule :

- c+d) otherwise, if *fOutxCtsFlow* is TRUE or *fOutxDsrFlow* is TRUE or *fDtrControl* is set to *DTR\_CONTROL\_HANDSHAKE* or *fRtsControl* is set to *RTS\_CONTROL\_HANDSHAKE*, flow control is implemented by hardware with the DTR and CTS signals,

Note

To ensure compatibility with future versions, use the combination that matches the cable you are actually using (e.g. if the incoming control signal is on CTS, use *fOutxDsrFlow* to manage it instead of *fOutxCtsFlow*).

---

### III APPENDIX : SPECIFIC ERROR CODES.

The error codes returned by the Win32 API are as described in the API documentation. Here, however, is a selected list of error codes that are not easily understood :

2 **ERROR\_FILE\_NOT\_FOUND**

The driver has not started. Consult the Event Viewer and the Device Manager.

5 **ERROR\_ACCESS\_DENIED**

The channel has already been opened by another process.

21 **ERROR\_NOT\_READY**

In **WriteFile**, transmission is impossible because the Data Link layer (LAPB) is not connected.

23 **ERROR\_CRC**

In a **ReadFile**, this error indicates that the received frame contains an error (all types of error generate this error code, and not just CRC errors).

38 **ERROR\_HANDLE\_EOF**

In LAPB protocol, indicates that the Data Link layer is disconnected before or after the execution of a **ReadFile**, and that there is no outstanding frame in the receive buffer.

57 **ERROR\_ADAP\_HDW\_ERR**

Unexpected error on the MINTR command. Probably a board malfunction.

87 **ERROR\_INVALID\_PARAMETER**

- 1) In **DeviceIoControl**, **WaitCommEvent**, **ReadFile** and **WriteFile**, this error can, in particular, indicate that the *lpoOverlapped* parameter does not match the options requested in the **CreateFile**; either that or the *hEvent* element in the OVERLAP structure is incorrect.
- 2) In **DeviceIoControl**, either one of the parameters or one of the elements in the structure supplied in the third parameter position is incorrect.

122 **ERROR\_INSUFFICIENT\_BUFFER**

- 1) The length specified in **DeviceIoControl** is wrong.
- 2) In **DeviceIoControl**, the value of the *Cb.length* element in the IOCTL\_SERIAL\_CMD or CMD\_AUTO function is too small.

995 **ERROR\_OPERATION\_ABORTED**

- 1) The board did not answer a command within the timeout. The most likely causes are : the interrupt supplying the active IRQ is not pushed in or is the wrong one ; the board is very busy, in which case the *CommandTimeout* parameter value should be increased ; or there is a constant influx of parasites on the channel.
- 2) In LAPB protocol, a **WriteFile** was attempted when the link was down.

**997 ERROR\_IO\_PENDING**

The operation has not been completed. See **GetOverlappedResult()**.

**1450 ERROR\_NO\_SYSTEM\_RESOURCES**

A **DeviceIoControl** sent to the driver has not been recognised. The likely causes are : the application sends a **DeviceIoControl** with a bad code ; the version of Windows NT is not supported.

**1784 ERROR\_INVALID\_USER\_BUFFER**

In **WriteFile**, the '*count*' parameter is too high for the size of the buffer or the frames, or exceeds the 31 Kbyte limit.

**IV APPENDIX : LIMITATIONS AND DIFFERENCES WITH THE COM PORTS.**

Several restrictions are due to the interactions between the different capabilities of the Win32 API, the driver, the firmware, and the board itself.

The board's basic software limits the available flow control types (see Appendix entitled "Flow control" above). The Multiprotocol option, though less effective in synchronous mode, is not affected by this restriction.

134.5 bauds transmission speed : the DCB does not support this speed because *BaudRate* is a LONG variable. However, the driver does support this speed via the value (ULONG)(-134).

The DSR signal (circuit 107) does not exist on all boards (see the board connector documentation). A configuration option and an API function can be used to swap the processing of this signal with RING, which enables a pseudo-DSR implementation, if an appropriate cable is used.

In synchronous mode, the RING signal (circuit 125) is only available as from release E of the MCX-BP connection package.

DTR in synchronous mode : see the Multiprotocol software manual [DT003] and the **MCX\_HDLC\_USERDTR** indicator on page 38.

Supported speeds : see the appropriate firmware manual [DT002], [DT003].



---

## **CONTACT ACKSYS**

---

Our sales department welcomes your enquiries about any aspect of our product range, from signal converters to passive boards, and will gladly offer specific technical support to help you with your own development projects. Please call us on :

01 39 11 62 81 in France or 33 1 39 11 62 81 from abroad

Our support hotline is charge-free on :

01 39 11 62 81 in France or 33 1 39 11 62 81 from abroad