DMX-Dongle II & Art-Net Driver SDK for Windows 95, 98, ME, 2000, NT, XP & DOS

Artistic Licence (UK) Ltd

Manual Revision V4.6. (DLL Revision 19)

C O N T E N T S

I N T R O D U C T I O N

Quick Start

Welcome to the DMX-Dongle II & Art-Net Software Development Kit.
This guide contains all the information required to develop lighting control applications using either the DMX-Dongle II or Art-Net.

The SDK supports Windows 95, 98, ME, 2000, NT5 & XP using a unified driver. The unified driver concept allows the application programmer to develop code that is independent of the actual control hardware.

The SDK also supports a DOS level interface for the DMX-Dongle II that includes source code.

DMX-DONGLE II

The DMX-Dongle II is a parallel port interface for DMX512. The DMX-Dongle II provides the following features:

- DMX512 transmission of 512 channel data.
- Programmable transmit Break, Mark after Break and Header Code
- DMX512 receive with programmable base address
- Receive 512 channels @ 8 bit resolution and full bandwidth
- Simultaneous transmit and receive
- Programmable Merge and Loop Through
- Programmable receive Header Code
- Analysis of receive errors
- Receive Break and Mark after Break timing analysis
- Multiple oscilloscope trigger outputs

ART-NET

Art-Net is an Ethernet protocol for lighting control. The protocol is public domain and royalty free. The full specification is available from www.ArtisticLicence.com\art-net.pdf

Art-Net is a 10BaseT Ethernet protocol that uses a sub-set of the TCP/IP protocol.

Art-Net is supported by numerous manufacturers including AC Lighting Ltd, AC Lighting Inc, ADB, MA Lighting, Doug Fleenor Design, ELC Lighting, Electronics Diversified, Enttec, Goddard Design Co, I-Light Group, IES, Medalion, Media Motion, SandNet, Touchlight Systems Ltd, Zero 88 and Artistic Licence.

Artistic Licence products supporting Art-Net include:

- Coloour-Tramp
- Grand-Master Flash!
- DMX-Workshop
- Art-Net View
- Net-View
- Down-Lynx
- Down-Lynx A
- Up-Lynx
- Up-Lynx A
- Net-Lynx O/P
- Net-Lynx I/P
- Ether-Lynx

WINDOWS DEVELOPMENT

Overview

The Windows development environment supports all six key platforms: 95, 98, ME, 2000, NT & XP.
The driver interface is independent of the operating system. The application simply needs to load the DLL, all other details are handled by the driver.
The DLL handles loading of the relevant .sys file for Windows 2000, NT or XP and the .vxd file for Windows 95, 98 and ME.
The following section provides an overview of application to driver access.

FILES

The files required by the driver are as follows:

Software Interface:

• DongleArtNet.h	Contains all function prototypes. Include this file in all application files that need access to driver functions
• Art-Net.h	Contains all Art-Net class definitions. This file is included by DongleArtNet.h
• DongleArtNet.lib	Include file for DLL
• DongleArtNet.dll	The driver

Windows 95, 98, ME low level drivers

• Dong32.vxd	Kernel mode driver

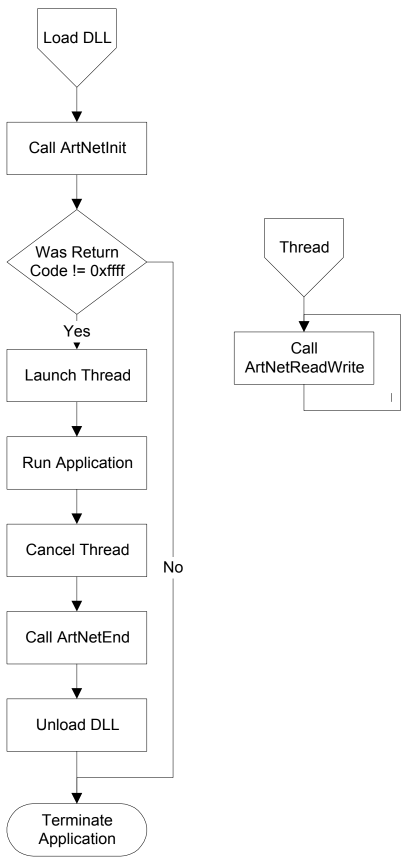
Windows 2000, NT & XP low level drivers

• DongleArtNet1.sys	Kernel mode driver for lpt1
• DongleArtNet2.sys	Kernel mode driver for lpt2
• DongleArtNet3.sys	Kernel mode driver for lpt3
• Windrvr.sys	Kernel interface driver

In order to simplify installation, it is acceptable to install all files on all platforms. A sample 'InstallShield' project is included with the SDK.

CODE INTERFACE

The following flow chart shows the most basic application interface to the driver:



The two functions ArtNetInit and ArtNetReadWrite are the core driver functions. ArtNetInit handles loading the lower level drivers and then reports back on the hardware status.

ArtNetReadWrite is the low level function that moves data from the local buffers to the hardware (DMX-Dongle II or Art-Net). This function is usually called from within a thread. It can be called from within the application. When the DMX-Dongle II is installed, the function requires a significant amount of processing time, so it is worth implementing code that ensures the call is only made when necessary.

MEMORY STRUCTURE

The data structures used by the driver are declared in the DongleArtNet.h file. The application has access to both structures:

```
PipeEntry PipeLibrary[MaxPipes];
NodeEntry NodeLibrary[MaxNodes];
```

Pipe Library

The PipeLibrary is an array of PipeEntry, indexed by the variable Pipe.
Each PipeEntry contains all the data relating to a single transmit and a single receive Universe. The API includes functions to access all members, although direct access is allowed.
The definition is as follows:

```
typedef struct {

bool        TxEnable;
                  // true if this transmit is enabled

T_ArtDmx    TxDmx;
                  // Contains the transmit data

T_ArtDmx    RxDmx;
                  // Contains the receive data

WORD        RxRequestUniverse;
                  // User sets the Universe number of
                  // the data that the application
                  // would like to receive. Only low
                  // byte is used: High Nib = Subnet.
                  // Application sets to 0xffff,
                  // Driver will then fill table with
                  // first received packets.

BYTE        RxIpAddress[4];

                  // Set to the IP address from which
                  // data was received

WORD        RxPacketCount;
                  // ArtDmx packet received counter

WORD        TxPacketCount;
                  // ArtDmx packet sent counter

                  // Private entries omitted for
                  // clarity

                  }PipeEntry;
```

Node Library

The NodeLibrary is an array of NodeEntry, indexed by the variable Node.
Each NodeEntry contains data relating to an Art-Net or DMX-Dongle II device. The key difference between NodeLibrary and PipeLibrary is that a Node may handle multiple universes. Therefore a single entry in NodeLibrary may correspond to multiple entries in PipeLibrary. API routines exist to allow cross reference between the two tables.
The definition is as follows:

```
typedef struct {

BYTE  IpAddress[4];

                  // Read Only Driver fills this field
                  // with the IP address of the node
                  // from which the reply was received

T_ArtPollReply    ArtPollReply;

                  // Filled by the driver when
                  // an ArtPollReply is received.

                  // Private entries omitted for
                  // clarity

                  }NodeEntry;
```

Transmit Strategy

As discussed in the previous section, ArtNetReadWrite is the primary function for data transfer.

The way that this function is called is possibly the most significant issue in using the driver.

There are two key considerations:

1. The execution time of ArtNetReadWrite
2. The network loading

When the DMX-Dongle is detected, the zeroth PipeEntry is allocated for the DMX-Dongle. A call to: **`ArtNetReadWrite(0)`** will refresh both transmit and receive DMX512 data via the DMX-Dongle. The execution time of this function is approximately 25mS when all channels are included. For this reason it is advisable to implement an intelligent application routine that only calls the function when necessary.

The execution time can be improved by the use of **`DongleSetTransferCount().`** This function is used to set the number of channels that are processed. If the application is not designed to handle all 512 channels, significant speed improvements can be obtained.

Calling **`ArtNetReadWrite(Pipe)`** for a PipeEntry that is used for Art-Net data, will force an ArtDmx packet to be sent to the network. In this instance, the execution time of this function is approximately 1mS. However, sending data unnecessarily should be avoided, as it wastes network bandwidth.
For this reason it is also advisable to implement an intelligent application routine that only calls the function when necessary.

The Art-Net specification requires that data be refreshed every 3 seconds even when it has not changed. This refresh is handled internally for any pipes that have been enabled.

All pipes including the DMX-Dongle can be forced to refresh with a call to **`ArtNetReadWrite(-1).`**

FUNCTION PROTOTYPES

Control Functions

The first set of functions is used for control and configuration. These functions control both DMX-Dongle II and Art-Net operation.

ArtNetInit

Purpose:	Start up driver operation and default broadcast IP address.
Syntax:	**`BYTE ArtNetInit(char* NewIp)`**
Remarks:	This function loads the low level, platform dependent drivers and initialises the hardware. If first checks for the existence of a DMX-Dongle II on either of the three parallel port locations. It then checks for the existence of any Art-Net devices. Any devices found are automatically patched into the NodeTable and initialised. The application can then interrogate the NodeTable and reallocate devices if required. Any call to ArtNetInit must be terminated by a call to ArtNetEnd, in order to free the low level drivers and memory used by the API. NewIp is not used. It is retained for backwards compatibility. The flow chart below shows the function operation.

Return Value:	A WORD is returned. Test the low byte for 0x05.	
	Value	**Interpretation**
	0x0000	No devices found.
	0x0045	Devices detected, but power on test failed.
	0x0005	DMX-Dongle detected.
	0x0105	Art-Net devices detected on IP address 2.x.x.x
	0x0205	Art-Net devices detected on IP address 10.x.x.x
	0xfff0	Load succeeded. No DMX-Dongle or Art-Net devices detected. Options: Either call **`ArtNetEnd()`** and exit or: Continue application and wait for Art-Net devices to come on-line.
	0xffff	Load failed. Do not call **`ArtNetEnd()`** This return usually suggested that the driver has not been fully installed. The application should usually terminate if this response is seen.
See also:	ArtNetEnd(); DongleGetPortAddress()	

ArtNetEnd

Purpose:	Close drivers and free memory prior to application exit.
Syntax:	**`void ArtNetEnd (void):`**
Remarks:	
Return Value:	Void.
See also:	ArtNetInit(char* NewIp)

ArtNetReadWrite

Purpose:	Send local data buffers to and from hardware.
Syntax:	`bool ArtNetReadWrite(int Pipe)`
Remarks:	ArtNetReadWrite is used to transfer data to and from the hardware. The argument Pipe defines which block of the data buffer is to be used as shown in the table below.

Pipe	**Processes channels**
0	0-511
1	512-1023
2	1024-1535
3	1536-2047
19	9729-10240
-1	Process all channels

The data obtained from NodeLibrary defines whether a particular pipe is allocated to the DMX-Dongle or an Art-Net device.
The processing performed by ArtNetReadWrite() is dependent upon the hardware assigned to the pipe as shown in the following table:

Type	**Receive Data**	**Transmit Data**
DMX-Dongle II	Copied from DMX-Dongle to local buffer	Copied from local buffer to DMX-Dongle II
Art-Net	No affect. RxDmx is filled upon receipt of Art-Net data.	Copied from local buffer PipeLibrary[Pipe].TxDmx.Data to Art-Net ArtDmx packet.

Return Value:	True on success, False on any error.
See also:	DongleSetTransferCount()

ArtNetGetDll Revision

Purpose:	Return the revision number of the DLL.
Syntax:	**`BYTE ArtNetGetDllRevision(void):`**
Remarks:	It is useful for applications using the driver to display this number in the application about box.
Return Value:	Revision number.
See also:	DongleGetFirmwareRevision();

Generalised Data Functions

The Data Functions are provided for access to the transmit and receive data. The Data functions operate on all channels independent of whether the relevant pipe is set to DMX-Dongle of Art-Net operation.

ArtNetSetTxData

Purpose:	Set channel level in PipeLibrary[Pipe].TxDmx.Data
Syntax:	**`void ArtNetSetTxData(WORD Channel, BYTE Data)`**
Remarks:	This function simply sets the level of a channel in TxDmx.Data []. As the application 'owns' TxDmx.Data [], this function can be replaced with an array assignment. However, this function calculates which pipe entry to use. Note: The pipe entry calculation is based on the index, not the universe number. Channel is in the range 0 to MaxDriverChannel-1 Data is in the range 0 to 255
Return Value:	void
See also:	ArtNetSetTxDataGroup()

ArtNetSetTxDataGroup

Purpose:	Set range of channel levels in PipeLibrary[Pipe].TxDmx.Data
Syntax:	**`void ArtNetSetTxDataGroup(WORD StartCh, WORD EndCh, BYTE Data)`**
Remarks:	This function simply sets the level of a range of channels in TxDmx.Data []. As the application 'owns' TxDmx.Data [], this function can be replaced with an array assignment. However, this function calculates which pipe entry to use. Note: The pipe entry calculation is based on the index, not the universe number. Channel is in the range 0 to MaxDriverChannel-1 Data is in the range 0 to 255
Return Value:	Void
See also:	ArtNetSetTxData()

ArtNetGetRxData	Purpose:	Returns channel level from PipeLibrary[Pipe].RxDmx.Data[].
	Syntax:	**`BYTE ArtNetGetRxData(WORD Channel)`**
	Remarks:	This function simply returns the level of a channel in RxDmx.Data[].As the application 'owns' RxDmx.Data[], this function can be replaced with an array read. However, this function calculates which pipe entry to use. Note: The pipe entry calculation is based on the index, not the universe number. Channel is in the range 0 to MaxDriverChannel-1 Data is in the range 0 to 255
	Return Value:	void
	See also:	ArtNetSetTxData()

ArtNetClearDataBuffer	Purpose:	Set all PipeLibrary receive and transmit channels to a value.
	Syntax:	**`void ArtNetClearDataBuffer(BYTE Data)`**
	Remarks:	All transmit and receive data in all pipe entries is set to a value in the range 0 to 255
	Return Value:	void
	See also:	ArtNetSetTxDataGroup()

DMX-Dongle Control Functions

The DMX512 control functions are specific to the DMX-Dongle. They have no affect on pipes set to Art-Net operation.

DongleSetTransferCount

Purpose:	Set the number of channels to transfer in the ArtNetReadWrite() function.
Syntax:	**`void DongleSetTransferCount (WORD Number):`**
Remarks:	Number is in the range 24 -> 512. This function allows the application to significantly improve the execution time of ArtNetReadWrite when less than 512 channels are required.
Return Value:	
See also:	ArtNetInit()

DongleGetTransferCount

Purpose:	Get the number of channels transferred in the ArtNetReadWrite() function.
Syntax:	**`WORD DongleGetTransferCount (void):`**
Remarks:	
Return Value:	Number is in the range 24 -> 512
See also:	DongleSetTransferCount ()

DongleGetFirmwareRevision

Purpose:	Return the revision number of the DMX-Dongle.
Syntax:	**`BYTE DongleGetFirmwareRevision(void):`**
Remarks:	It is useful for applications using the driver to display this number in the application about box.
Return Value:	Firmware Revision number.
See also:	DongleGetDllRevision();

DongleGetPortAddress

Purpose:	Return the LPT number at which the DMX-Dongle was detected.
Syntax:	**`char DongleGetPortAddress(void):`**
Remarks:	
Return Value:	If the DMX-Dongle was detected, either 1, 2 or 3 is returned, meaning LPT1, LPT2 or LPT3. If the DMX-Dongle was not detected, 0 is returned.
See also:	ArtNetInit();

DongleGetType	Purpose:	Return the Type number of the DMX-Dongle.
	Syntax:	**`BYTE DongleGetType(void):`**
	Remarks:	
	Return Value:	If the DMX-Dongle is operating correctly, 0x05 is returned. If the DMX-Dongle is not detected, 0x00 is returned. If the DMX-Dongle failed the power on test, 0x45 is returned.
	See also:	ArtNetInit();

DongleSetRx StartCode	Purpose:	Set the start code to be used for receiving DMX512.
	Syntax:	**`void DongleSetRxStartCode(BYTE StartCode)`**
	Remarks:	This function is only applicable to the DMX-Dongle. All standard lighting data uses a start code of zero.
	Return Value:	void
	See also:	DongleSetTxStartCode()

DongleSetTx StartCode	Purpose:	Set the start code to be used for transmitting DMX512.
	Syntax:	**`void DongleSetTxStartCode(BYTE StartCode)`**
	Remarks:	This function is only applicable to the DMX-Dongle. All standard lighting data uses a start code of zero.
	Return Value:	void
	See also:	DongleSetRxStartCode()

DongleSetTx BreakTime	Purpose:	Set the DMX-Dongle transmit break time.
	Syntax:	**`void DongleSetTxBreakTime(BYTE Time)`**
	Remarks:	This function is only applicable to the DMX-Dongle. Time is calibrated in uS. The function allows out of range values to be set in order that 'bad' DMX512 can be generated for test purposes. Set to a value greater than 88 uS for valid DMX512.
	Return Value:	void
	See also:	

DongleSetTxMabTime		
	Purpose:	Set the DMX-Dongle transmit MaB time.
	Syntax:	`void DongleSetTxMabTime(BYTE Time)`
	Remarks:	This function is only applicable to the DMX-Dongle. Time is calibrated in uS. Valid settings are in multiples of 4uS in the range 2,6,10,14 -> 254. The function allows out of range values to be set in order that 'bad' DMX512 can be generated for test purposes. Set to a value greater than 8 uS for valid DMX512.
	Return Value:	void
	See also:	

DongleGetBreakTime		
	Purpose:	Retrieve the received DMX512 break time from the DMX-Dongle.
	Syntax:	`WORD DongleGetBreakTime(void)`
	Remarks:	This function is only applicable to the DMX-Dongle.
	Return Value:	The return value is the received DMX512 break time calibrated in uS.
	See also:	

DongleGetMabTime		
	Purpose:	Retrieve the received DMX512 MaB time from the DMX-Dongle.
	Syntax:	`WORD DongleGetMabTime(void)`
	Remarks:	This function is only applicable to the DMX-Dongle.
	Return Value:	The return value is the received DMX512 MaB time calibrated in uS.
	See also:	

DongleGetPeriod		
	Purpose:	Retrieve the received DMX512 cycle (or frame) time from the DMX-Dongle.
	Syntax:	`WORD DongleGetPeriod(void)`
	Remarks:	This function is only applicable to the DMX-Dongle.
	Return Value:	The return value is the received DMX512 cycle time calibrated in uS. The cycle time is the time between two consecutive start of breaks
	See also:	DongleGetFrequency();

DongleGet Frequency	Purpose:	Retrieve the frequency of DMX512 frames received from the DMX-Dongle.
	Syntax:	`WORD DongleGetFrequency(void)`
	Remarks:	This function is only applicable to the DMX-Dongle.
	Return Value:	The return value is the received DMX512 frame rate calibrated in Hertz. It is the reciprocal of DongleGetPeriod().
	See also:	DongleGetPeriod();
DongleGetChannel Count	Purpose:	Retrieve the number of channels received by the DMX-Dongle during the last frame of DMX512.
	Syntax:	`WORD DongleGetChannelCount(void)`
	Remarks:	This function is only applicable to the DMX-Dongle.
	Return Value:	The return value is in the range 1->512 if data is being received. If no data is received, the return value is zero.
	See also:	
DongleSetLoopOn	Purpose:	Set the DMX-Dongle to operate in loop through mode.
	Syntax:	`void DongleSetLoopOn(void)`
	Remarks:	This function is only applicable to the DMX-Dongle. This function connects the DMX512 input to the DMX512 output. The PC is able to read received data as normal, however transmit data will be ignored.
	Return Value:	
	See also:	DongleSetLoopOff()
DongleSetMerge On	Purpose:	Set the DMX-Dongle to operate in merge mode.
	Syntax:	`void DongleSetMergeOn(void)`
	Remarks:	This function is only applicable to the DMX-Dongle. This function sets the DMX-Dongle to htp merge received data with data transmitted by the PC. The PC is able to read received data as normal. This mode is cancelled by DongleSetLoopOff().
	Return Value:	
	See also:	DongleSetLoopOff()

DongleSetLoopOff

Purpose:	Set the DMX-Dongle to operate in normal mode.
Syntax:	`void DongleSetLoopOff(void)`
Remarks:	This function cancels both merge and loop through operation.
Return Value:	
See also:	DongleSetLoopOn();DongleSetMergeOn();

DongleSetAux

Purpose:	Write data to the aux control registers.
Syntax:	`void DongleSetAux(BYTE Aux, BYTE Data)`
Remarks:	This function is used for factory test and is not for application use. Aux is in the range 1-10 representing Aux1 -> Aux10.
Return Value:	
See also:	

DMX-Dongle Counter Functions

The Counter functions all relate to logging and counting functions provided by the DMX-Dongle. These functions no not relate to any pipes set to Art-Net operation.

DongleResetRxCounters

Purpose:	Reset the DMX-Dongle frame and error counters to zero.
Syntax:	`void DongleResetRxCounters(void)`
Remarks:	This function is only applicable to the DMX-Dongle.
Return Value:	void
See also:	

DongleGetPacketCount

Purpose:	Return the number of DMX512 frames received by the DMX-Dongle since the last ResetRxCounters().
Syntax:	`WORD DongleGetPacketCount(void)`
Remarks:	This function is only applicable to the DMX-Dongle.
Return Value:	Frame count.
See also:	ResetRxCounters()

DongleGetFramingErrorCount

Purpose:	Return the number of DMX512 framing errors received by the DMX-Dongle since the last ResetRxCounters().
Syntax:	`WORD DongleGetFramingErrorCount(void)`
Remarks:	This function is only applicable to the DMX-Dongle.
Return Value:	Framing error count.
See also:	ResetRxCounters()

DongleGetStartCodeErrorCount

Purpose:	Return the number of DMX512 frames received by the DMX-Dongle since the last ResetRxCounters() that have a start code not matching the currently selected receive start code.
Syntax:	`WORD DongleGetStartCodeErrorCount (void)`
Remarks:	This function is only applicable to the DMX-Dongle. Whilst the function name includes the word 'error', a non zero value should not be considered an error situation.
Return Value:	Non-matching start code count.
See also:	ResetRxCounters(); DongleSetRxStartCode();

DongleGetOverrunErrorCount

Purpose:	Return the number of DMX512 overrun errors received by the DMX-Dongle since the last ResetRxCounters().
Syntax:	**`WORD DongleGetOverrunErrorCount(void)`**
Remarks:	This function is only applicable to the DMX-Dongle.
Return Value:	void
See also:	ResetRxCounters();

Art-Net Command Functions

In most applications, the Art-Net specific functions are not required. Using ArtNetInit() and ArtNetReadWrite() will provide all the functionality required to transmit and receive data using both the DMX-Dongle and Art-Net devices.
These functions are provided to allow more sophisticated control of Art-Net devices.

ArtNetSendUdp	Purpose:	Send an Art-Net message to the network.
	Syntax:	`void ArtNetSendUdp(BYTE* Ptr, WORD Size)`
	Remarks:	Ptr is a pointer to a buffer of max size 5000 bytes. The buffer should be cast to the required Art-Net structure. Size is the size off the cast structure.
	Return Value:	void
	See also:	

	Code Example:	The following code example shows how to use the function to send an ArtAddress packet. This is used to remotely program the address switches and name of an Art-Net node.
		
		#include "Art-Net.h"
		#include "DongleArtNet.h"
		
		// load DLL and init
		
		T_ArtAddress ArtAddress;
		
		strncpy(ArtAddress.ID,"Art-Net",8);
		ArtAddress.VersionH=0;
		ArtAddress.Version=ProtocolVersion;
		ArtAddress.OpCode=OpAddress;
		ArtAddress.Filler1=0;
		ArtAddress.Filler2=0;
		strncpy(ArtAddress.ShortName,"Short Name", ShortNameLength);
		ArtAddress.ShortName[ShortNameLength-1]=0;
		strncpy(ArtAddress.LongName,"Long Name", LongNameLength);
		ArtAddress.LongName[LongNameLength-1]=0;
		
		ArtAddress.Swin[0]=0;
		ArtAddress.Swin[1]=1;
		ArtAddress.Swin[2]=2;
		ArtAddress.Swin[3]=3;
		ArtAddress.Swout[0]=4;
		ArtAddress.Swout[1]=5;
		ArtAddress.Swout[2]=6;
		ArtAddress.Swout[3]=7;
		ArtAddress.SwSub=0;
		
		ArtNetSendUdp(&((char)ArtAddress), sizeof(T_ArtAddress);

ArtNetSetCallBackUdpReceivePre		
	Purpose:	Assigns a user function that will be called prior to the driver software when Art-Net data is received.
	Syntax:	**`void ArtNetSetCallBackUdpReceivePre ( void(*Function)( BYTE* Ptr, int NumberByte, char* FromIp, bool* Handled) )`**
	Remarks:	The user supplied call back function is of the form: UserCallBack(BYTE* Ptr, int NumberBytes, char* FromIp, bool* Handled) The code in the user function is executed prior to the driver code. Ptr is a pointer to a buffer of max size 5000 bytes. NumberByte is the valid portion of the buffer. FromIp is a pointer to a string of char representing the IP address of the sender. It is in the form aaa.bbb.ccc.ddd. Handled is a pointer to bool. Set this to true if the driver should not handle this data. i.e. true == application handles data. false == driver handles data.
	Return Value:	void
	See also:	
	Code Example:	The following code example shows how to use the callback function to test for an ArtVideoSetup packet. node.

```
SomeFunction()
{
ArtNetSetCallBackUdpReceivePre(UserCallBack);
// tell the dll the address of the callback
}

// Define the application callback
void UserCallBack(BYTE* Ptr, int NumberBytes,
char* FromIp, bool* Handled)
{
switch(((T_ArtPoll*)Ptr->OpCode)
        {
        case OpVideoSetup:
              *Handled=true;
              // application should handle the packet
         default:
              *Handled =false;
              // else let the driver handle it
         }
}
```

ArtNetSetCallBackUdpReceivePost		
	Purpose:	Assigns a user function that will be called after the driver software when Art-Net data is received.
	Syntax:	**`void ArtNetSetCallBackUdpReceivePost ( void(*Function)( BYTE* Ptr, int NumberByte, char* FromIp) )`**
	Remarks:	The user supplied call back function is of the form: UserCallBack(BYTE* Ptr, int NumberBytes, char* FromIp) The code in the user function is executed after the driver code. Ptr is a pointer to a buffer of max size 5000 bytes. The buffer should be cast to the required Art-Net structure. Size is the size off the cast structure. NumberByte is the valid portion of the buffer. FromIp is a pointer to a string of char representing the IP address of the sender. It is in the form aaa.bbb.ccc.ddd.
	Return Value:	void
	See also:	

ArtNetSetCallBackUdpTrasmit		
	Purpose:	Assigns a user function that will be called just prior to the data being sent to the Ethernet output. This allows the application to parse all Art-Net output.
	Syntax:	**`void ArtNetSetCallBackUdpTransmit ( void(*Function)( BYTE* Ptr, int NumberByte) )`**
	Remarks:	The user supplied call back function is of the form: UserCallBack(BYTE* Ptr, int NumberBytes) The code in the user function is executed after the driver has configured the packet buffer and prior to transmission. Ptr is a pointer to a buffer of max size 5000 bytes. The buffer should be cast to the required Art-Net structure. Size is the size off the cast structure. NumberByte is the valid portion of the buffer.
	Return Value:	void
	See also:	

ArtNetCancelCallBack		
	Purpose:	This function cancels all user call back functions. It is executed automatically at DongleEnd().
	Syntax:	**`void ArtNetCancelCallBack( void )`**
	Remarks:	
	Return Value:	void
	See also:	

PipeLibrary Receive Functions

The following functions are used to access data related to receiving data within the PipeLibrary.

ArtNetSetRxPipeRequestUniverse

Purpose:	Sets the RequestUniverse entry in the PipeEntry structure indexed by Pipe. This is the 8 bit universe number that the application would like to read from this pipe. The application can set this value to a valid number in the range 0x00 -> 0xff. If Art-Net data matching this universe is received, it will be placed in the receive structure assigned to this pipe. The application may not know the Universe address of the data that it wishes to receive. In this case, the RequestUniverse can be set to the out of range value 0xffff. The driver will then assign the next received universe to the next available pipe. The application should then call ArtNetGetRxPipeRequestUniverse() to monitor for new data.
Syntax:	**`void ArtNetSetRxPipeRequestUniverse( int Pipe, BYTE Universe)`**
Remarks:	Pipe is in the range 0 -> MaxPipe-1 Universe is in the range 0x00 -> 0xff. The high nibble is the 'Sub-Net'.
Return Value:	Void
See also:	ArtNetReadWrite(); ArtNetEnumerate()

ArtNetGetRxPipeRequestUniverse

Purpose:	Return the RequestUniverse entry in the PipeEntry structure indexed by Pipe.
Syntax:	**`BYTE ArtNetGetRxPipeRequestUniverse( int Pipe)`**
Remarks:	Pipe is in the range 0 -> MaxPipe-1
Return Value:	RequestUniverse is in the range 0x00 -> 0xff. The high nibble is the 'Sub-Net'. If the return value is 0xffff, then no new Art-Net data has been received by this pipe.
See also:	ArtNetSetRxPipeRequestUniverse()

ArtNetGetRxPipe Universe	Purpose:	Return the Universe currently received by the PipeEntry structure indexed by Pipe.
	Syntax:	`BYTE ArtNetGetRxPipeUniverse( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Universe is in the range 0x00 -> 0xff. The high nibble relates to the 'Sub-Net' control on Artistic Licence products.
	See also:	ArtNetGetRxPipeRequestUniverse()
ArtNetGetRxPipe IndexForIp	Purpose:	Return the pipe index of the PipeEntry for which the receive IP address matches.
	Syntax:	`void ArtNetGetRxPipeIndexForIp( char* IpAddress )`
	Remarks:	IpAddress is a character string in the format www.xxx.yyy.zzzz.
	Return Value:	MaxPipe is returned if a match is not found
	See also:	ArtNetGetNodeIndexForIp()
ArtNetGetRxPipe IndexFor Universe	Purpose:	Return the pipe index of the PipeEntry for which the receive universe matches.
	Syntax:	`void ArtNetGetRxPipeIndexForUniverse( WORD Universe )`
	Remarks:	
	Return Value:	MaxPipe is returned if a match is not found
	See also:	ArtNetGetRxPipIndexForIp()
ArtNetIsRxPipe Active	Purpose:	Return the active status of this receive pipe.
	Syntax:	`bool ArtNetIsTxPipeActive( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if the RequestUniverse and the actual received universe match.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetIsPipe0 Dongle	Purpose:	Returns the connection status of the DMX-Dongle.
	Syntax:	`bool ArtNetIsPipe0Dongle(void)`
	Remarks:	
	Return Value:	True if the zeroth PipeLibrary is connected to the DMX-Dongle. False if in use by Art-Net devices.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetAnyArt NetReceived	Purpose:	Returns whether any ArtDmx packets have been received by any pipes.
	Syntax:	`bool ArtNetAnyArtNetReceived(void)`
	Remarks:	
	Return Value:	True if any Pipe has received ArtDmx packets.
	See also:	ArtNetIsRxPipeActive()

ArtNetGetNodeIndexForRxPipe	Purpose:	Cross references an entry in the receive part of PipeLibrary to the NodeLibrary.
	Syntax:	**`int ArtNetGetNodeIndexForRxPipe(void)`**
	Remarks:	
	Return Value:	Returns a NodeLibrary Index value if the IP addresses can be matched. Otherwise returns MaxNodes.
	See also:	ArtNetGetPipeIndexForUniverse
ArtNet Enumerate	Purpose:	Enumerates the Art-Net Network: All RequestUniverse entries in PipeLibrary are set to out of range value of 0xffff. An ArtPoll is then issued. All ArtPollReply packets are then parsed to configure the pipes based on nodes found on the network.
	Syntax:	**`void ArtNetEnumerate( void)`**
	Remarks:	This function is called as part of ArtNetInit(). It can be considered as a 'hard reset' of the network.
	Return Value:	Void
	See also:	ArtNetSoftEnumerate()
ArtNet SoftEnumerate	Purpose:	Enumerates the Art-Net Network: An ArtPoll is issued. All ArtPollReply packets are then parsed to configure any currently unused pipes based on nodes found on the network.
	Syntax:	**`void ArtNetSoftEnumerate( void)`**
	Remarks:	This function is used to add nodes to the PipeLibrary that were not detected when the driver started. The API issues this command on a three second cycle. It is therefore not necessary for the application to call this function.
	Return Value:	Void
	See also:	ArtNetEnumerate()
ArtNetGetRxPipe IpAddress	Purpose:	Return the IP address of the node from which data was last received.
	Syntax:	**`BYTE* ArtNetGetRxPipeIpAddress( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The return value is a pointer to a string of four unsigned characters. The most significant byte of the IP address is returned first. The default value is "0000"
	See also:	ArtNetSetRxPipeRequestUniverse()

ArtNetGetRxPipe EstaMan	Purpose:	Return the EstMan field of the node from which data was last received.
	Syntax:	`WORD ArtNetGetRxPipeEstaMan( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the ESTA manufacturer ID of the node from which data was last received. The default value is 0x0000.
	See also:	ArtNetGetNodeEstaMan()

ArtNetGetRxPipe EstaManString	Purpose:	Returns a string detailing the name of the Node manufacturer.
	Syntax:	`char* ArtNetGetRxPipeEstaManString( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Variable length string containing the manufacturer name. If the name is unknown, the return string indicates the hexadecimal value of this field.
	See also:	ArtNetGetRxPipeEstaMan()

ArtNetGetRxPipe OemH	Purpose:	Return the Art-Net OemH field of the node from which data was last received.
	Syntax:	`BYTE ArtNetGetRxPipeOemH( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the high byte of the Oem code of the node from which data was last received. The default value is 0x0000. The Oem code uniquely defines the node hardware platform.
	See also:	ArtNetSetRxPipeRequestUniverse()

ArtNetGetRxPipe Oem	Purpose:	Return the Art-Net Oem field of the node from which data was last received.
	Syntax:	`BYTE ArtNetGetRxPipeOem( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the low byte of the Oem code of the node from which data was last received. The default value is 0x0000. The Oem code uniquely defines the node hardware platform.
	See also:	ArtNetSetRxPipeRequestUniverse()

ArtNetGetRxPipe OemString	Purpose:	Returns a string detailing the product name and type of node from which data was last received.
	Syntax:	`char* ArtNetGetRxPipeEstaManString( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Variable length string containing the product name and type. If the name is unknown, the return string indicates the hexadecimal value of this field.
	See also:	ArtNetGetRxPipeOem()

ArtNetGetRxPipe VersionInfoH	Purpose:	Return the firmware revision of the node from which data was last received.
	Syntax:	**`BYTE ArtNetGetRxPipeVersionInfoH( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the high byte of the firmware revision of the node from which data was last received. The default value is 0x00.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetGetRxPipe VersionInfo	Purpose:	Return the firmware revision of the node from which data was last received.
	Syntax:	**`BYTE ArtNetGetRxPipeVersionInfo( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the low byte of the firmware revision of the node from which data was last received. The default value is 0x00.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetGetRxPipe NumberPorts	Purpose:	Return the number of receive DMX512 ports supported by the node from which data was last received.
	Syntax:	**`BYTE ArtNetGetRxPipeNumberPorts( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The maximum number of ports supported by an Art-Net node is 4. The return value is in the range 1 -> 4. The default value is 0x00. NB. A node with 4 ports would have 4 associated pipes.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetGetRx PipeRxReceived	Purpose:	Return the receive status of the node from which data was last received.
	Syntax:	**`bool ArtNetGetRxPipeRxReceived( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if data is being received.
	See also:	
ArtNetGetRx PipeRxErrors	Purpose:	Return the receive errors status of the node from which data was last received.
	Syntax:	**`bool ArtNetGetRxPipeRxErrors( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if errors detected.
	See also:	

ArtNetGetRx PipeRxTest	Purpose:	Return the receive test packet status of the node from which data was last received.
	Syntax:	`bool ArtNetGetRxPipeRxTest( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if test packets are being received.
	See also:	

ArtNetGetRx PipeRxSip	Purpose:	Return the system information packet status of the node from which data was last received.
	Syntax:	`bool ArtNetGetRxPipeRxSip( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if system information packets are being received.
	See also:	

ArtNetGetRx PipeRxText	Purpose:	Return the receive text packet status of the node from which data was last received.
	Syntax:	`bool ArtNetGetRxPipeRxText( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if text packets are being received.
	See also:	

ArtNetGetRx PipeRxDisable	Purpose:	Return the receive enable status of the node from which data was last received.
	Syntax:	`bool ArtNetGetRxPipeRxTest( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if this receiver has been disabled.
	See also:	

ArtNetGetRx PipeStatus Indicators	Purpose:	Returns the state of the node's front panel indicators. The Unknown option allows for earlier products that were shipped before this functionality was added to Art-Net.	
	Syntax:	`unsigned char ArtNetGetRxPipeStatusIndicators( int Pipe)`	
	Remarks:	Pipe is in the range 0 -> MaxPipe-1	
	Return Value:	0	Unknown.
		1	Locate.
		2	Mute.
		3	Normal.
	See also:		

ArtNetGetRx PipeStatus Authority	Purpose:	Returns the Programming Authority for the sub-net and universe settings. These settings may be set by the front panel control or programmed via an ArtAddress packet. The Unknown option allows for earlier products that were shipped before this functionality was added to Art-Net.	
	Syntax:	`unsigned char ArtNetGetRxPipeStatusAuthority( int Pipe)`	
	Remarks:	Pipe is in the range 0 -> MaxPipe-1	
	Return Value:	0	Unknown.
		1	Manual (front panel controls).
		2	Network (Set by an ArtAddress Packet).
		3	Normal.
	See also:		

ArtNetGetRx PipeStatusUbea Active	Purpose:	Returns the User Bios Extension Area (UBEA) status of the node from which data was last received.
	Syntax:	`bool ArtNetGetRxPipeStatusUbeaActive( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if the node has UBEA firmware installed. The UBEA allows third party software developers to supply 'plug-in' software enhancements for Art-Net nodes.
	See also:	

ArtNetGetRx PipeUbeaVersion	Purpose:	Returns the firmware version number of the User Bios Extension Area (UBEA) of the node from which data was last received.
	Syntax:	`BYTE ArtNetGetRxPipeUbeaVersion( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Zero is returned if the UBEA is not installed.
	See also:	

ArtNetGetRxPipeStatusRdmCapable	Purpose:	Returns the Remote Device Management (RDM) capabilities of the node from which data was last received.
	Syntax:	`bool ArtNetGetRxPipeStatusRdmCapable( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if the node is able to support RDM. RDM is a new DMX512-A feature allowing bi-directional communication.
	See also:	

ArtNetGetRxPipeStatusRomBoot	Purpose:	Returns the firmware status of the node from which data was last received.
	Syntax:	`bool ArtNetGetRxPipeStatusRomBoot( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if the node is executing original factory programmed firmware. False if the node is executing from non-volatile memory.
	See also:	

ArtNetGetRxPipeShortName	Purpose:	Return the short name of the node from which data was last received.
	Syntax:	`Char* ArtNetGetRxPipeShortName( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The short name.
	See also:	ArtNetGetRxPipeLongName()

ArtNetGetRxPipeLongName	Purpose:	Return the long name of the node from which data was last received.
	Syntax:	`Char* ArtNetGetRxPipeLongName( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The long name.
	See also:	ArtNetGetRxPipeShortName()

ArtNetGetRxPipeNodeReport	Purpose:	Return the text string status report of the node from which data was last received.
	Syntax:	`Char* ArtNetGetRxPipeNodeReport( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The Node Report contains human readable status information from the node, such as the results of it's power on test.
	See also:	ArtNetGetRxPipeLongName()

ArtNetGetRx PipeMacroKeys	Purpose:	Return the macro key bit fields of the node from which data was last received.
	Syntax:	`BYTE ArtNetGetRxPipeMacroKeys( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Byte containing bit fields representing the macro keys.
	See also:	ArtNetGetRxPipeRemoteKeys()

ArtNetGetRx PipeRemoteKeys	Purpose:	Return the remote key bit fields of the node from which data was last received.
	Syntax:	`BYTE ArtNetGetRxPipeRemoteKeys( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Byte containing bit fields representing the remote keys.
	See also:	ArtNetGetRxPipeMacroKeys()

ArtNetGetRx PipeVideo	Purpose:	Return the remote video display switch of the node from which data was last received.
	Syntax:	`BYTE ArtNetGetRxPipeVideo( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	
	See also:	ArtNetGetRxPipeMacroKeys()

PipeLibrary Transmit Functions

The following functions are used to access data related to transmitted data within the PipeLibrary.

ArtNetSetTxPipeUniverse

Purpose:	Sets the Universe entry in the PipeLibrary structure indexed by Pipe. This is the 8 bit universe number to which data will be sent when ArtNetReadWrite(Pipe) is called. This call enables transmission. ArtDmx packets will be sent automatically at three second intervals.
Syntax:	`void ArtNetSetTxPipeUniverse( int Pipe, BYTE Universe)`
Remarks:	Pipe is in the range 0 -> MaxPipe-1 Universe is in the range 0x00 -> 0xff. The high nibble is the 'Sub-Net'. This command is used to address the destination of this universe of data.
Return Value:	Void
See also:	ArtNetCancelTxPipe()

ArtNetCancelTxPipe

Purpose:	Disables transmission from the PipeLibrary structure indexed by Pipe.
Syntax:	`void ArtNetCancelTxPipe( int Pipe)`
Remarks:	Pipe is in the range 0 -> MaxPipe-1
Return Value:	Void
See also:	ArtNetSetTxPipeUniverse()

ArtNetGetTxPipeUniverse

Purpose:	Returns the Universe entry in the PipeLibrary structure indexed by Pipe. This is the 8 bit universe number to which data will be sent when ArtNetReadWrite(Pipe) is called.
Syntax:	`BYTE ArtNetGetTxPipeUniverse( int Pipe)`
Remarks:	Pipe is in the range 0 -> MaxPipe-1
Return Value:	Universe is in the range 0x00 -> 0xff. The high nibble is the 'Sub-Net'.
See also:	ArtNetReadWrite()

ArtNetIsTxPipeActive

Purpose:	Return the enabled status of this transmit pipe.
Syntax:	`bool ArtNetIsTxPipeActive( int Pipe)`
Remarks:	Pipe is in the range 0 -> MaxPipe-1
Return Value:	True if TxEnable is set for this entry in the PipeLibrary.
See also:	ArtNetSetTxPipeUniverse()

ArtNetGetNode IndexForTxPipe	Purpose:	Cross references an entry in the transmit part of PipeLibrary to the NodeLibrary.
	Syntax:	**`int ArtNetGetNodeIndexForTxPipe(void)`**
	Remarks:	
	Return Value:	Returns a NodeLibrary Index value if the IP addresses can be matched. Otherwise returns MaxNodes.
	See also:	ArtNetGetPipeIndexForUniverse
ArtNetGetTxPipe IpAddress	Purpose:	Return the IP address of the node to which data was last sent.
	Syntax:	**`BYTE* ArtNetGetTxPipeIpAddress( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The return value is a pointer to a string of four unsigned characters. The most significant byte of the IP address is returned first. The default value is "0000"
	See also:	ArtNetGeRxPipeIpAddress()
ArtNetGetTxPipe EstaMan	Purpose:	Return the EstMan field of the node to which data was last sent.
	Syntax:	**`WORD ArtNetGetTxPipeEstaMan( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the ESTA manufacturer ID of the node to which data was last sent. The default value is 0x0000.
	See also:	ArtNetGetNodeEstaMan()
ArtNetGetTxPipe EstaManString	Purpose:	Returns a string detailing the name of the Node manufacturer.
	Syntax:	**`char* ArtNetGetTxPipeEstaManString( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Variable length string containing the manufacturer name. If the name is unknown, the return string indicates the hexadecimal value of this field.
	See also:	ArtNetGetTxPipeEstaMan()
ArtNetGetTxPipe OemH	Purpose:	Return the Art-Net OemH field of the node to which data was last sent.
	Syntax:	**`BYTE ArtNetGetTxPipeOemH( int Pipe)`**
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the high byte of the Oem code of the node to which data was last sent. The default value is 0x0000. The Oem code uniquely defines the node hardware platform.
	See also:	ArtNetSetRxPipeRequestUniverse()

ArtNetGetTxPipeOem	Purpose:	Return the Art-Net Oem field of the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetTxPipeOem( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the low byte of the Oem code of the node to which data was last sent. The default value is 0x0000. The Oem code uniquely defines the node hardware platform.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetGetTxPipeOemString	Purpose:	Returns a string detailing the product name and type of the node to which data was last sent.
	Syntax:	`char* ArtNetGetTxPipeEstaManString( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Variable length string containing the product name and type. If the name is unknown, the return string indicates the hexadecimal value of this field.
	See also:	ArtNetGetRxPipeOem()
ArtNetGetTxPipeVersionInfoH	Purpose:	Return the firmware revision of the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetTxPipeVersionInfoH( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the high byte of the firmware revision of the node to which data was last sent. The default value is 0x00.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetGetTxPipeVersionInfo	Purpose:	Return the firmware revision of the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetRxPipeVersionInfo( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	This field contains the low byte of the firmware revision of the node to which data was last sent. The default value is 0x00.
	See also:	ArtNetSetRxPipeRequestUniverse()
ArtNetGetTxPipeNumberPorts	Purpose:	Return the number of transmit DMX512 ports supported by the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetTxPipeNumberPorts( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The maximum number of ports supported by an Art-Net node is 4. The return value is in the range 1 -> 4. The default value is 0x00. NB. A node with 4 ports would have 4 associated pipes.
	See also:	ArtNetSetRxPipeRequestUniverse()

ArtNetGetTxPipeStatusUbeaActive	Purpose:	Returns the User Bios Extension Area (UBEA) status of the node to which data was last sent.
	Syntax:	`bool ArtNetGetTxPipeStatusUbeaActive( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if the node has UBEA firmware installed. The UBEA allows third party software developers to supply 'plug-in' software enhancements for Art-Net nodes.
	See also:	

ArtNetGetTxPipeUbeaVersion	Purpose:	Returns the firmware version number of the User Bios Extension Area (UBEA) of the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetTxPipeUbeaVersion( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Zero is returned if the UBEA is not installed.
	See also:	

ArtNetGetTxPipeStatusRdmCapable	Purpose:	Returns the Remote Device Management (RDM) capabilities of the node to which data was last sent.
	Syntax:	`bool ArtNetGetTxPipeStatusRdmCapable( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if the node is able to support RDM. RDM is a new DMX512-A feature allowing bi-directional communication.
	See also:	

ArtNetGetTxPipeStatusRomBoot	Purpose:	Returns the firmware status of the node to which data was last sent.
	Syntax:	`bool ArtNetGetTxPipeStatusRomBoot( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	True if the node is executing original factory programmed firmware. False if the node is executing from non-volatile memory.
	See also:	

ArtNetGetTxPipeShortName	Purpose:	Return the short name of the node from which data was last received.
	Syntax:	`Char* ArtNetGetTxPipeShortName( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The short name.
	See also:	ArtNetGetRxPipeLongName()

ArtNetGetTxPipe LongName	Purpose:	Return the long name of the node to which data was last sent.
	Syntax:	`Char* ArtNetGetTxPipeLongName( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The long name.
	See also:	ArtNetGetRxPipeShortName()

ArtNetGetTxPipe NodeReport	Purpose:	Return the text string status report of the node to which data was last sent.
	Syntax:	`Char* ArtNetGetTxPipeNodeReport( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	The Node Report contains human readable status information from the node, such as the results of it's power on test.
	See also:	ArtNetGetRxPipeLongName()

ArtNetGetTxPipe MacroKeys	Purpose:	Return the macro key bit fields of the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetTxPipeMacroKeys( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Byte containing bit fields representing the macro keys.
	See also:	ArtNetGetRxPipeRemoteKeys()

ArtNetGetTxPipe RemoteKeys	Purpose:	Return the remote key bit fields of the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetTxPipeRemoteKeys( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	Byte containing bit fields representing the remote keys.
	See also:	ArtNetGetRxPipeMacroKeys()

ArtNetGetTxPipe Video	Purpose:	Return the remote video display switch of the node to which data was last sent.
	Syntax:	`BYTE ArtNetGetTxPipeVideo( int Pipe)`
	Remarks:	Pipe is in the range 0 -> MaxPipe-1
	Return Value:	
	See also:	ArtNetGetRxPipeMacroKeys()

NodeLibrary Functions

The following functions are used to access data from the NodeLibrary.

ArtNetGetNodeIpAddress

Purpose:	Return the IP address of the node.
Syntax:	**`char* ArtNetGetNodeIpAddress( int Node)`**
Remarks:	Node is in the range 0 -> MaxNode-1
Return Value:	Returns a string in the format www.xxx.yyy.zzz. If the node has not received data, 000.000.000.000 will be returned
See also:	

ArtNetGetNodeTxUniverse

Purpose:	Returns the transmit Universe entry in the NodeLibrary structure indexed by Node for the specified universe. This is the 8 bit universe number to which ArtDmx data will be sent.
Syntax:	**`BYTE ArtNetGetNodeTxUniverse( int Node, int Uni)`**
Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
Return Value:	Universe is in the range 0x00 -> 0xff. The high nibble is the 'Sub-Net'.
See also:	ArtNetReadWrite()

ArtNetGetNodeIndexForIp

Purpose:	Return the node index of the NodeEntry for which the IP address matches.
Syntax:	**`void ArtNetGetNodeIndexForIp( char* IpAddress )`**
Remarks:	IpAddress is a character string in the format www.xxx.yyy.zzzz.
Return Value:	MaxNode is returned if a match is not found
See also:	ArtNetGeRxPipetIndexForIp()

ArtNetGetNodeTxImplemented

Purpose:	Returns whether this transmit universe is physically implemented for the specified universe.
Syntax:	**`unsigned int ArtNetGetNodeTxImplemented( int Node, int Uni)`**
Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
Return Value:	True if the port exists.
See also:	ArtNetReadWrite()

ArtNetGetNodeRxUniverse	Purpose:	Returns the receive Universe entry in the NodeLibrary structure indexed by Node for the specified universe. This is the 8 bit universe number from which ArtDmx data will be received.
	Syntax:	`BYTE ArtNetGetNodeRxUniverse( int Node, int Uni)`
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	Universe is in the range 0x00 -> 0xff. The high nibble is the 'Sub-Net'.
	See also:	ArtNetReadWrite()
ArtNetGetNodeRxImplemented	Purpose:	Returns whether this receive universe is physically implemented for the specified universe.
	Syntax:	`unsigned int ArtNetGetNodeRxImplemented( int Node, int Uni)`
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if the port exists.
	See also:	ArtNetReadWrite()
ArtNetGetNodeEstaMan	Purpose:	Return the EstMan field of the node.
	Syntax:	`WORD ArtNetGetNodeEstaMan( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	This field contains the ESTA manufacturer ID of the node to which data was last sent. The default value is 0x0000.
	See also:	
ArtNetGetNodeEstaManString	Purpose:	Returns a string detailing the name of the Node manufacturer.
	Syntax:	`char* ArtNetGetNodeEstaManString( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	Variable length string containing the manufacturer name. If the name is unknown, the return string indicates the hexadecimal value of this field.
	See also:	
ArtNetGetNodeOemH	Purpose:	Return the Art-Net OemH field.
	Syntax:	`BYTE ArtNetGetNodeOemH( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	This field contains the high byte of the Oem code of the node. The default value is 0x0000. The Oem code uniquely defines the node hardware platform.
	See also:	

ArtNetGetNodeOem	Purpose:	Return the Art-Net Oem field.
	Syntax:	`BYTE ArtNetGetNodeOem( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	This field contains the low byte of the Oem code of the node. The default value is 0x0000. The Oem code uniquely defines the node hardware platform.
	See also:	

ArtNetGetNodeOemString	Purpose:	Returns a string detailing the product name and type of the node.
	Syntax:	`char* ArtNetGetNodeOemString( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	Variable length string containing the product name and type. If the name is unknown, the return string indicates the hexadecimal value of this field.
	See also:	

ArtNetGetNodeVersionInfoH	Purpose:	Return the firmware revision of the node.
	Syntax:	`BYTE ArtNetGetNodeVersionInfoH( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	This field contains the high byte of the firmware revision of the node. The default value is 0x00.
	See also:	

ArtNetGetNodeVersionInfo	Purpose:	Return the firmware revision of the node.
	Syntax:	`BYTE ArtNetGetNodeVersionInfo( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	This field contains the low byte of the firmware revision of the node. The default value is 0x00.
	See also:	

ArtNetGetTxPipeNumberPorts	Purpose:	Return the number of DMX512 ports supported by the node.
	Syntax:	`BYTE ArtNetGetNodeNumberPorts( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	The maximum number of ports supported by an Art-Net node is 4. The return value is in the range 1 -> 4. The default value is 0x00. NB. A node with 4 ports would have 4 associated pipes.
	See also:	

ArtNetGetNodeStatusIndicators	Purpose:	Returns the state of the node's front panel indicators.	
	Syntax:	`bool ArtNetGetNodeStatusIndicators( int Node)`	
	Remarks:	Node is in the range 0 -> MaxNode-1	
	Return Value:	0	Unknown.
		1	Locate.
		2	Mute.
		3	Normal.
	See also:		

ArtNetGetNodeStatusAuthority	Purpose:	Returns the Programming Authority for the sub-net and universe settings. These settings may be set by the front panel control or programmed via an ArtAddress packet. The Unknown option allows for earlier products that were shipped before this functionality was added to Art-Net.	
	Syntax:	`bool ArtNetGetNodeStatusAuthority( int Node)`	
	Remarks:	Pipe is in the range 0 -> MaxPipe-1	
	Return Value:	0	Unknown.
		1	Manual (front panel controls).
		2	Network (Set by an ArtAddress Packet).
		3	Normal.
	See also:		

ArtNetGetNodeStatusUbea Active	Purpose:	Returns the User Bios Extension Area (UBEA) status of the node.
	Syntax:	`bool ArtNetGetNodeStatusUbeaActive( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	True if the node has UBEA firmware installed. The UBEA allows third party software developers to supply 'plug-in' software enhancements for Art-Net nodes.
	See also:	

ArtNetGetNodeUbeaVersion	Purpose:	Returns the firmware version number of the User Bios Extension Area (UBEA) of the node.
	Syntax:	`BYTE ArtNetGetNodeUbeaVersion( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	Zero is returned if the UBEA is not installed.
	See also:	

ArtNetGetNodeStatusRdmCapable	Purpose:	Returns the Remote Device Management (RDM) capabilities of the node.
	Syntax:	`bool ArtNetGetNodeStatusRdmCapable( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	True if the node is able to support RDM. RDM is a new DMX512-A feature allowing bi-directional communication.
	See also:	

ArtNetGetNodeStatusRomBoot	Purpose:	Returns the firmware status of the.
	Syntax:	`bool ArtNetGetNodeStatusRomBoot( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	True if the node is executing original factory programmed firmware. False if the node is executing from non-volatile memory.
	See also:	

ArtNetGetNodeShortName	Purpose:	Return the short name of the node.
	Syntax:	`char* ArtNetGetNodeShortName( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	The short name.
	See also:	

ArtNetGetNodeLongName	Purpose:	Return the long name of the.
	Syntax:	`char* ArtNetGetNodeLongName( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	The long name.
	See also:	

ArtNetGetNodeNodeReport	Purpose:	Return the text string status report of the node.
	Syntax:	`Char* ArtNetGetNodeNodeReport( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	The Node Report contains human readable status information from the node, such as the results of it's power on test.
	See also:	

ArtNetGetNodeMacroKeys	Purpose:	Return the macro key bit fields of the node
	Syntax:	`BYTE ArtNetGetNodeMacroKeys( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	Byte containing bit fields representing the macro keys.
	See also:	

ArtNetGetNodeRemoteKeys	Purpose:	Return the remote key bit fields of the node.
	Syntax:	`BYTE ArtNetGetNodeRemoteKeys( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	Byte containing bit fields representing the remote keys.
	See also:	

ArtNetGetNodeVideo	Purpose:	Return the remote video display switch of the node.
	Syntax:	`BYTE ArtNetGetNodeVideo( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	
	See also:	

ArtNetGetNodeVersionInfoH	Purpose:	Return the firmware revision of the node.
	Syntax:	`BYTE ArtNetGetNodeVersionInfoH( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	This field contains the high byte of the firmware revision of the node. The default value is 0x00.
	See also:	

ArtNetGetNodeVersionInfo	Purpose:	Return the firmware revision of the node.
	Syntax:	`BYTE ArtNetGetNodeVersionInfo( int Node)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	This field contains the low byte of the firmware revision of the node. The default value is 0x00.
	See also:	

ArtNetGetNodeRxReceived	Purpose:	Return the receive status of the node.
	Syntax:	**`bool ArtNetGetNodeRxReceived(int Node, int Uni)`**
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if data is being received.
	See also:	

ArtNetGetNodeRxErrors	Purpose:	Return the receive errors status of the node.
	Syntax:	**`bool ArtNetGetNodeRxErrors(int Node, int Uni)`**
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if data errors are detected.
	See also:	

ArtNetGetNodeRxTest	Purpose:	Return the receive test packet status of the node.
	Syntax:	**`bool ArtNetGetNodeRxTest(int Node, int Uni)`**
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if test packets are being received.
	See also:	

ArtNetGetNodeRxSip	Purpose:	Return the system information test packet status of the node.
	Syntax:	**`bool ArtNetGetNodeRxSip(int Node, int Uni)`**
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if system information packets are being received.
	See also:	

ArtNetGetNodeRxText	Purpose:	Return the receive text packet status of the node.
	Syntax:	**`bool ArtNetGetNodeRxText(int Node, int Uni)`**
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if text packets are being received.
	See also:	

ArtNetGetNodeRxDisable	Purpose:	Return the receive enable status of the node.
	Syntax:	**`bool ArtNetGetNodeRxDisable(int Node, int Uni)`**
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if this receiver has been disabled.
	See also:	

ArtNetGetNodeTxGood	Purpose:	Return the transmit active status of the node.
	Syntax:	`bool ArtNetGetNodeTxGood(int Node, int Uni)`
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if this transmitter is currently transmitting DMX512.
	See also:	
ArtNetGetNodeTxMerge	Purpose:	Return the transmit merge status of the node.
	Syntax:	`bool ArtNetGetNodeTxMerge(int Node, int Uni)`
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if this transmitter is currently merging data from two sources.
	See also:	
ArtNetGetNodeTxLtp	Purpose:	Return the transmit merge mode of the node.
	Syntax:	`bool ArtNetGetNodeTxLtp(int Node, int Uni)`
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if this transmitter is set to merge in LTP. False if Htp.
	See also:	
ArtNetGetNodeTxShort	Purpose:	Return the transmit hardware driver status of the node.
	Syntax:	`bool ArtNetGetNodeTxShort(int Node, int Uni)`
	Remarks:	Node is in the range 0 -> MaxNode-1 Uni is in the range 0 -> 3
	Return Value:	True if this transmitter output is shorted.
	See also:	
ArtNetSetNodeShortName	Purpose:	Sets the short name of a node.
	Syntax:	`void ArtNetSeNodeShortName( int Node char* Name)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	Void
	See also:	
ArtNetSetNodeLongName	Purpose:	Sets the long name of the node.
	Syntax:	`void ArtNetSetIpLongName (int Node char* Name)`
	Remarks:	Node is in the range 0 -> MaxNode-1
	Return Value:	Void
	See also:	

ArtNetSetNode Switches

Purpose:	Sets the front panel switches of the node.
Syntax:	`void ArtNetSetIpSwitches( int Node, BYTE In0, BYTE In1, BYTE In2, BYTE In3, BYTE Out0, BYTE Out1, BYTE Out2, BYTE Out3, BYTE SubNet, BYTE Video)`
Remarks:	The In parameters represent the low nibble input universe address of the four possible DMX512 input ports supported by the node. The Out parameters represent the low nibble output universe address of the four possible DMX512 input ports supported by the node. The SubNet parameter is the high nibble of all input and output universe addresses. Each parameter will be ignored unless bit 7 is set.
Return Value:	Void
See also:	

Example:	To set a DMX-Hub at IP address 2.0.0.1 (which has 4 inputs and 4 outputs) to input on universes 10, 11, 12, 13, and output on universes 16, 17, 18, 19 without affecting the video switch setting, send:
	
	`ArtNetSetIpSwitches(`
	`        0,        // NodeLib entry`
	`        0x80,     // low byte input 0 address`
	`        0x81,`
	`        0x82,     // NB Bit 7 set to enable change`
	`        0x83,`
	`        0x86,     // low byte output 0 address`
	`        0x87,`
	`        0x88,`
	`        0x89,`
	`        0x81,     // hi byte of in & out address`
	`        0x00      // bit 7 not set, so no affect`
	`    );`

ArtNetSetNode Command

Purpose:	Sets the long name of the node that matches IP address.
Syntax:	`void ArtNetSetNodeCommand(int Node, BYTE NewCommand)`
Remarks:	Node is in the range 0 -> MaxNode-1 NewCommand is defined as follows:

Mnemonic	Value	Function
AcNone	0	No Action.
AcCancelMerge	1	The next ArtDmx packet cancels Node's Merge.
AcLedNormal	2	Node front panel indicators operate normally
AcLedMute	3	Node front panel indicators are muted
AcLedLocate	4	Node front panel indicators all flash for locating device.
AcResetRxFlags	5	Resets the Node's Sip, Text, Test and data error flags.
AcMergeLtp0	0x10	Set DMX Port 0 to Merge in LTP mode.
AcMergeLtp1	0x11	Set DMX Port 1 to Merge in LTP mode.
AcMergeLtp2	0x12	Set DMX Port 2 to Merge in LTP mode.
AcMergeLtp3	0x13	Set DMX Port 3 to Merge in LTP mode.
AcMergeHtp0	0x50	Set DMX Port 0 to Merge in HTP.
AcMergeHtp1	0x51	Set DMX Port 1 to Merge in HTP.
AcMergeHtp2	0x52	Set DMX Port 2 to Merge in HTP.
AcMergeHtp3	0x53	Set DMX Port 3 to Merge in HTP.
AcClearOp0	0x90	Clear DMX Output buffer for Port 0
AcClearOp1	0x91	Clear DMX Output buffer for Port 1
AcClearOp2	0x92	Clear DMX Output buffer for Port 2
AcClearOp3	0x93	Clear DMX Output buffer for Port 3

Return Value:	void
See also:	

Art-Net Counter Functions

These counter functions are generalised for operation with both Art-Net and DMX-Dongle II.

ArtNetResetCounters

Purpose:	Reset the packet counters for both transmit and receive.
Syntax:	**`void ArtNetResetCounters(void)`**
Remarks:	
Return Value:	void
See also:	

ArtNetGetRxPipePacketCount

Purpose:	Return the number of receive packets for this pipe since the last ArtNetResetCounters.
Syntax:	**`WORD ArtNetGetRxPipePacketCount(void)`**
Remarks:	If the pipe is receiving Art-Net, this cis the number of ArtDmx packets received. If the pipe is connected to the DMX-Dongle II, this is the number of DMX512 packets received.
Return Value:	Packet count.
See also:	ArtNetResetCounters()

ArtNetGetTxPipePacketCount

Purpose:	Return the number of transmit packets sent via this pipe since the last ArtNetResetCounters.
Syntax:	**`WORD ArtNetGetTxPipePacketCount(void)`**
Remarks:	If the pipe is transmitting Art-Net, this is the number of ArtDmx packets sent. If the pipe is connected to the DMX-Dongle II, this is the number calls to ArtNetReadWrite(0).
Return Value:	Packet count.
See also:	ArtNetResetCounters()

ArtNetGetRxPipeChannelCount

Purpose:	Return the number of DMX512 channels (slots) received by this pipe.
Syntax:	**`WORD ArtNetGetRxPipeChannelCount(void)`**
Remarks:	
Return Value:	0 if no data received, otherwise in the range 1 -> 512.
See also:	ArtNetResetCounters()

Dialogue Display Functions

The following group of functions are provided to allow the application access to the dialogues available from the driver tray icon.
These dialogues only use system resources when visible. Forms and timer resources are all dynamically allocated.
All dialogues are 'stay on top'.

ArtNetShowScopeForm

Purpose:	Displays the 'Scope' Form which contains an 'oscilloscope' and a 'digital meter' for channel levels.
Syntax:	`bool ArtNetShowScopeForm(void)`
Remarks:	
Return Value:	False if not enough resources to create dialogue.
See also:	

ArtNetShowAboutBox

Purpose:	Displays the driver about box which among other items contains the credits for all us hard working programmers!
Syntax:	`bool ArtNetShowAboutBox(void)`
Remarks:	
Return Value:	False if not enough resources to create dialogue.
See also:	

ArtNetShowNetworkStatusForm

Purpose:	Displays the Network Status Form. This is a listing of all Art-Net devices detected. It provides status monitoring. The right click popup menu is used for firmware upload.
Syntax:	`bool ArtNetShowNetworkStatusForm(void)`
Remarks:	
Return Value:	False if not enough resources to create dialogue.
See also:	

ArtNetShowDataMonitorForm

Purpose:	Displays the data monitor form which displays an entire 512 channel page of any transmit or receive pipe. When viewing DMX-Dongle II receive data, it also displays the timing analyser.
Syntax:	`bool ArtNetShowDataMonitorForm(void)`
Remarks:	
Return Value:	False if not enough resources to create dialogue.
See also:	

MS-DOS DEVELOPMENT

Overview

The Dos DMX-Dongle library is supplied on CD. The User Guide describes the installation procedure.

Please note that the *.LIB files are compiled for Borland BC V3.2. Users of other compilers will need to recompile the source code. The files are as follows:

❑ DONGLE.C	Library source code
❑ DONGLE.H	Library header file
❑ TC_CDONG.LIB	Library Borland compatible, Compact model
❑ TC_SDONG.LIB	Library Borland compatible, Small model
❑ TC_MDONG.LIB	Library Borland compatible, Medium model
❑ TC_LDONG.LIB	Library Borland compatible, Large model
❑ TC_HDONG.LIB	Library Borland compatible, Huge model
❑ CHECK.C	Example program

Initialising the DMX-Dongle

The following code skeleton shows how to detect and initialise the DMX-Dongle and transmit DMX512.
This listing is the check.exe program that is supplied with the DMX-Dongle. It also calculates how fast the computer can access the DMX-Dongle.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<bios.h>
#include<dos.h>
#include<stdlib.h>
#include<ctype.h>
#include<time.h>
#include"c:\tc\dongle\lib\dongle.h"

char trial(unsigned int addr);
void main()
{           /* Search for a Dongle on standard locations */
if(trial(0x378))
    exit(0);
printf("DMX-Dongle not found at 0x378\r\n");
if(trial(0x278))
    exit(0);
printf("DMX-Dongle not found at 0x278\r\n");
if(trial(0x3bc))
    exit(0);
printf("DMX-Dongle not found at 0x3bc\r\n");
end_dongle();
exit(0);
}/* end main */
char trial(unsigned int addr)
{
long start_time;
char count=100;
if(init_dongle(addr)==0x5)              /* Dongle type shows it’s there ! */
    {
    printf(“DMX-Dongle found at address 0x%x\r\n”,addr);
    printf(“Dongle type number = %d\r\n”,get_dongle_type());
    printf(“Dongle firmware revision = %d\r\n\r\n”,get_firmware());
    printf(“Please Wait for Benchmark.\r\n ”);
    clr_memory();

    /* Time 100 DMX transmit frames to calculate the maximum rate. */

    start_time=biostime(0,0);
    while(count--)
        tx_dmx(1);
    printf(“This PC requires %dmS to transmit a full DMX frame.\r\n\r\n”,
        ((biostime(0,0)-start_time)*55)/100);
    end_dongle();
    return(YES);
    }
return(NO);
}
```

Receiving DMX512

The following code skeleton shows how to receive DMX512.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<bios.h>
#include<dos.h>
#include<stdlib.h>
#include<ctype.h>
#include<time.h>
#include”c:\tc\dongle\lib\dongle.h”

void main()
{
char receive[MAX_CHAN];

if(init_dongle(0)==0x5)
    {
    set_rx_header(0);
    reset_rx_base(0);

    /* read 128 channels of dmx data starting at channel 1 */

    rx_dmx_ptr(receive);

    /* handle data */
    end_dongle();
    }
exit(0);
}/* end main */
```

FUNCTION PROTOTYPES

init_dongle

Purpose:	Start up dongle operation and specify port address.
Syntax:	unsigned char init_dongle(unsigned int addr)
Remarks:	Set addr=0 to use the default library address for the printer port.
Return Value:	Dongle type which should be checked for 0x05 for Dongle. The Dongle II returns 0x45 if the power on ..... test fails.
See also:	end_dongle

end_dongle

Purpose:	End dongle operation.
Syntax:	char end_dongle(void)
Remarks:	Final call to library before exiting application.
Return Value:	Void.
See also:	init_dongle

tx_dmx

Purpose:	Transmit a 512 byte memory of dmx data.
Syntax:	char tx_dmx(memory)
Remarks:	Data is taken from mem array indexed by memory.
Return Value:	YES if successful.
See also:	

tx_control_block

Purpose:	Send command and header to dongle.
Syntax:	char tx_control_block(char command)
Remarks:	Also updates rx_ctrl structure from dongle.
Return Value:	YES if successful.
See also:	

Is_dmx_block_finished

Purpose:	Hold until dongle has finished transmitting a .... 128/256/512 byte block of data.
Syntax:	char is_dmx_block_finished(void)
Remarks:	Used by tx_dmx to pause between data blocks. The ......
Return Value:	YES.
See also:	

tx_byte

Purpose:	Send two nibbles of data to dongle, and retrieve next two nibbles from dongle.
Syntax:	char tx_byte(unsigned int address, unsigned char data)
Remarks:	Control status inactive during transfer
Return Value:	Next data byte from dongle.
See also:	tx_byte_ctrl

tx_nibble

Purpose:	Send nibbles of data to dongle, and retrieve next nibble from dongle.
Syntax:	char tx_nibble(unsigned int address, unsigned char data)
Remarks:	Control status inactive during transfer
Return Value:	Next data nibble from dongle.
See also:	tx_nibble_ctrl

tx_byte_ctrl

Purpose:	Send two nibbles of data to dongle, and retrieve next two nibbles from dongle.
Syntax:	char tx_byte_ctrl(unsigned int address, unsigned char data)
Remarks:	Control status active during transfer
Return Value:	Next data byte from dongle.

tx_nibble_ctrl

Purpose:	Send nibbles of data to dongle, and retrieve next nibble from dongle.
Syntax:	char tx_nibble_ctrl(unsigned int address, unsigned .... char data)
Remarks:	Control status active during transfer
Return Value:	Next data nibble from dongle.
See also:	tx_nibble

process_return_data

Purpose:	Adjust nibble from dongle for bit shift caused by printer port hardware.
Syntax:	char process_return_data(char ret)
Remarks:	Used by tx_byte and tx_nibble to re frame data returned by dongle.
Return Value:	High nibble contains data.
See also:	

rx_dmx

Purpose:	Update rx structure with 128, 256 or 512 bytes of received dmx data. Data copied to structure is offset by the current receive base address. Data size is 256 for firmware version 0x17, 512 for version 0x20.
Syntax:	char rx_dmx(void)
Remarks:	Only for use in uncompressed mode
Return Value:	YES
See also:	rx_dmx_ptr

rx_dmx_ptr

Purpose:	Update a buffer supplied by caller with receive dmx data. The supplied buffer should be 512 characters wide.
Syntax:	char rx_dmx_ptr(char* ptr)
Remarks:	Operation of this function is modified by the global variable dongle_compress. dongle_compress=NO. 128, 256 or 512 bytes of data are transferred from the dongle. the data represents the 128 channels received after the current base address. Data size is 256 for firmware version 0x17, 512 for version 0x20. dongle_compress-YES. 512 bytes of data are transferred from the dongle. The data represents all 512 channels received. The data is converted from two bit resolution to 8 bit resolution as it is transferred. To use this mode effectively the receive base address should be set to 1
Return Value:	YES.
See also:	set_compress, get_compress, rx_dmx

wait_for_break

Purpose:	Hold until the dongle detects the next receive dmx frame or a timeout occurs.
Syntax:	char wait_for_break(void)
Remarks:	
Return Value:	YES on break, NO on timeout.
See also:	

rx_reset_counters

Purpose:	Reset dongle frame and error counters to zero.
Syntax:	char process_return_data(void)
Remarks:	
Return Value:	Void.
See also:	

reset_rx_base

Purpose:	Set a new value for the dmx receive start channel.
Syntax:	char reset_rx_base(unsigned int base)
Remarks:	base in range 0 to 511
Return Value:	YES
See also:	

clr_error_flag

Purpose:	Show dongle communication is operating correctly.
Syntax:	void clr_error_flag(void)
Remarks:	
Return Value:	void.
See also:	set_error_flag, is_error_flag

set_error_flag

Purpose:	Show dongle communication has failed.
Syntax:	void set_error_flag(void)
Remarks:	
Return Value:	void.
See also:	clr_error_flag, is_error_flag

is_error_flag

Purpose:	Retrieve dongle communication status.
Syntax:	char is_error_flag(void)
Remarks:	
Return Value:	YES=good NO=communication fail
See also:	set_error_flag, clr_error_flag

clr_memory

Purpose:	Clear entire mem array.
Syntax:	void clr_memory(void)
Remarks:	
Return Value:	void.
See also:	tx_memory

clr_rx

Purpose:	Reset all data in rx structure.
Syntax:	void clr_rx(void)
Remarks:	
Return Value:	void.
See also:	

set_rx_header

Purpose:	Set new receive dmx header code.
Syntax:	unsigned char set_rx_header(unsigned char rheader)
Remarks:	Updates header structure
Return Value:	header
See also:	

get_rx_header

Purpose:	Get current receive dmx header code.
Syntax:	unsigned char get_rx_header(void)
Remarks:	
Return Value:	header
See also:	

set_tx_header

Purpose:	Set new transmit dmx header code.
Syntax:	unsigned char set_tx_header(unsigned char theader)
Remarks:	Updates header structure
Return Value:	header
See also:	

get_tx_header

Purpose:	Get current transmit dmx header code.
Syntax:	unsigned char get_tx_header(void)
Remarks:	
Return Value:	header
See also:	

set_tx_break_time

Purpose:	Set new transmit break time.
Syntax:	unsigned char set_tx_break_time(unsigned char break_time)
Remarks:	Updates header structure. Data calibrated in microseconds in range 0 - 255. Data must be > 88uS for valid dmx.
Return Value:	break_time
See also:	

get_tx_break_time

Purpose:	Return transmit break time.
Syntax:	unsigned char get_tx_break_time(void)
Remarks:	
Return Value:	break_time
See also:	

set_tx_mab_time

Purpose:	Set new transmit mark after break time.
Syntax:	unsigned char set_tx_mab_time(unsigned char mab_time)
Remarks:	Updates header structure. Data calibrated in microseconds in range 2,6,10,14..254. Must be >8uS for valid dmx.
Return Value:	mab_time
See also:	

get_tx_mab_time

Purpose:	Return transmit mark after break time.
Syntax:	unsigned char get_tx_mab_time(void)
Remarks:	
Return Value:	mab_time
See also:	

set_compress

Purpose:	Select receive dmx compression mode. See rx_dmx_ptr for description.
Syntax:	unsigned char set_compress(unsigned char compress)

Remarks:	compress=YES to enable 2 bit mode.
	compress=NO to enable to 8 bit mode.
	Dongle II does not support compress mode.

Return Value:	dongle_compress
See also:	

get_compress

Purpose:	Return the current state of dongle_compress.
Syntax:	unsigned char get_compress(void)

Return Value:	compress=YES for 2 bit mode.
	compress=NO for 8 bit mode.

See also:	set_compress, rx_dmx_ptr

get_firmware

Purpose:	Return the current firmware of the dongle.
Syntax:	unsigned char get_firmware(void)
Return Value:	Firmware version
Notes:	Three major releases exist. V16, V17 and V20. V17
has	
	variable block size for transmitting and receiving dmx. V20 is the Dongle II.
See also:	

get_dongle_type

Purpose:	Return the type number of the dongle.
Syntax:	unsigned char get_dongle_type(void)
Return Value:	0x05
See also:	

set_tx_aux1

Purpose:	Set new header.aux1 value.
Syntax:	unsigned char set_tx_aux1(unsigned char aux1)
Remarks:	Controls number of bytes transmitted when a command_start_tx is issued. A data value of zero causes 128 bytes to be transmitted. This is to retain compatibility with V16 firmware. All other values from 1 to 255 represent one less than the number of bytes to transmit. For example a value of 3 will transmit 4 bytes.

Return Value: aux1

V A R I A B L E S

GLOBAL The following global data variables are used by the library:

unsigned int address

Specifies the port address for access of the dongle. The library uses a default address of 0x378 which is usual for LPT1. This variable is set by the **init_dongle()** function.

char dongle_fail

Set to YES if communication to the dongle times out. Normal value is NO. Variable is accessed by functions clr_error_flag(), set_error_flag(), get_error_flag().

char dongle_compress

Used to select between the two available methods of receiving DMX512.

dongle_compress=NO is the default which allows 128 bytes of dmx data to be received at 8 bit resolution.
dongle_compress=YES allows 512 bytes of dmx data to be received at 2 bit resolution.
Variable is accessed by functions **set_compress(), get_compress().**

unsigned int delay_time

Defined for future implementation. See **short_delay(), set_delay().**

STRUCTURES

The following data structures are used by the library:

HEADER

```
typedef struct    {
    unsigned char    break_time;  /* DMX Break Time uS */
    unsigned char    mab_time;    /* DMX Mark after Break time uS */
    unsigned char    control_3;   /* Not used */
    unsigned char    rx_header;   /* DMX receive header code */
    unsigned char    rx_base_hi;  /* DMX receive base address */
    unsigned char    rx_base_lo;  /* (range 1 - 512) */
    unsigned char    tx_header;   /* DMX transmit header code */
    unsigned char    aux1;      /* Not used */
    unsigned char    aux2;
    unsigned char    aux3;
    unsigned char    aux4;
    unsigned char    aux5;
    unsigned char    aux6;
    unsigned char    aux7;
    unsigned char    aux8;
    unsigned char    aux9;
    unsigned char    aux10;
    }HEADER;
```

The HEADER structure defines all control variables transmitted to the dongle. All structure items have associated **get..()** and **set..()**functions. These functions should be used in preference to direct structure access.

unsigned char header.break_time

Set the dmx transmit break time in microseconds. Any value in the range 0 to 255 may be used, but values less than 88uS will produce a dmx output which is outside the protocol specification. The library uses a default of 200uS.

unsigned char header.mab_time

Set the dmx protocol delay between the end of break and start bit of the header code. The value is measured in microseconds. Any value from the sequence 2,6,10,14,18,22,26 ... 254 may be used, but a value less than 8uS will produce a dmx output which is outside the protocol specification. The library uses a default of 26uS.

unsigned char header.rx_header

Set the dmx header code used by the dongle when receiving dmx. Any dmx frames with a header code not equal to this value will be ignored by the receive software. The default is 0.

unsigned char header.rx_base_hi/lo

Set the start address of dmx data to be received. The two bytes form a 16 bit word in the range 1 to 512. When set to 1 the dongle will receive data starting with the first channel. The default is 0x0001.

unsigned char header.tx_header

Set the dmx header code used by the dongle when transmitting dmx. The default is 0.

unsigned char header.aux1

Implemented from firmware V17. Defines the number of bytes to be transmitted or received by the dongle. For compatability with earlier firmware, a value of zero represents 128 byte transmission. All other data values represent one less than the block size. For example set aux1 = 255 for a block size of 256 bytes and set aux1=2 for a block size of 3 bytes. This value is ignored when the dongle is receiving in compressed mode.

RX_CTRL

```
typedef struct  {
   unsigned int  channel_count;  /* number of rx chan 0-512 */
   unsigned int  period;         /* dmx cycle time in mS */
   unsigned int  cap1;
   unsigned int  cap2;
   unsigned int  brk;         /* dmx break period in uS */
   unsigned int  mab;         /* mark after break in uS */
   unsigned int  frequency    ;    /* dmx update rate in Hz */
   unsigned int  break_count; /* dmx frame counter */
   unsigned int  ovr_count;    /* overrun counter */
   unsigned int  hdr_count;    /* bad header counter */
   unsigned int  frm_count;    /* framing error counter */
   }RX_CTRL;
```

The rx_ctrl structure contains all control and timing data returned by the dongle. It is a read only array and may be accessed directly by applications. The structure is updated by each call of function **tx_control_block().**

unsigned int rx_ctrl.channel_count

Contains a value in the range 0 to 512 which represents the number of channels received during the last dmx frame. The dmx protocol allows any number of channels up to 512 to be transmitted.

unsigned int rx_ctrl.period

Represents the repeat time of the last two dmx frames received. Calibrated in milliseconds.

unsigned int rx_ctrl.frequency

Represents the update rate of the received dmx signal. Calculated as the reciprocal of the period. Calibrated in Hz.

unsigned int rx_ctrl.brk

The break active time measured on the last dmx frame received. Calibrated in microseconds.

unsigned int rx_ctrl.mab

The time delay between the end of break and the start bit of the header code measured on the last dmx frame received. Calibrated in microseconds.

unsigned int rx_ctrl.break_count

A 16 bit count of the number of dmx frames received since the function **rx_reset_counters()** was last called.

unsigned int rx_ctrl.ovr_count

A 16 bit count of the number of serial overrun errors detected since the function **rx_reset_counters()** was last called.

unsigned int rx_ctrl.hdr_count

A 16 bit count of the number of non matching header codes received since the function **rx_reset_counters()** was last called.

unsigned int rx_ctrl.frm_count

A 16 bit count of the number of framing errors detected since the function **rx_reset_counters()** was last called. Framing errors exclude line breaks.

RX

```
typedef struct  {
        unsigned char    cur_level[MAX_CHAN];
        unsigned char    min_level[MAX_CHAN];
        unsigned char    max_level[MAX_CHAN];
        }RX;
```

The rx structure contains all channel data received by the dongle. The structure is updated by the function call **rx_dmx().**

unsigned char rx.cur_level[]

An array of 512 entries showing the current channel levels at the last **rx_dmx()** call.

unsigned char rx.min_level[]

An array of 512 entries showing the minimum level received.

unsigned char rx.max_level[]

An array of 512 entries showing the maximum level received.

MEM

```
#define     MAX_MEM       50
#define     LEGEND_LENGTH  35
#define     MAX_CHAN      512

typedef struct  {
        unsigned char    level[MAX_CHAN];
        char   legend[LEGEND_LENGTH+1];
        }MEM;
MEM mem[MAX_MEM];
```

The mem array is used to build 50 complete dmx memories complete with alphanumeric legends. The tx_memory() function is used to transmit any one of the memories to the dmx output. The mem array may be accessed directly to set data levels.

CONVERSION TABLES

Dec	Hex	Binary	Dec	Hex	Binary	Dec	Hex	Binary
0	00	0000 0000	32	20	0010 0000	64	40	0100 0000
1	01	0000 0001	33	21	0010 0001	65	41	0100 0001
2	02	0000 0010	34	22	0010 0010	66	42	0100 0010
3	03	0000 0011	35	23	0010 0011	67	43	0100 0011
4	04	0000 0100	36	24	0010 0100	68	44	0100 0100
5	05	0000 0101	37	25	0010 0101	69	45	0100 0101
6	06	0000 0110	38	26	0010 0110	70	46	0100 0110
7	07	0000 0111	39	27	0010 0111	71	47	0100 0111
8	08	0000 1000	40	28	0010 1000	72	48	0100 1000
9	09	0000 1001	41	29	0010 1001	73	49	0100 1001
10	0A	0000 1010	42	2A	0010 1010	74	4A	0100 1010
11	0B	0000 1011	43	2B	0010 1011	75	4B	0100 1011
12	0C	0000 1100	44	2C	0010 1100	76	4C	0100 1100
13	0D	0000 1101	45	2D	0010 1101	77	4D	0100 1101
14	0E	0000 1110	46	2E	0010 1110	78	4E	0100 1110
15	0F	0000 1111	47	2F	0010 1111	79	4F	0100 1111
16	10	0001 0000	48	30	0011 0000	80	50	0101 0000
17	11	0001 0001	49	31	0011 0001	81	51	0101 0001
18	12	0001 0010	50	32	0011 0010	82	52	0101 0010
19	13	0001 0011	51	33	0011 0011	83	53	0101 0011
20	14	0001 0100	52	34	0011 0100	84	54	0101 0100
21	15	0001 0101	53	35	0011 0101	85	55	0101 0101
22	16	0001 0110	54	36	0011 0110	86	56	0101 0110
23	17	0001 0111	55	37	0011 0111	87	57	0101 0111
24	18	0001 1000	56	38	0011 1000	88	58	0101 1000
25	19	0001 1001	57	39	0011 1001	89	59	0101 1001
26	1A	0001 1010	58	3A	0011 1010	90	5A	0101 1010
27	1B	0001 1011	59	3B	0011 1011	91	5B	0101 1011
28	1C	0001 1100	60	3C	0011 1100	92	5C	0101 1100
29	1D	0001 1101	61	3D	0011 1101	93	5D	0101 1101
30	1E	0001 1110	62	3E	0011 1110	94	5E	0101 1110
31	1F	0001 1111	63	3F	0011 1111	95	5F	0101 1111

Dec	Hex	Binary	Dec	Hex	Binary	Dec	Hex	Binary
96	60	0110 0000	128	80	1000 0000	160	A0	1010 0000
97	61	0110 0001	129	81	1000 0001	161	A1	1010 0001
98	62	0110 0010	130	82	1000 0010	162	A2	1010 0010
99	63	0110 0011	131	83	1000 0011	163	A3	1010 0011
100	64	0110 0100	132	84	1000 0100	164	A4	1010 0100
101	65	0110 0101	133	85	1000 0101	165	A5	1010 0101
102	66	0110 0110	134	86	1000 0110	166	A6	1010 0110
103	67	0110 0111	135	87	1000 0111	167	A7	1010 0111
104	68	0110 1000	136	88	1000 1000	168	A8	1010 1000
105	69	0110 1001	137	89	1000 1001	169	A9	1010 1001
106	6A	0110 1010	138	8A	1000 1010	170	AA	1010 1010
107	6B	0110 1011	139	8B	1000 1011	171	AB	1010 1011
108	6C	0110 1100	140	8C	1000 1100	172	AC	1010 1100
109	6D	0110 1101	141	8D	1000 1101	173	AD	1010 1101
110	6E	0110 1110	142	8E	1000 1110	174	AE	1010 1110
111	6F	0110 1111	143	8F	1000 1111	175	AF	1010 1111
112	70	0111 0000	144	90	1001 0000	176	B0	1011 0000
113	71	0111 0001	145	91	1001 0001	177	B1	1011 0001
114	72	0111 0010	146	92	1001 0010	178	B2	1011 0010
115	73	0111 0011	147	93	1001 0011	179	B3	1011 0011
116	74	0111 0100	148	94	1001 0100	180	B4	1011 0100
117	75	0111 0101	149	95	1001 0101	181	B5	1011 0101
118	76	0111 0110	150	96	1001 0110	182	B6	1011 0110
118	77	0111 0111	151	97	1001 0111	183	B7	1011 0111
119	78	0111 1000	152	98	1001 1000	184	B8	1011 1000
120	79	0111 1001	153	99	1001 1001	185	B9	1011 1001
121	7A	0111 1010	154	9A	1001 1010	186	BA	1011 1010
122	7B	0111 1011	155	9B	1001 1011	187	BB	1011 1011
123	7C	0111 1100	156	9C	1001 1100	188	BC	1011 1100
124	7D	0111 1101	157	9D	1001 1101	189	BD	1011 1101
125	7E	0111 1110	158	9E	1001 1110	190	BE	1011 1110
126	7F	0111 1111	159	9F	1001 1111	191	BF	1011 1111

Dec	Hex	Binary	Dec	Hex	Binary
192	C0	1100 0000	224	E0	1110 0000
193	C1	1100 0001	225	E1	1110 0001
194	C2	1100 0010	226	E2	1110 0010
195	C3	1100 0011	227	E3	1110 0011
196	C4	1100 0100	228	E4	1110 0100
197	C5	1100 0101	229	E5	1110 0101
198	C6	1100 0110	230	E6	1110 0110
199	C7	1100 0111	231	E7	1110 0111
200	C8	1100 1000	232	E8	1110 1000
201	C9	1100 1001	233	E9	1110 1001
202	CA	1100 1010	234	EA	1110 1010
203	CB	1100 1011	235	EB	1110 1011
204	CC	1100 1100	236	EC	1110 1100
205	CD	1100 1101	237	ED	1110 1101
206	CE	1100 1110	238	EE	1110 1110
207	CF	1100 1111	239	EF	1110 1111
208	D0	1101 0000	240	F0	1111 0000
209	D1	1101 0001	241	F1	1111 0001
210	D2	1101 0010	242	F2	1111 0010
211	D3	1101 0011	243	F3	1111 0011
212	D4	1101 0100	244	F4	1111 0100
213	D5	1101 0101	245	F5	1111 0101
214	D6	1101 0110	246	F6	1111 0110
215	D7	1101 0111	247	F7	1111 0111
216	D8	1101 1000	248	F8	1111 1000
217	D9	1101 1001	249	F9	1111 1001
218	DA	1101 1010	250	FA	1111 1010
219	DB	1101 1011	251	FB	1111 1011
220	DC	1101 1100	252	FC	1111 1100
221	DD	1101 1101	253	FD	1111 1101
222	DE	1101 1110	254	FE	1111 1110
223	DF	1101 1111	255	FF	1111 1111

INDEX

Artistic Licence
© Artistic Licence (UK) Ltd. 1994-2004
B1 & B3 Livingstone Court
Peel Road
Harrow
Middlesex
England
HA3 7QT
Tel: +44 (0)20 88 63 45 15
Fax: +44 (0)20 84 26 05 51
Email: Sales@ArtisticLicence.com

PROFESSIONAL LIGHTING AND SOUND ASSOCIATION