SOUNDLIGHT
The DMX Company

DMXCRD32.DLL

Win32 DLL for
2512A PCMCIA DMXCard
1514PCI PCI Dual Link
2514PCI PCI Quad Link
DMX-512 PC Card interfaces

TABLE OF CONTENTS

SOUNDLIGHT

SOUNDLIGHT *The DMX Company* Glashüttenstrasse 11 D-30165 HANNOVER Germany
Tel: +49-511-3730 267 Fax: +49-511-3730 423 E-Mail: info@soundlight.de Web: www.soundlight.de

PCI DMX INTERFACE CARD 1514PCI

PCMCIA DMX INTERFACE CARD 2512A

PREFACE

This Manual has been designed to assist in programming the SOUNDLIGHT 1514 PCI interface card, and the PCMCIA interface card 2512A, using the supplied DMXCRD32.DLL. All examples given are for demonstration only, they do not necessarily show the full combined functionality nor do they represent all required steps to complete a full working project. Also, driver access is not being discussed in detail. It is assumed that the card has been installed properly and checked for functionality.

Please note that while the DMXCRD32.DLL will work for all Windows versions, including Windows 95, Windows 98/ME, Windows2000 and Windows XP. You will, however, need to install a suitable card driver for the operating system under consideration. Presently, for the PCI card, a combined driver suitable for Windows 98, 2000 and XP is available, and for the PCMCIA card two different driver sets are available, supporting either Windows 95/98/Millennium or Windows2000/XP. Please check our support website regularly for driver updates.

To find programming samples incorporating both, the card driver and the DLL functions, we suggest you check our support website ***http://www.pcdmx512.com*** regularly. Look for software, and check the drivers and skeleton pages. Demos will be issued for various, but not all programming languages. As all sources are available in ASCII format, you will be able to compare the steps required, even if you have to do some re-coding,

The SOUNDLIGHT support team

SOUNDLIGHT
The DMX Company

1 - Introduction

The 2512A PCMCIA DMX Card is a Dual Link DMX-512 interface supplied by SOUNDLIGHT. The interface may be used on any Windows Compatible notebook or laptop computer to develop lighting control solutions.

The PCI card comes in two versions, either as dual link (1024 DMX channels) or as quad link (2048 DMX channels) model. As the electronic circuitry used for the PCI card widely resembles that of the PCMCIA interface card, the DLL function calls can be used for both, the PCI and the PCMCIA interface.

This manual describes the functions provided by the Library developed for the PCMCIA DMX Card, called DMXCRD32 and distributed as a Win32 compatible Dynamic Link Library (DLL) file called DMXCRD32.DLL.

As with any Win32 DLL, the library may be used by all high-level language environments including the modern visual type programming environments (e.g. Visual C++, Visual Basic and Delphi), operating under Windows95, Windows98, Windows Millennium, Windows2000 and Windows XP.

The operation of the card is enabled by a Plug and Play driver. This document does not provide information on driver installation, and assumes the card is properly installed in the target environment before the program using DMXCRD32.DLL is started. For specific information about the PCI card installation, the PCMCIA DMX card installation and different environment support, please check our support website at ***http://www.pcdmx512.com***.

IMPORTANT
Using the DLL functions requires a properly defined set of global variables. These variables will be read and modified as required when calling the DLL, even if SUB calls are used throughout. Please note, that all variables are accessed by reference rather than by value. That is, a variable must have been defined before a call using that variable can be made.

The DMXCard DLL (DMXCRD32.DLL) has a revision identifier in the 0.xy form, with x and y being a minor and subminor revision code. This manual applies to DLL version 0.12 and higher. For specific information about different revision levels and related functions, please contact SOUNDLIGHT support.

2 - DLL function definitions

Following the Win32 definition, DMXCRD32.DLL uses the stdcall calling convention for all parameters. The list of functions supported by DMXCRD32.DLL is listed in the following table:

LIBVER(version)

DMXINIT (address,dmxsize,dmxmst)
DMXSENSE(address,dmxstatus,dmxtype)
DMXIDENT(address,dmxstatus,dmxhard,dmxrev,dmxsubrev,serialno)
DMXREV(address,revsiz,revbuf)
DMXDEMO(address)

DMXDRIVE(address,dmxstart,dmxstatus)
DMXDRIVE2(address,dmxstart,dmxstatus)
DMXRELAX(address)
DMXSTANDARD(address)

DMXSEND(address,dmxchan)
DMXSEND2(address,dmxchan)

DMXCAPTURE(address,dmxstart,dmxpkt,dmxchan)
DMXCAPTURE2(address,dmxstart,dmxpkt,dmxchan)

In the following chapters, each function is defined in more detail. As seen, all DLL functions that involve operation of the DMX line are address-parametric, i.e. they accept the Base Address (modulo 256, see DMXInit function description) of the virtual peripheral memory window (4kB size) used by DMXCard driver to let user code access card resources in Windows Protected Memory environment.

The detail on how to instantiate the driver in different programming languages and how to obtain the Base Address from the driver are reported in the DMXCard Driver Programming Guide, separately provided by SOUNDLIGHT. The information how to install, setup and check the driver can be obtained from our support website at ***http://www.pcdmx512.com.***

2.1 - LIBVER

The LIBVER function is simply used to report the revision level of the DMXCRD32.DLL library to the high-level calling program. This is used to support global configuration report in user applications and to provide traceability for field maintenance.

The only returned parameter is version, that is a ASCII-packed word of 4 bytes in the x.yz format. For example, DLL revision 0.12 returns the four Ascii characters "0","."."1","2" or 0x30,0x2e,0x31,0x32 sequentially packed in a single 32-bit unsigned number.

```
;*************************************************************************
;LIBVER(version)
;;function:      return DMX library version
;;C++ prototype: extern __stdcall LIBVER(int *);
;;VB4 prototype: Declare Sub LIBVER(version);
;;input parameters:
;;  none
;;output parameters: none
;;  version         32-bit pointer to version structure x.yz ASCII
;;*************************************************************************
```

2.2 - DMXINIT

The DMXINIT function is called once at program initialization in order to setup the DLL variables associated with the specified PCMCIA DMX Card interface.

The ***address*** parameter specifies the Base Address (modulo 256, that is shifted 8-bits to remove least significant byte that is always zero for the PCMCIA Card). This Base Address is obtained by the User Program during the call to the DMXCard driver that configures the card (for details, refer to driver documentation and sample code available from our support website).

The ***dmxsize*** parameter is used to setup the size of the buffer used by the DLL to handle DMX channel values, and would normally be set at 512.

The ***dmxmst*** parameter is used to setup the driving state (master vs. slave) of the DMX interface. Master mode is selected with dmxmst=1 and Slave Mode with dmxmst=0.
The default value for DMXInit is dmxmst=1 (Master Mode) to allow automatic Demo Mode setting in User Application when DMXCard is not installed (see the section on card startup sequence).

Here is the function template:

```
;*******************************************************************************
;DMXINIT (address,dmxsize,dmxmst)
;;function:      Initializes DMXLIB module variables: to be issued before
;;               issuing any other DMX command
;;C++ prototype: extern __stdcall DMXINIT(int *, int *, int *);
;;VB4 prototype: Declare Sub DMXINIT(add,dmxsize,dmxmst);
;;input parameters:
;;  address      32-bit pointer to Base Address modulo 256
;;  dmxsize      32-bit pointer to DMX string size (1 to 512 channels)
;;  dmxmst       32-bit pointer to DMX Master/Slave mode (1=Master, 0=Slave)
;;output parameters: none
;;*******************************************************************************
```

2.3 - DMXSENSE

The DMXSENSE function is used to check the presence of the DMX interface and it reports some information on the interface type.

The ***address*** parameter is used to specify the Base Address of the DMX (modulo 256, see DMXInit function description).

The ***dmxstatus*** return parameter is used to report cumulative errors and should always be returned at 0. If different than zero, the user program should be terminated and the value should be reported for problem addressability.

The ***dmxtype*** return parameter is used to report some information on the DMX Interface.
Using the PCMCIA DMX Interface Card this parameter is always returned at 1.

Note that to simplify the use of the high-level program with no DMX interface connected (e.g. for off-line show preparation), the DLL enters an automatic demo mode when the first DMXSENSE call executed after DMXINIT does not detect the DMX Interface. In this case all the following function calls are skipped to let the program run unchanged with no hardware connected and no timeout overhead. The type of interface connected, including the no-hardware demo case is reported in detail by the DMXIDENT function.

Here is the function template:

```
;*************************************************************************
;DMXSENSE(add,dmxstatus,dmxtype)
;;function:      Issues Sense command to Dmx Interface to test if it is present
;               and what type of hardware is present.
;               First DMXSense call to be done after DMXInit before any
;               other function call
;;C++ prototype: extern __stdcall DMXSENSE(int*, int *, int *);
;;VB4 prototype: Declare Sub DMXSENSE(address,dmxstatus,dmxtype);
;;input parameters:
;;       address          32-bit pointer to Base Address modulo 256
;;output parameters:
;;       dmxstatus        32-bit pointer to DMXStatus
;                         DMXStatus (0000h=DmxCard present, no errors logged
;                         xxyy=errors where:
;                         xx<>0 = Sense failed  yy<>0 = logged errors
;                         Program Should be terminated on any Error
;;       dmxtype          32-bit pointer to DMXType (valid on DMXStatus=0)
;                         for DMXCard DmxType is returned =1 (Profess. interface)
;;*************************************************************************
```

2.4 - DMXIDENT

The DMXIDENT function is used to report a cumulative set of information regarding the DMX interface hardware. This function is used to handle revision updates as these may occur during production run of the interface.

The usual address parameter specifies the interface Base Address Modulo 256 (see definition in DMXInit function).

The ***dmxstatus*** is the return variable indicating inquiry success (0 returned) or the occurrence of any problem during hardware inquiry. The user program should be terminated in case dmxstatus is returned non-zero.

The function, when successfully completed (dmxstatus=0) reports the information about the interface in the dmxhard, dmxrev, dmxsubrev, dummy variables in this way:

VARIABLE	MEANING	RETURNED VALUES
dmxhard	hardware type	0=none 4=PCMCIA DMX Card
dmxrev	firmware revision	1 ASCII character
dmxsubrev	firmware sub-revision	2 Ascii characters
dummy	for future use	Reported Null for PCMCIA DMX Card

This is the function prototype:

```
;*************************************************************************
;DMXIDENT(address,dmxstatus,dmxhard,dmxrev,dmxsubrev,dummy)
;;function:      Used to identify the DMX interface hardware installed
;;C++ prototype: extern __stdcall DMXIDENT(int*, int *, int , int*, int*, int*);
;VB4 prototype: Declare Sub
DMXIDENT(address,dmxstatus,dmxhard,dmxrev,dmxsubrev,serialno);
;;input parameters:
;       address             32-bit pointer to Base Address modulo 256
;;output parameters:
;       dmxstatus           (0000h=DMXCard Interface present, no errors logged
;                           xxyy=errors where:
;                                xx<>0 = Sense failed
;                                yy<>0 = logged errors)
;;      dmxhard             32-bit pointer to DMX Interface Hardware type encoding
;                           0000h=no hardware (demo mode)
;                           0004h=DMXCard
;                           (values 1 to 3 reserved for future use)
;;      dmxrev              32-bit to Hardware Revision code (1 Ascii)
;;      dmxsubrev           32-bit pointer to Hardware SubRevision code (2 Ascii)
;;      dummy               32-bit serial number, null for PCMCIA Card
;;*************************************************************************
```

2.5 - DMXREV

The DMXREV function is used to report the full revision template of the DMX interface firmware revision. This is normally not required since DMXIDENT is used to report firmware revision/subrevision. As with DMXIDENT, this function let the DMX interface be used as a hardware key. DMXREV may be only called when in Slave Mode, i.e. it must be called before the DMXDRIVE function. Calling DMXREV from Master mode clears the active driving state and requires a DMXDRIVE command to be re-issued before making DMXSEND calls.

The ***address*** parameter specifies the interface Base Address (modulo 256, see DMXInit function).

The ***revsiz*** return parameter specifies the number of revision bytes returned by the interface (revision size for DMXCard is 15 bytes stored in a string of 15 32-bit word, maximum size for all present PCI / PCMCIA DMX hardware is 30 32-bit word).

The ***revbuf*** parameter is a pointer to a revision buffer that should be an array of 32 words of 32-bit for future expandability.

The typical returned revision string for DMXCard is a string of 15 characters stored in an array of 15 32-bit words, in the following format:

`DMXOEM1øøøøVr.r` (ø representing a space or blank character)

"r.r" is a revision/subrevision field (e.g. V3.0), that indicates the Hardware Revision of the PCMCIA Card.

This is the function prototype:

```
;*****************************************************************************
;DMXREV(address,revsiz,revbuf)
;;function:       Captures a revision-level packet from DMXCard
;                 Checks handshake timeouts and returns DMXPkt=0 on any error
;;C++ prototype: extern __stdcall DMXREV(int*, int *);
;VB4 prototype: Declare Sub DMXREV(add,revsiz,revbuf);
;;;input parameters: none
;;  address    32-bit pointer to Base Address modulo 256
;;output parameters:
;;  revsiz     32-bit pointer to size of received rev packet
;   revbuf     32-bit pointer to received rev packet (string of  32-bit words with
;              ASCII charcaters)
;;*****************************************************************************
```

2.6 - DMXDEMO

The DMXDEMO function is used to force the demo mode of the DMXCRD32 library in order to prevent any memory access during the configuration phase.

This is useful when the user program is run in operating environments such as Windows2000/XP that even do not allow real memory accesses, and the program is designed to call the DLL functions also when the card is not present and the driver can not report a valid Base Address.

DMXDEMO should be called before any other function in the Card Configuration Phase in order to avoid any active action in the following function calls. The functions are just void and do not perform any action on the hardware, always returning success codes.

The address parameter is used as always to specify the interface Base Address.
The calling syntax is the following:

```
;*************************************************************************
;DMXDEMO(address)
;;function:        Set DMXCRD32 library in demo mode to prevent any peripheral
;;                 memory access;
;;C++ prototype: extern "C" DMXDEMO(int *);
;;VB4 prototype: Declare Sub DMXDEMO(add);
;;input parameters:
;;       address   32-bit pointer to Base Address
;;output parameters:
;;       none
;;*************************************************************************
```

2.7 - DMXDRIVE / DMXDRIVE2

The DMXDRIVE and DMXDRIVE2 functions are used to enable the DMX interfaces of DMX Link 1 and DMX Link 2 to drive the DMX line.

The address parameter is used to specify the Base Address of DMXCard (modulo 256, see DMXInit function).

The ***dmxstart*** parameter is used to set the Startcode Byte as defined by DMX-512/1990 and is normally set at 0 for standard DMX devices like dimmers, scrollers and scanners.

The ***dmxstatus*** is a return value that is used to verify normal execution of the function. When DMXDRIVE execution is successful, the return value of dmxstatus must be 0. Any value different than 0 indicate a fatal error and should terminate user program.

Prior to DMXDRIVE call, you may want to call the DMXRELAX function to select relaxed DMX timings (see the description for DMXRELAX and DMXSTANDARD functions).

The standard configuration for the PCI / PCMCIA Card, after the DMXDRIVE function has been called, is with Link1 set as output and Link2 set as input. In this configuration, a DMXCAPTURE may be called at any time to input 512 DMX channels from Link2.

If a 1024 DMX channel output is required, then DMXDRIVE2 should be called after DMXDRIVE to set also Link2 in output mode, achieving a Out/Out configuration for Link1 and Link2. DMXDRIVE2 has the same calling syntax as DMXDRIVE, and the Base Address to be used is the same used by DMXDRIVE (see section 3 for details on card Input/Output configurations).

```
;*************************************************************************
;DMXDRIVE(address,dmxstart,dmxstatus)
;DMXDRIVE2(address,dmxstart,dmxstatus)
;;function:        Issue Drive command to DmxCard to set DMX Output active on
;;                 Link1 (DMXDRIVE)  or Link2 (DMXDRIVE2)
;;C++ prototype: extern __stdcall DMXDRIVE(int*, int *, int *);
;;C++ prototype: extern __stdcall DMXDRIVE2(int*, int *, int *);
;;VB4 prototype: Declare Sub DMXDRIVE(add,dmxstart,dmxstatus);
;;VB4 prototype: Declare Sub DMXDRIVE2(add,dmxstart,dmxstatus);
;;input parameters:
;;       address            32-bit pointer to Base Address modulo 256
;;       dmxstart           32-bit pointer to dmxstart (DMX Start Code)
;;output parameters:
;;       dmxstatus          32-bit pointer to dmxstatus
;;                          dmxstatus: (0000h=successful,<>0000h=fatal error)
;;*************************************************************************
```

2.8 - DMXRELAX / DMXSTANDARD

The DMXRELAX and DMXSTANDARD functions are used to modify the DMX timings in order to compensate for dynamic limitation of DMX luminaires not fully compliant with the current version of the DMX-512 standard, as defined in USITT DMX512/1990 or in DIN56930-2.

The DMXRELAX function is used to set relaxed DMX timings before a DMXDRIVE function is performed, while DMXSTANDARD function is used to restore full-speed DMX timings compatible with DMX-512/1990 (also in this case the function is only active after the next DMXDRIVE command is executed).

The ***address*** parameter is used to specify the port where the DMXCard interface is connected (modulo 256, see DMXInit function).

This function is typically useful when working with critical or marginal DMX-512 equipment that cannot tolerate the maximum theoretical speed and may therefore need relaxed timing.

Here is how typical timings are altered:

Parameter	Unit	Standard Timing	Relaxed Timing
Break	us	100	200
MAB (min)	us	10	10
Bytes/packet		512	512
BREAK-to-BREAK (min)	us	22710	28660
Updates/sec (max)		44	35

```
;*************************************************************************
;DMXRELAX(address)
;DMXSTANDARD(address)
;;function:       Set Relaxed or standardTimings on DMX line. Changes are
;                 effective after  next  DMXDRIVE function call.
;;C++ prototype: extern "C" DMXRELAX(int *);
;C++ prototype: extern "C" DMXSTANDARD(int *);
;VB4 prototype: Declare Sub DMXRELAX (add);
;VB4 prototype: Declare Sub DMXSTANDARD (add);
;;input parameters:
;;       address       32-bit pointer to Base Address modulo 256
;;output parameters:
;;       none
;;*************************************************************************
```

2.9 - DMXSEND / DMXSEND2

The DMXSEND and DMXSEND2 functions are simply used to update the channel values to be transmitted on the DMX line. Like all DMX interfaces supplied by SOUNDLIGHT, both, the PCI interface card and the the PCMCIA card are intelligent interfaces, performing automatic refresh of the DMX line using the last transferred values.

DMXSEND is used to set ouput values for Link1 (after DMXDRIVE has been issued), while DMXSEND2 is used to set output values for Link2 (after having issued a DMXDRIVE2 command).

The ***address*** parameter specifies the interface Base Address (modulo 256, see DMXInit function).

The ***dmxchan*** parameter is a pointer to the channel buffer, that must be an array of 32-bit words, each containing the DMX channel value in the least significant byte.

```
;*************************************************************************
;DMXSEND(address,dmxchan)
;DMXSEND2(address,dmxchan)
;;function:        send DMX channel values for pre-configured size in the
;;                 output buffer of DMXCard Link1 (DMXSEND) or
;;                 Link2 (DMXSEND2)
;;C++ prototype: extern __stdcall DMXSEND(int* , int*);
;;C++ prototype: extern __stdcall DMXSEND2(int *,int*);
;;VB4 prototype: Declare Sub DMXSEND(address,dmxchan());
;;VB4 prototype: Declare Sub DMXSEND2(address,dmxchan2());
;;input parameters:
;;       address            32-bit pointer to Base Address modulo 256
;;       dmxchan, dmxchan2  32-bit pointer to DMX channels array (32-bit integer,
;;                           max size 512)
;;output parameters: none
;;*************************************************************************
```

2.10 - DMXCAPTURE / DMXCAPTURE2

The DMXCAPTURE and DMXCAPTURE2 functions are used to enter Slave Mode (in case the DMX Link was previously driving DMX line) and capture the first packet detected on the DMX line of DMX Link 2 (DMXCAPTURE) or Link 1 (DMXCAPTURE2). In this way the state of the DMX line may be monitored, captured and stored in the user program.

The capturing performance allows some level of dynamic input capability (e.g. to capture real-time DMX-512 pattern for later playback with DMX direct-to-disk programs).

As usually, the address parameter specifies the DMXCard Base Address (modulo 256, see definition in DMXInit function).

The dmxstart return parameter reports the value of the DMX Startcode of the captured packet (normally 0 for dimmer applications). The dmxpkt value reports the acquired number of channels from 0 to 512, where a value of 0 indicates a capture timeout (i.e. no DMX signal found).

The dmxchan parameter is a pointer to the channel buffer, that must be an array of 32-bit words, where DMX values are stored in the least significant byte.

```
;*****************************************************************************
;DMXCAPTURE(address,dmxstart,dmxpkt,dmxchan)
;DMXCAPTURE2(address,dmxstart,dmxpkt,dmxchan)
;;function:      Captures a packet from DMXCard interface Link2 (DMXCAPTURE) or
;;               Link1 (DMXCAPTURE2), after issuing a reverse line command if
;;               line was previuosly used for driving.
;;               Captures DMX-512 startcode, channel count and values.
;;               Packet size is adaptive and returned to calling program
;;               (returns DMXPkt=0 if no DMX signal is detected or on any error).
;;C++ prototype: extern __stdcall DMXCAPTURE(int *, int *, int *, int *);
;;C++ prototype: extern __stdcall DMXCAPTURE2(int *, int *, int *, int *);
;;VB4 prototype: Declare Sub DMXCAPTURE(address,dmxstart,dmxpkt,dmxchan);
;;VB4 prototype: Declare Sub DMXCAPTURE2(address,dmxstart,dmxpkt,dmxchan);
;;
;;input parameters:
;;       address       32-bit pointer to Base Address modulo 256
;;       dmxchan       32-bit pointer to DMX channels array (max 512 32-bit words)
;;output parameters:
;;  dmxstart      32-bit pointer to startcode of received DMX packet
;;  dmxpkt        32-bit pointer to received DMX packet lenght
;;  <implicit>    DMX channels stored into dmxchan input array
;;*****************************************************************************
```

3. CARD IDENTIFICATION, INITIALIZATION AND FUNCTIONAL USE

This section specifies the sequence to be used for DMXCard identification and initialization.

The DMXCard is supposed to have been previously configured in the system by the Windows Device Manager, using the Plug and Play driver provided by SOUNDLIGHT with the card.

Note that in order to avoid any peripheral memory access (that could cause program termination under Windows2000/XP) in case the program is executed without hardware (for example in case the user wants to preprogram a show without DMXCard Hardware installed), the DMXDEMO(add) function may be executed first, resulting in dynamic skip of all the following functions, letting the user program access the function with no effect.

The recommended sequence of DMXCRD32 DLL functions to be used in user programs is reported hereafter.

```
-------------- Example of  DMXCRD32.DLL functions use  ------------------------
-------------- ©SLH/DML 2000-2002,  All Rights Reserved ----------------------

-------------- DMXCard initialization ----------------------------------------

Receiving Base Address from PnP driver instantiation
address=Base Address MOD 256
dmxsize=512
dmxmst=1
Call DMXINIT (add,dmxsize,dmxmst)

Call DMXSENSE(add,dmxstatus,dmxtype)
        Check dmxstatus=0 and dmxtype=1, stop on error

Call DMXIDENT(add,dmxstatus,dmxhard,dmxrev,dmxsubrev,dummy)
        Check dmxstatus=0 and dmxhard=4, flag card not recognized otherwise
        Get dmxrev,dmxsubrev for user logging, dummy=null for PCMCIA Card

-------------- optional information logging (recommended) ----------------------

Call DMXREV(add,revsiz,revbuf)
         for Hardware Revision logging (optional, recommended)
Call LIBVER(version)
         for Library Version logging (optional, recommended)

-------------- user functional code start, enable output Links -----------------

Call DMXDRIVE(add,dmxstart,dmxstatus)
         to enable Link1 output with dmxstart StartCode
         Check dmxstatus=0, stop on error
Call DMXDRIVE2(add,dmxstart,dmxstatus)
         to enable Link2 output with dmxstart StartCode
         Check dmxstatus=0, stop on error

-------------- user functional code for DMX output -----------------------------

Set dmxchan array with channel values to be sent on Link1
Call DMXSEND(add,dmxchan) to update Link1 channels output

         Same as above with dmxchan2 and DMXSEND2 to send on Link2
```

```
-------------- user functional code for DMX input -----------------------------

Call DMXCAPTURE(add,dmxstart,dmxpkt,dmxchan) to capture Link2 channels
          Check dmxpkt not zero to check channels effectively captured
          Use dmxstart information if required
               (just for special usages, normally =0)
          Process DMX channel input from  dmxchan array

          Same as above with DMXCAPTURE2 and dmxchan2 to capture from Link1
```

This example shows the case of single or dual link output and single or double link input. Note that all the following combinations are possible:

MODE	Functions to be called
Link1 Out / Link 2 In (512 channels, Full Duplex Mode)	Card Configuration sequence . . DMXDRIVE . . DMXSEND (each Link1 update) . DMXCAPTURE (each Link2 capture)
Link1 Out / Link 2 Out (1024 output channels, Half-Duplex Mode)	Card Configuration sequence . . DMXDRIVE DMXDRIVE2 . . DMXSEND (each Link1 update) DMXSEND2 (each Link2 update)
Link1 In / Link 2 In (1024 input channels, Half-Duplex Mode)	Card Configuration sequence . . DMXCAPTURE (each Link2 capture) DMXCAPTURE2 (each Link1 capture)

Note that the default line numbering is defined for the most common case of 512 channels Full-Duplex mode (where Link 1 is in input and Link 2 is in output, allowing real-time transfer/merge from Link2 to Link1).

When operating in 1024-channels Half-Duplex mode, DMXCAPTURE function captures from Link2 and DMXCAPTURE2 captures from Link1 (note in this case the non intuitive link numbering arrangement). In Half-Duplex mode, the line switch and termination setting is automatically performed by the PCMCIACard hardware, in a way that the 2 DMX In and OUT links may be kept constantly connected.

While performing DMXDRIVE on one Link, the input for that link is disconnected from DMXCard, the line is source terminated and the line is driven by the PCMCIA Card.

While performing DMXCAPTURE on a link instead the line is reversed, DMXCard is set in input mode, the termination is removed and the input link is propagated on the output link, to perform passive, pass-through capture.

APPENDIX A - Using DMXCRD32.DLL in different languages

This Appendix reports two simple examples of use of the DMXCRD32 DLL functions from a user program in Visual C++ and Visual Basic. Other examples will be published on our support website, which can be visited at ***http://www.pcdmx512.com***.

The examples are only reported for programming guidance and are not representative of complete handling of the interface by the user program.

A.1 - Visual C++

```
BOOL dmxcard_ok = FALSE;
HINSTANCE hLibrary;
FARPROC dcFonction;

int add, dmxsize, dmxmst, dmxstart, dmxstatus;

BOOL Init_dmxcard()
{
      add     = 0xe18;      // address obtained using driver base address
                                                          modulo 256 //
      dmxsize = 512;
      dmxmst  = 1;          // set to default initial state //
      dmxstart = 0;         // startcode for DMX-512/1990 dimmers //

      hLibrary = LoadLibrary("DMXCRD32");
      if (hLibrary == NULL)
            return FALSE;

      dcFonction = GetProcAddress(hLibrary, "DMXINIT");
      (*dcFonction) (&add,&dmxsize,&dmxmst);   // setup DLL variables //
      //DMXINIT (&add,&dmxsize,&dmxmst);       // setup DLL variables //

      dcFonction = GetProcAddress(hLibrary, "DMXSENSE");
      (*dcFonction) (&add,&dmxstatus,&dmxtype); // setup DLL variables //
      //DMXSENSE (&add,&dmxstatus,&dmxtype);   // setup DLL variables //

      if (dmxstatus != 0)
         return FALSE;

      dcFonction = GetProcAddress(hLibrary, "DMXIDENT");
      (*dcFonction) (&add,&dmxstatus,&dmxhard,&dmxrev,&dmxsubrev,&dummy);
      //DMXIDENT (&add,&dmxstatus,&dmxhard,&dmxrev,&dmxsubrev,&dummy) //

      if (dmxstatus != 0)
         return FALSE;

      dcFonction = GetProcAddress(hLibrary, "DMXDRIVE");
      (*dcFonction) (&add,&dmxstart,&dmxstatus); // setup DLL variables //
      //DMXDRIVE(&add,&dmxstart,&dmxstatus);   // enable DMX line driver //

      if (dmxstatus != 0)
         return FALSE;
      return TRUE;
}

void dmxcard(unsigned char *bloc, int channels)
{
      int dmxchan[512],i;

      for(i=0;i<512;i++)
        if (i<channels)
              dmxchan[i] = (int) bloc[i];
        else  dmxchan[i] = (int) 0;
      dcFonction = GetProcAddress(hLibrary, "DMXSEND");
      (*dcFonction) (&add,&dmxchan[0]);             // setup DLL variables  //
      //DMXSEND(&add,&dmxchan[0]);        // send new channels on DMX line //
                           // last packet sent is automatically refreshed  //
                           // on DMX line by PCMCIA Card hardware          //
}
```

A.2 Visual BASIC

```
`******************************
`Definitions in main BAS module
'******************************

`*********************
`variables declaration
`*********************

Public add As Long
Public dmxsize  As Long
Public dmxmst As Long
Public dmxstatus As Long
Public dmxhard As Long
Public dmxrv As Long
Public dmxsbrv As Long
Public dmxstart As Long
Public dmxpkt As Long
Public dmxtype As Long
Public revsiz As Long
Public mast As Long
Public version As Long
Public serialno As Long

`******************
`arrays declaration
`******************

Public revbuf(32) As Long
Public dmxchan(512) As Long
Public revstr(32) As String
Public verstr(4) As String

`***************************
`DLL subroutines declaration
`***************************

Declare Sub LIBVER Lib "DMXCRD32.DLL" (ByRef version As Long)
Declare Sub DMXINIT Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxsize As Long,
     ByRef dmxmast As Long)
Declare Sub DMXSENSE Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxstatus As
     Long, ByRef dmxtype As Long)
Declare Sub DMXIDENT Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxstatus As
     Long, ByRef dmxhard As Long, ByRef dmxrv As Long, ByRef dmxsbrv As Long,
     ByRef dummy As Long)
Declare Sub DMXREV Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef revsiz As Long,
     ByRef revbuf As Long)
Declare Sub DMXDEMO Lib "DMXCRD32.DLL" (ByRef add As Long)
Declare Sub DMXDRIVE Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxstart As
     Long, ByRef dmxstatus As Long)
Declare Sub DMXDRIVE2 Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxstart As
     Long, ByRef dmxstatus As Long)
Declare Sub DMXRELAX Lib "DMXCRD32.DLL" (ByRef add As Long)
Declare Sub DMXSTANDARD "DMXCRD32.DLL" (ByRef add As Long)
Declare Sub DMXSEND Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxchan As Long)
Declare Sub DMXSEND2 Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxchan As Long)
Declare Sub DMXCAPTURE Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxstart As
     Long, ByRef dmxpkt As Long, ByRef dmxchan As Long)
Declare Sub DMXCAPTURE2 Lib "DMXCRD32.DLL" (ByRef add As Long, ByRef dmxstart As
     Long, ByRef dmxpkt As Long, ByRef dmxchan As Long)
```

```
`*****************************
`PROGRAM in form or module code
`*****************************

`***************
`variables setup
`***************

dmxsize = 512
dmxmst = 1
dmxstart = 0
dmxstatus = 0
dmxhard = 0
dmxrv = 0
dmxsbrv = 0
dmxsize = 512
dmxpkt = 0
dmxtype = 0
revsiz = 0
mast = 100
version = 0
serialno = 0

`***************************
`DMX Interface initialization
`***************************

`assuming base address returned from driver=&H000E1800
add = &HE18
`initialize DMX interface
Call DMXINIT(add, dmxsize, dmxmst)
Call DMXSENSE(add, dmxstatus, dmxtype)
If (dmxstatus <> 0) Then
      `card error, exit user program
End If

Call DMXIDENT(add, dmxstatus, dmxhard, dmxrv, dmxsbrv, dummy)
If (dmxstatus = 0) Then
      `insert here card description management to validate card usage
      Else
      `card error, exit user program
End If

`***********************************
`simple DMX send, all channels to zero
`***********************************

`case Standard DMX timings (default)
`Issue Drive command to enable DMX output
Call DMXDRIVE(add, dmxstart, dmxstatus)
Debug.Print "status=" + Str(dmxstatus)

`case Relaxed DMX timings (alternative)
Call DMXRELAX(add)
`Issue Drive command to enable DMX output
Call DMXDRIVE(add, dmxstart, dmxstatus)
Debug.Print "status=" + Str(dmxstatus)

`setup channels
For J% = 1 To dmxsize
dmxchan(J%) = 0
Next J%
```

```
`send DMX channels values to DMX

Call DMXSEND(add, dmxchan(1))

..............

`******************
`simple DMX receive
`******************

`setup array to pre-receive data to check update from DMX
For J% = 1 To dmxsize
dmxchan(J%) = &H55
Next J%

Call DMXCAPTURE(add, dmxstart, dmxpkt, dmxchan(1))

Debug.Print "Received " + Str(dmxpkt) + " channels from StartCode " +
Str(dmxstart)
Debug.Print "Capture Complete"
`captured channels are in dmxchan(i) array

..............

`*************************
`Get DMX interface revision
`*************************

Call DMXREV(add, revsiz, revbuf(1))

For i% = 1 To 32
revstr(i%) = ""
Next i%
For i% = 1 To revsiz
revstr(i%) = Chr$(revbuf(i%) Mod 256)
Next i%

..............

`***************
`get DLL Version
`***************

Call LIBVER(version)

'extract version string from zy.x byte string
For i% = 1 To 4
verstr(i%) = Chr$(version Mod 256)
version = version \ 256
Next i%

Debug.Print "Library Version = " + verstr(1) + verstr(2) + verstr(3) + verstr(4)
```

Remarks