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

Artistic Licence Engineering Ltd
Manual Revision V4.7. (DLL Revision 32)
CONTENTS

INTRODUCTION ........................................................................................................................................... 9
Quick Start .................................................................................................................................................. 9
DMX-DONGLE II ...................................................................................................................................... 9
ART-NET ...................................................................................................................................................... 10
WINDOWS DEVELOPMENT .................................................................................................................. 11
Overview .................................................................................................................................................. 11
FILES ....................................................................................................................................................... 11
CODE INTERFACE .................................................................................................................................. 12
MEMORY STRUCTURE .......................................................................................................................... 13
Pipe Library .............................................................................................................................................. 13
Node Library ............................................................................................................................................. 14
Transmit Strategy .................................................................................................................................... 15
FUNCTION PROTOTYPES .................................................................................................................... 16
Control Functions .................................................................................................................................... 16
ArtNetInit ................................................................................................................................................ 16
ArtNetEnd ............................................................................................................................................... 17
ArtNetReadWrite ................................................................................................................................... 18
ArtNetGetDllRevision .......................................................................................................................... 19
Generalised Data Functions .................................................................................................................. 20
ArtNetSetTxData ..................................................................................................................................... 20
ArtNetSetTxDataGroup .......................................................................................................................... 20
ArtNetGetRxData ................................................................................................................................... 21
ArtNetClearDataBuffer .......................................................................................................................... 21
DMX-DONGLE Control Functions ......................................................................................................... 22
DongleSetTransferCount ....................................................................................................................... 22
DongleGetTransferCount ....................................................................................................................... 22
DongleGetFirmwareRevision ............................................................................................................... 22
DongleGetPortAddress .......................................................................................................................... 22
DongleGetType ......................................................................................................................................... 23
DongleSetRxStartCode ........................................................................................................................... 23
DongleSetTxStartCode ........................................................................................................................... 23
DongleSetTxBreakTime .......................................................................................................................... 23
DongleSetTxMabTime ............................................................................................................................. 24
DongleGetBreakTime .............................................................................................................................. 24
DongleGetMabTime .................................................................................................................................. 24
DongleGetPeriod ...................................................................................................................................... 24
DongleGetFrequency ............................................................................................................................. 25
DongleGetChannelCount ......................................................................................................................... 25
DongleSetLoopOn .................................................................................................................................... 25
DongleSetMergeOn ................................................................................................................................... 25
DongleSetLoopOff .................................................................................................................................... 26
DongleSetAux ............................................................................................................................................ 26

Artistic Licence Engineering Ltd.  DMX-Dongle II & Art-Net Driver SDK  4 / 82
ArtNet Command Functions ........................................................................................................... 29
ArtNetSendUdp .................................................................................................................................. 30
  Code Example: ................................................................................................................................. 30
ArtNetSetCall BackUdp ReceivePre ............................................................................................... 31
  Code Example: ................................................................................................................................. 31
ArtNetSetCall BackUdp ReceivePost ............................................................................................... 32
ArtNetSetCall BackUdp Transmit ..................................................................................................... 32
ArtNetCancelCallback ....................................................................................................................... 32
PipeLibrary Receive Functions ........................................................................................................... 33
ArtNetGetRxPipeRequestUniverse ................................................................................................. 33
ArtNetGetRxPipeRequestUniverse ................................................................................................. 33
ArtNetGetRxPipeUniverse ................................................................................................................. 34
ArtNetGetRxPipeIndexForIp ............................................................................................................. 34
ArtNetGetRxPipeIndexForUniverse ................................................................................................. 34
ArtNetIsRxPipe Active ......................................................................................................................... 34
ArtNetIsPipe0 Dongle .......................................................................................................................... 34
ArtNetAnyArtNetReceived .................................................................................................................. 34
ArtNetGetNode IndexForRxPipe ......................................................................................................... 35
ArtNetEnumerate ................................................................................................................................. 35
ArtNetSoftEnumerate ........................................................................................................................... 35
ArtNetGetRxPipeIpAddress .................................................................................................................. 35
ArtNetGetRxPipeEstatMan .................................................................................................................. 36
ArtNetGetRxPipeEstatManString ........................................................................................................ 36
ArtNetGetRxPipeOemH ......................................................................................................................... 36
ArtNetGetRxPipeOem ........................................................................................................................... 36
ArtNetGetRxPipeOemString .................................................................................................................. 36
ArtNetGetRxPipeVersionInfoH ............................................................................................................. 37
ArtNetGetRxPipeVersionInfo .............................................................................................................. 37
ArtNetGetRxPipeNumberPorts ............................................................................................................ 37
ArtNetGetRxPipeRxReceived ............................................................................................................... 37
ArtNetGetRxPipeRxErrors .................................................................................................................. 37
ArtNetGetRxPipeRxTest ....................................................................................................................... 38
ArtNetGetRxPipeRxSip ........................................................................................................................... 38
ArtNetGetRxPipeRxText ....................................................................................................................... 38
ArtNetGetRxPipeRxDisable .................................................................................................................. 38
ArtNetGetRxPipeStatus Indicators .................................................................................................... 38
ArtNetGetRxPipeStatusUbea Active ................................................................................................. 39
ArtNetGetRxPipeUbeaVersion ............................................................................................................ 39
ArtNetGetRxPipeStatusRdm Capable ................................................................................................. 40
ArtNetGetRxPipeStatusRomBoot ........................................................................................................ 40
ArtNetGetRxPipeShortName ............................................................................................................... 40
ArtNetGetRxPipeLongName ............................................................................................................... 40
ArtNetGetRxPipeNodeReport ............................................................................................................. 40
ArtNetGetRxPipeMacroKeys ........................................................................... 41
ArtNetGetRxPipeRemoteKeys ........................................................................... 41
ArtNetGetRxPipeVideo ..................................................................................... 41
PipeLibrary Transmit Functions ........................................................................ 42
ArtNetSetTxPipeUniverse .................................................................................. 42
ArtNetCancelTxPipe ........................................................................................... 42
ArtNetGetTxPipeUniverse .................................................................................. 42
ArtNetIsTxPipeActive ......................................................................................... 42
ArtNetGetNodeIndexForTxPipe .......................................................................... 43
ArtNetGetTxPipeIpAddress ................................................................................ 43
ArtNetGetTxPipeEstaMan ................................................................................... 43
ArtNetGetTxPipeEstaManString ......................................................................... 43
ArtNetGetTxPipeOemH ..................................................................................... 43
ArtNetGetTxPipeOem ....................................................................................... 44
ArtNetGetTxPipeOemString ............................................................................... 44
ArtNetGetTxPipeVersionInfoH .......................................................................... 44
ArtNetGetTxPipeVersionInfo ............................................................................. 44
ArtNetGetTxPipeNumberPorts ......................................................................... 44
ArtNetGetTxPipeStatusUbeaActive .................................................................. 45
ArtNetGetTxPipeUbeaVersion .......................................................................... 45
ArtNetGetTxPipeStatusRdmCapable .................................................................. 45
ArtNetGetTxPipeStatusRomBoot ...................................................................... 45
ArtNetGetTxPipeShortName ............................................................................. 45
ArtNetGetTxPipeLongName ............................................................................... 46
ArtNetGetTxPipeNodeReport ............................................................................ 46
ArtNetGetTxPipeMacroKeys ............................................................................. 46
ArtNetGetTxPipeRemoteKeys ............................................................................ 46
ArtNetGetTxPipeVideo .................................................................................... 46
NodeLibrary Functions ....................................................................................... 47
ArtNetGetNode .................................................................................................. 47
IpAddress ........................................................................................................... 47
ArtNetGetNodeTxUniverse ................................................................................. 47
ArtNetGetNodeIndexForIp .................................................................................. 47
ArtNetGetNodeTxImplemented ......................................................................... 47
ArtNetGetNodeRxUniverse ............................................................................... 48
ArtNetGetNodeRxImplemented ......................................................................... 48
ArtNetGetNode .................................................................................................. 48
EstaMan ............................................................................................................ 48
ArtNetGetNode .................................................................................................. 48
EstaManString ................................................................................................... 48
ArtNetGetNodeOemH ....................................................................................... 48
ArtNetGetNodeOem .......................................................................................... 49
ArtNetGetNodeOemString ................................................................................ 49
ArtNetGetNodeVersionInfoH ............................................................................ 49
ArtNetGetNodeVersionInfo ............................................................................... 49
ArtNetGetTxPipeNumberPorts ......................................................................... 49
ArtNetGetNodeStatusIndicators ...................................................................... 50
ArtNetGetNodeStatusUbeaActive ................................................................... 50
ArtNetGetNodeUbeaVersion ............................................................................. 50
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.

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 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.


Artistic Licence products supporting Art-Net include:

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

```c
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:

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

### ArtNetInit

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Start up driver operation and default broadcast IP address.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>BYTE ArtNetInit(char* NewIp)</td>
</tr>
<tr>
<td>Remarks:</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Return Value:

<table>
<thead>
<tr>
<th>Value</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>No devices found.</td>
</tr>
<tr>
<td>0x0045</td>
<td>Devices detected, but power on test failed.</td>
</tr>
<tr>
<td>0x0005</td>
<td>DMX-Dongle detected.</td>
</tr>
<tr>
<td>0x0105</td>
<td>Art-Net devices detected on IP address 2.x.x.x</td>
</tr>
<tr>
<td>0x0205</td>
<td>Art-Net devices detected on IP address 10.x.x.x</td>
</tr>
<tr>
<td>0xffff</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

See also: ArtNetEnd(); DongleGetPortAddress()

### ArtNetEnd

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Close drivers and free memory prior to application exit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td><code>void ArtNetEnd (void):</code></td>
</tr>
<tr>
<td>Remarks:</td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>Void.</td>
</tr>
<tr>
<td>See also:</td>
<td>ArtNetInit(char* NewIp)</td>
</tr>
</tbody>
</table>
### ArtNetReadWrite

**Purpose:** Send local data buffers to and from hardware.

**Syntax:**
```cpp
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.

<table>
<thead>
<tr>
<th>Pipe</th>
<th>Processes channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0-511</td>
</tr>
<tr>
<td>1</td>
<td>512-1023</td>
</tr>
<tr>
<td>2</td>
<td>1024-1535</td>
</tr>
<tr>
<td>3</td>
<td>1536-2047</td>
</tr>
<tr>
<td>19</td>
<td>9729-10240</td>
</tr>
<tr>
<td>-1</td>
<td>Process all channels</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Receive Data</th>
<th>Transmit Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMX-Dongle II</td>
<td>Copied from DMX-Dongle to local buffer</td>
<td>Copied from local buffer to DMX-Dongle II</td>
</tr>
</tbody>
</table>

**Return Value:** True on success, False on any error.

**See also:** DongleSetTransferCount()
<table>
<thead>
<tr>
<th>ArtNetGetDllRevision</th>
<th>Purpose:</th>
<th>Return the revision number of the DLL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>BYTE ArtNetGetDllRevision(void):</td>
<td></td>
</tr>
<tr>
<td>Remarks:</td>
<td>It is useful for applications using the driver to display this number in the application about box.</td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>Revision number.</td>
<td></td>
</tr>
<tr>
<td>See also:</td>
<td>DongleGetFirmwareRevision();</td>
<td></td>
</tr>
</tbody>
</table>
## 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArtNetSetTxData</strong></td>
<td>Set channel level in PipeLibrary[Pipe].TxDmx.Data</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td><code>void ArtNetSetTxData(WORD Channel, BYTE Data)</code></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
<td>void</td>
</tr>
<tr>
<td><strong>See also:</strong></td>
<td>ArtNetSetTxDataGroup()</td>
</tr>
</tbody>
</table>

<p>| <strong>ArtNetSetTxDataGroup</strong> | Set range of channel levels in PipeLibrary[Pipe].TxDmx.Data                   |
| <strong>Syntax:</strong>              | <code>void ArtNetSetTxDataGroup(WORD StartCh, WORD EndCh, BYTE Data)</code>            |
| <strong>Remarks:</strong>             | 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 |
| <strong>Return Value:</strong>        | Void                                                                        |
| <strong>See also:</strong>            | ArtNetSetTxData()                                                            |</p>
<table>
<thead>
<tr>
<th><strong>ArtNetGetRxData</strong></th>
<th><strong>Purpose:</strong></th>
<th>Returns channel level from PipeLibrary[Pipe].RxDmx.Data[].</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>BYTE ArtNetGetRxData(WORD Channel)</td>
<td></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
<td>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</td>
<td></td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
<td>void</td>
<td></td>
</tr>
<tr>
<td><strong>See also:</strong></td>
<td>ArtNetSetTxData()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ArtNetClearDataBuffer</strong></th>
<th><strong>Purpose:</strong></th>
<th>Set all PipeLibrary receive and transmit channels to a value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>void ArtNetClearDataBuffer(BYTE Data)</td>
<td></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
<td>All transmit and receive data in all pipe entries is set to a value in the range 0 to 255</td>
<td></td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
<td>void</td>
<td></td>
</tr>
<tr>
<td><strong>See also:</strong></td>
<td>ArtNetSetTxDataGroup()</td>
<td></td>
</tr>
</tbody>
</table>
The DMX512 control functions are specific to the DMX-Dongle. They have no affect on pipes set to Art-Net operation.

### DMX-Dongle Control Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>DongleSetTransferCount</td>
<td>Set the number of channels to transfer in the ArtNetReadWrite() function.</td>
<td>void DongleSetTransferCount (WORD Number):</td>
<td>Number is in the range 24 -&gt; 512. This function allows the application to significantly improve the execution time of ArtNetReadWrite when less than 512 channels are required.</td>
<td></td>
<td>ArtNetInit()</td>
</tr>
<tr>
<td>DongleGetTransferCount</td>
<td>Get the number of channels transferred in the ArtNetReadWrite() function.</td>
<td>WORD DongleGetTransferCount (void):</td>
<td></td>
<td>Number is in the range 24 -&gt; 512</td>
<td>DongleSetTransferCount ()</td>
</tr>
<tr>
<td>DongleGetFirmwareRevision</td>
<td>Return the revision number of the DMX-Dongle.</td>
<td>BYTE DongleGetFirmwareRevision (void):</td>
<td>It is useful for applications using the driver to display this number in the application about box.</td>
<td>Firmware Revision number.</td>
<td>DongleGetDllRevision()</td>
</tr>
<tr>
<td>DongleGetPortAddress</td>
<td>Return the LPT number at which the DMX-Dongle was detected.</td>
<td>char DongleGetPortAddress (void):</td>
<td>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.</td>
<td></td>
<td>ArtNetInit()</td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td>Return Value</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td><strong>DongleGetType</strong></td>
<td>Return the Type number of the DMX-Dongle.</td>
<td><code>BYTE DongleGetType(void);</code></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DongleSetRxStartCode</strong></td>
<td>Set the start code to be used for receiving DMX512.</td>
<td><code>void DongleSetRxStartCode(BYTE StartCode)</code></td>
<td>This function is only applicable to the DMX-Dongle. All standard lighting data uses a start code of zero.</td>
<td>void</td>
<td></td>
</tr>
<tr>
<td><strong>DongleSetTxStartCode</strong></td>
<td>Set the start code to be used for transmitting DMX512.</td>
<td><code>void DongleSetTxStartCode(BYTE StartCode)</code></td>
<td>This function is only applicable to the DMX-Dongle. All standard lighting data uses a start code of zero.</td>
<td>void</td>
<td></td>
</tr>
<tr>
<td><strong>DongleSetTxBreakTime</strong></td>
<td>Set the DMX-Dongle transmit break time.</td>
<td><code>void DongleSetTxBreakTime(BYTE Time)</code></td>
<td>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.</td>
<td>void</td>
<td></td>
</tr>
</tbody>
</table>

See also: `ArtNetInit();`
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DongleSetTxMabTime</strong></td>
<td>Set the DMX-Dongle transmit MaB time.</td>
<td><code>void DongleSetTxMabTime(BYTE Time)</code></td>
<td>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.</td>
<td>void</td>
</tr>
<tr>
<td><strong>DongleGetBreakTime</strong></td>
<td>Retrieve the received DMX512 break time from the DMX-Dongle.</td>
<td><code>WORD DongleGetBreakTime(void)</code></td>
<td>This function is only applicable to the DMX-Dongle.</td>
<td>The return value is the received DMX512 break time calibrated in uS.</td>
</tr>
<tr>
<td><strong>DongleGetMabTime</strong></td>
<td>Retrieve the received DMX512 MaB time from the DMX-Dongle.</td>
<td><code>WORD DongleGetMabTime(void)</code></td>
<td>This function is only applicable to the DMX-Dongle.</td>
<td>The return value is the received DMX512 MaB time calibrated in uS.</td>
</tr>
<tr>
<td><strong>DongleGetPeriod</strong></td>
<td>Retrieve the received DMX512 cycle (or frame) time from the DMX-Dongle.</td>
<td><code>WORD DongleGetPeriod(void)</code></td>
<td>This function is only applicable to the DMX-Dongle.</td>
<td>The return value is the received DMX512 cycle time calibrated in uS. The cycle time is the time between two consecutive start of breaks</td>
</tr>
</tbody>
</table>

See also:

**DongleGetFrequency();**
### DongleGetFrequency

**Purpose:** Retrieve the frequency of DMX512 frames received from the DMX-Dongle.

**Syntax:**

```c
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()`

### DongleGetChannelCount

**Purpose:** Retrieve the number of channels received by the DMX-Dongle during the last frame of DMX512.

**Syntax:**

```c
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:**

```c
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()`

### DongleSetMergeOn

**Purpose:** Set the DMX-Dongle to operate in merge mode.

**Syntax:**

```c
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:**
```c
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:**
```c
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 do not relate to any pipes set to Art-Net operation.

#### DongleResetRxCounters

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Reset the DMX-Dongle frame and error counters to zero.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void DongleResetRxCounters(void)</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function is only applicable to the DMX-Dongle.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>void</td>
</tr>
</tbody>
</table>

See also: ResetRxCounters()

#### DongleGetPacketCount

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Return the number of DMX512 frames received by the DMX-Dongle since the last ResetRxCounters().</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>WORD DongleGetPacketCount(void)</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function is only applicable to the DMX-Dongle.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Frame count.</td>
</tr>
</tbody>
</table>

See also: ResetRxCounters()

#### DongleGetFramingErrorCount

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Return the number of DMX512 framing errors received by the DMX-Dongle since the last ResetRxCounters().</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>WORD DongleGetFramingErrorCount(void)</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function is only applicable to the DMX-Dongle.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Framing error count.</td>
</tr>
</tbody>
</table>

See also: ResetRxCounters()

#### DongleGetStartCodeErrorCount

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>WORD DongleGetStartCodeErrorCount (void)</td>
</tr>
<tr>
<td>Remarks:</td>
<td>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.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Non-matching start code count.</td>
</tr>
</tbody>
</table>

See also: ResetRxCounters(); DongleSetRxStartCode();
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>DongleGetOverrunErrorCount</td>
<td>Return the number of DMX512 overrun errors received by the DMX-Dongle since the last ResetRxCounters().</td>
<td>WORD DongleGetOverrunErrorCount(void)</td>
<td>This function is only applicable to the DMX-Dongle.</td>
<td>void</td>
<td>ResetRxCounters();</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Send an Art-Net message to the network.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void ArtNetSendUdp(BYTE* Ptr, WORD Size)</td>
</tr>
<tr>
<td>Remarks:</td>
<td>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.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>void</td>
</tr>
<tr>
<td>See also:</td>
<td>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.</td>
</tr>
</tbody>
</table>

```c
#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[3]=7;
ArtAddress.SwSub=0;

ArtNetSendUdp( &((char)ArtAddress), sizeof(T_ArtAddress));
```
## ArtNetSetCallBackUdpReceivePre

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

```c
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
   }
}
| **ArtNetSetCallBackBackUdpReceivePost** | **Purpose:** | Assigns a user function that will be called after the driver software when Art-Net data is received. |
|                                         | **Syntax:** | `void ArtNetSetCallBackBackUdpReceivePost (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. Pfr 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:** | |

| **ArtNetSetCallBackBackUdpTransmit** | **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 ArtNetSetCallBackBackUdpTransmit (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. Pfr 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetSetRxPipeRequestUniverse</td>
<td>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 -&gt; 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.</td>
<td>void ArtNetSetRxPipeRequestUniverse(int Pipe, BYTE Universe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1 Universe is in the range 0x00 -&gt; 0xff. The high nibble is the 'Sub-Net'.</td>
<td>Void</td>
<td>ArtNetReadWrite(); ArtNetEnumerate()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeRequestUniverse</td>
<td>Return the RequestUniverse entry in the PipeEntry structure indexed by Pipe.</td>
<td>BYTE ArtNetGetRxPipeRequestUniverse(int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1 RequestUniverse is in the range 0x00 -&gt; 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.</td>
<td>RequestUniverse</td>
<td>ArtNetSetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td>Return Value</td>
<td>See also</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>ArtNetGetRxPipeUniverse</td>
<td>Return the Universe currently received by the PipeEntry structure indexed by Pipe.</td>
<td>\texttt{BYTE ArtNetGetRxPipeUniverse( int Pipe) }</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>Universe is in the range 0x00 -&gt; 0xff. The high nibble relates to the 'Sub-Net' control on Artistic Licence products.</td>
<td>ArtNetGetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeIndexForIp</td>
<td>Return the pipe index of the PipeEntry for which the receive IP address matches.</td>
<td>\texttt{void ArtNetGetRxPipeIndexForIp( char* IpAddress )}</td>
<td>IpAddress is a character string in the format <a href="http://www.xxx.yyy.zzzz">www.xxx.yyy.zzzz</a>.</td>
<td>MaxPipe is returned if a match is not found</td>
<td>ArtNetGetNodeIndexForIp()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeIndexForUniverse</td>
<td>Return the pipe index of the PipeEntry for which the receive universe matches.</td>
<td>\texttt{void ArtNetGetRxPipeIndexForUniverse( WORD Universe )}</td>
<td></td>
<td>MaxPipe is returned if a match is not found</td>
<td>ArtNetGetRxPipeIndexForIp()</td>
</tr>
<tr>
<td>ArtNetIsRxPipeActive</td>
<td>Return the active status of this receive pipe.</td>
<td>\texttt{bool ArtNetIsRxPipeActive( int Pipe) }</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if the RequestUniverse and the actual received universe match.</td>
<td>ArtNetSetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>ArtNetIsPipe0Dongle</td>
<td>Returns the connection status of the DMX-Dongle.</td>
<td>\texttt{bool ArtNetIsPipe0Dongle(void) }</td>
<td></td>
<td>True if the zeroth PipeLibrary is connected to the DMX-Dongle. False if in use by Art-Net devices.</td>
<td>ArtNetSetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>ArtNetAnyArtNetReceived</td>
<td>Returns whether any ArtDmx packets have been received by any pipes.</td>
<td>\texttt{bool ArtNetAnyArtNetReceived(void) }</td>
<td></td>
<td>True if any Pipe has received ArtDmx packets.</td>
<td>ArtNetIsRxPipeActive()</td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td>Return Value</td>
<td>See also</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>ArtNetGetNodeIndexForRxPipe</td>
<td>Cross references an entry in the receive part of PipeLibrary to the NodeLibrary.</td>
<td>int ArtNetGetNodeIndexForRxPipe(void)</td>
<td>Returns a NodeLibrary Index value if the IP addresses can be matched. Otherwise returns MaxNodes.</td>
<td></td>
<td>ArtNetGetPipeIndexForUniverse</td>
</tr>
<tr>
<td>ArtNetEnumerate</td>
<td>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.</td>
<td>void ArtNetEnumerate( void)</td>
<td>This function is called as part of ArtNetInit(). It can be considered as a 'hard reset' of the network.</td>
<td>Void</td>
<td>ArtNetSoftEnumerate()</td>
</tr>
<tr>
<td>ArtNetSoftEnumerate</td>
<td>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.</td>
<td>void ArtNetSoftEnumerate( void)</td>
<td>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.</td>
<td>Void</td>
<td>ArtNetEnumerate()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeIpAddress</td>
<td>Return the IP address of the node from which data was last received.</td>
<td>BYTE* ArtNetGetRxPipeIpAddress( int Pipe)</td>
<td>Pipe is in the range 0 → MaxPipe-1. 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 &quot;0000&quot;</td>
<td></td>
<td>ArtNetSetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td>Return Value</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeEstMan</td>
<td>Return the EstMan field of the node from which data was last received.</td>
<td><code>WORD ArtNetGetRxPipeEstMan(int Pipe)</code></td>
<td>This field contains the ESTA manufacturer ID of the node from which data was last received. The default value is 0x0000.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeEstManString</td>
<td>Returns a string detailing the name of the Node manufacturer.</td>
<td><code>char* ArtNetGetRxPipeEstManString(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>Variable length string containing the manufacturer name. If the name is unknown, the return string indicates the hexadecimal value of this field.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeOemH</td>
<td>Return the Art-Net OemH field of the node from which data was last received.</td>
<td><code>BYTE ArtNetGetRxPipeOemH(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeOem</td>
<td>Return the Art-Net Oem field of the node from which data was last received.</td>
<td><code>BYTE ArtNetGetRxPipeOem(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeOemString</td>
<td>Returns a string detailing the product name and type of node from which data was last received.</td>
<td><code>char* ArtNetGetRxPipeOemString(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>Variable length string containing the product name and type. If the name is unknown, the return string indicates the hexadecimal value of this field.</td>
<td></td>
</tr>
</tbody>
</table>

See also: ArtNetGetNodeEstMan()
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetGetRxPipeVersionInfoH</td>
<td>Return the firmware revision of the node from which data was last received.</td>
<td>BYTE ArtNetGetRxPipeVersionInfoH( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>This field contains the high byte of the firmware revision of the node from which data was last received. The default value is 0x00.</td>
<td>ArtNetSetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeVersionInfo</td>
<td>Return the firmware revision of the node from which data was last received.</td>
<td>BYTE ArtNetGetRxPipeVersionInfo( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>This field contains the low byte of the firmware revision of the node from which data was last received. The default value is 0x00.</td>
<td>ArtNetSetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeNumberPorts</td>
<td>Return the number of receive DMX512 ports supported by the node from which data was last received.</td>
<td>BYTE ArtNetGetRxPipeNumberPorts( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>The maximum number of ports supported by an Art-Net node is 4. The return value is in the range 1 -&gt; 4. The default value is 0x00. NB. A node with 4 ports would have 4 associated pipes.</td>
<td>ArtNetSetRxPipeRequestUniverse()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeRxReceived</td>
<td>Return the receive status of the node from which data was last received.</td>
<td>bool ArtNetGetRxPipeRxReceived( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if data is being received.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeRxErrors</td>
<td>Return the receive errors status of the node from which data was last received.</td>
<td>bool ArtNetGetRxPipeRxErrors( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if errors detected.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td>Return Value</td>
<td>See also</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>ArtNetGetRxPipeRxTest</td>
<td>Return the receive test packet status of the node from which data was last received.</td>
<td><code>bool ArtNetGetRxPipeRxTest( int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if test packets are being received.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeRxSip</td>
<td>Return the system information packet status of the node from which data was last received.</td>
<td><code>bool ArtNetGetRxPipeRxSip( int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if system information packets are being received.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeRxText</td>
<td>Return the receive text packet status of the node from which data was last received.</td>
<td><code>bool ArtNetGetRxPipeRxText( int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if text packets are being received.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeRxDisable</td>
<td>Return the receive enable status of the node from which data was last received.</td>
<td><code>bool ArtNetGetRxPipeRxTest( int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if this receiver has been disabled.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeStatusIndicators</td>
<td>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.</td>
<td><code>unsigned char ArtNetGetRxPipeStatusIndicators( int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 Unknown.</td>
<td>1 Locate.</td>
<td>2 Mute.</td>
<td>3 Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ArtNetGetRxPipeStatusAuthority

**Purpose:** Returns the Programming Authority for the subnet 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:**
```c
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.

### ArtNetGetRxPipeStatusUbeaActive

**Purpose:** Returns the User Bios Extension Area (UBEA) status of the node from which data was last received.

**Syntax:**
```c
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.

### ArtNetGetRxPipeUbeaVersion

**Purpose:** Returns the firmware version number of the User Bios Extension Area (UBEA) of the node from which data was last received.

**Syntax:**
```c
BYTE ArtNetGetRxPipeUbeaVersion( int Pipe)
```

**Remarks:** Pipe is in the range 0 -> MaxPipe-1

**Return Value:** Zero is returned if the UBEA is not installed.
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetGetRxPipeStatusRdmCapable</td>
<td>Returns the Remote Device Management (RDM) capabilities of the node from which data was last received</td>
<td>bool ArtNetGetRxPipeStatusRdmCapable( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if the node is able to support RDM. RDM is a new DMX512-A feature allowing bi-directional communication.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeStatusRomBoot</td>
<td>Returns the firmware status of the node from which data was last received.</td>
<td>bool ArtNetGetRxPipeStatusRomBoot( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if the node is executing original factory programmed firmware. False if the node is executing from non-volatile memory.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetRxPipeShortName</td>
<td>Return the short name of the node from which data was last received.</td>
<td>Char* ArtNetGetRxPipeShortName( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>The short name.</td>
<td>ArtNetGetRxPipeLongName()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeLongName</td>
<td>Return the long name of the node from which data was last received.</td>
<td>Char* ArtNetGetRxPipeLongName( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>The long name.</td>
<td>ArtNetGetRxPipeShortName()</td>
</tr>
<tr>
<td>ArtNetGetRxPipeNodeReport</td>
<td>Return the text string status report of the node from which data was last received.</td>
<td>Char* ArtNetGetRxPipeNodeReport( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>The Node Report contains human readable status information from the node, such as the results of it's power on test.</td>
<td>ArtNetGetRxPipeLongName()</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td><strong>Syntax</strong></td>
<td><strong>Remarks</strong></td>
<td><strong>Return Value</strong></td>
<td><strong>See also</strong></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td><strong>ArtNetGetRxPipeMacroKeys</strong></td>
<td>Return the macro key bit fields of the node from which data was last received.</td>
<td>BYTE ArtNetGetRxPipeMacroKeys( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>Byte containing bit fields representing the macro keys.</td>
<td>ArtNetGetRxPipeRemoteKeys()</td>
</tr>
<tr>
<td><strong>ArtNetGetRxPipeRemoteKeys</strong></td>
<td>Return the remote key bit fields of the node from which data was last received.</td>
<td>BYTE ArtNetGetRxPipeRemoteKeys( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>Byte containing bit fields representing the remote keys.</td>
<td>ArtNetGetRxPipeMacroKeys()</td>
</tr>
<tr>
<td><strong>ArtNetGetRxPipeVideo</strong></td>
<td>Return the remote video display switch of the node from which data was last received.</td>
<td>BYTE ArtNetGetRxPipeVideo( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td></td>
<td>ArtNetGetRxPipeMacroKeys()</td>
</tr>
</tbody>
</table>
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:**
```c
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:**
```c
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:**
```c
BYTE ArtNetGetTxPipeUniverse( int Pipe)
```

**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:** 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:**
```c
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
### ArtNetGetNodeIndexForTxPipe

**Purpose:** Cross references an entry in the transmit part of PipeLibrary to the NodeLibrary.

**Syntax:**
```c
int ArtNetGetNodeIndexForTxPipe(void)
```

**Remarks:**

- Returns a NodeLibrary Index value if the IP addresses can be matched. Otherwise returns MaxNodes.

**See also:** ArtNetGetPipeIndexForUniverse

### ArtNetGetTxPipeIpAddress

**Purpose:** Return the IP address of the node to which data was last sent.

**Syntax:**
```c
BYTE* ArtNetGetTxPipeIpAddress( int Pipe)
```

**Remarks:**
- Pipe is in the range 0 -> MaxPipe-1
- 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:** ArtNetGetRxPipeIpAddress()

### ArtNetGetTxPipeEstaMan

**Purpose:** Return the EstMan field of the node to which data was last sent.

**Syntax:**
```c
WORD ArtNetGetTxPipeEstaMan( int Pipe)
```

**Remarks:**
- Pipe is in the range 0 -> MaxPipe-1
- This field contains the ESTA manufacturer ID of the node to which data was last sent. The default value is 0x0000.

**See also:** ArtNetGetNodeEstaMan()

### ArtNetGetTxPipeEstaManString

**Purpose:** Returns a string detailing the name of the Node manufacturer.

**Syntax:**
```c
char* ArtNetGetTxPipeEstaManString( int Pipe)
```

**Remarks:**
- Pipe is in the range 0 -> MaxPipe-1
- 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()

### ArtNetGetTxPipeOemH

**Purpose:** Return the Art-Net OemH field of the node to which data was last sent.

**Syntax:**
```c
BYTE ArtNetGetTxPipeOemH( int Pipe)
```

**Remarks:**
- Pipe is in the range 0 -> MaxPipe-1
- 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()
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetGetTxPipeOem</td>
<td>Return the Art-Net Oem field of the node to which data was last sent.</td>
<td>BYTE ArtNetGetTxPipeOem( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>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.</td>
</tr>
<tr>
<td>ArtNetGetTxPipeOemString</td>
<td>Returns a string detailing the product name and type of the node to which data was last sent.</td>
<td>char* ArtNetGetTxPipeEstaManString( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>Variable length string containing the product name and type. If the name is unknown, the return string indicates the hexadecimal value of this field.</td>
</tr>
<tr>
<td>ArtNetGetTxPipeVersionInfoH</td>
<td>Return the firmware revision of the node to which data was last sent.</td>
<td>BYTE ArtNetGetTxPipeVersionInfoH( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>This field contains the high byte of the firmware revision of the node to which data was last sent. The default value is 0x00.</td>
</tr>
<tr>
<td>ArtNetGetTxPipeVersionInfo</td>
<td>Return the firmware revision of the node to which data was last sent.</td>
<td>BYTE ArtNetGetTxPipeVersionInfo( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>This field contains the low byte of the firmware revision of the node to which data was last sent. The default value is 0x00.</td>
</tr>
<tr>
<td>ArtNetGetTxPipeNumberPorts</td>
<td>Return the number of transmit DMX512 ports supported by the node to which data was last sent.</td>
<td>BYTE ArtNetGetTxPipeNumberPorts( int Pipe)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>The maximum number of ports supported by an Art-Net node is 4. The return value is in the range 1 -&gt; 4. The default value is 0x00. NB: A node with 4 ports would have 4 associated pipes.</td>
</tr>
</tbody>
</table>

See also: ArtNetSetRxPipeRequestUniverse()
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArtNetGetTxPipeStatusUbeaActive</strong></td>
<td>Returns the User Bios Extension Area (UBEA) status of the node to which data was last sent.</td>
<td><code>bool ArtNetGetTxPipeStatusUbeaActive(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><strong>ArtNetGetTxPipeUbeaVersion</strong></td>
<td>Returns the firmware version number of the User Bios Extension Area (UBEA) of the node to which data was last sent.</td>
<td><code>BYTE ArtNetGetTxPipeUbeaVersion(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>Zero is returned if the UBEA is not installed.</td>
<td></td>
</tr>
<tr>
<td><strong>ArtNetGetTxPipeStatusRdmCapable</strong></td>
<td>Returns the Remote Device Management (RDM) capabilities of the node to which data was last sent.</td>
<td><code>bool ArtNetGetTxPipeStatusRdmCapable(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if the node is able to support RDM. RDM is a new DMX512-A feature allowing bi-directional communication.</td>
<td></td>
</tr>
<tr>
<td><strong>ArtNetGetTxPipeStatusRomBoot</strong></td>
<td>Returns the firmware status of the node to which data was last sent.</td>
<td><code>bool ArtNetGetTxPipeStatusRomBoot(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>True if the node is executing original factory programmed firmware. False if the node is executing from non-volatile memory.</td>
<td></td>
</tr>
<tr>
<td><strong>ArtNetGetTxPipeShortName</strong></td>
<td>Return the short name of the node from which data was last received.</td>
<td><code>Char* ArtNetGetTxPipeShortName(int Pipe)</code></td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td>The short name.</td>
<td>ArtNetGetRxPipeLongName()</td>
</tr>
</tbody>
</table>
### ArtNetGetTxPipeLongName

**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()

### ArtNetGetTxPipeNodeReport

**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 its power on test.

**See also:** ArtNetGetRxPipeLongName()

### ArtNetGetTxPipeMacroKeys

**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()

### ArtNetGetTxPipeRemoteKeys

**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()

### ArtNetGetTxPipeVideo

**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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetGetNodeIpAddress</td>
<td>Return the IP address of the node.</td>
<td><code>char* ArtNetGetNodeIpAddress(int Node)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>Returns a string in the format <code>www.xxx.yyy.zzz</code>. If the node has not received data, 000.000.000.000 will be returned</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeTxUniverse</td>
<td>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.</td>
<td><code>BYTE ArtNetGetNodeTxUniverse(int Node, int Uni)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>Uni is in the range 0 -&gt; 3</td>
<td>Universe is in the range 0x00 -&gt; 0xff. The high nibble is the 'Sub-Net'.</td>
</tr>
<tr>
<td>ArtNetGetNodeIndexForIp</td>
<td>Return the node index of the NodeEntry for which the IP address matches.</td>
<td><code>void ArtNetGetNodeIndexForIp(char* IpAddress)</code></td>
<td>IpAddress is a character string in the format <code>www.xxx.yyy.zzz.zzzz</code>.</td>
<td>MaxNode is returned if a match is not found</td>
<td>ArtNetGetRxPipeIndexForIp()</td>
</tr>
<tr>
<td>ArtNetGetNodeTxImplemented</td>
<td>Returns whether this transmit universe is physically implemented for the specified universe.</td>
<td><code>unsigned int ArtNetGetNodeTxImplemented(int Node, int Uni)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>Uni is in the range 0 -&gt; 3</td>
<td>True if the port exists.</td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td>Return Value</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNode RxUniverse</td>
<td>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.</td>
<td>BYTE ArtNetGetNodeRxUniverse( int Node, int Uni)</td>
<td>Node is in the range 0 -&gt; MaxNode-1 Un is in the range 0 -&gt; 3</td>
<td>Universe is in the range 0x00 -&gt; 0xff. The high nibble is the 'Sub-Net'.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNode RxImplemented</td>
<td>Returns whether this receive universe is physically implemented for the specified universe.</td>
<td>unsigned int ArtNetGetNodeRxImplemented(int Node, int Uni)</td>
<td>Node is in the range 0 -&gt; MaxNode-1 Un is in the range 0 -&gt; 3</td>
<td>True if the port exists.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNode EstaMan</td>
<td>Return the EstMan field of the node.</td>
<td>WORD ArtNetGetNodeEstaMan( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>This field contains the ESTA manufacturer ID of the node to which data was last sent. The default value is 0x0000.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNode EstaManString</td>
<td>Returns a string detailing the name of the Node manufacturer.</td>
<td>char* ArtNetGetNodeEstaManString( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>Variable length string containing the manufacturer name. If the name is unknown, the return string indicates the hexadecimal value of this field.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNode OemH</td>
<td>Return the Art-Net OemH field.</td>
<td>BYTE ArtNetGetNodeOemH( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td>Return Value</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeOem</td>
<td>Return the Art-Net Oem field.</td>
<td>BYTE ArtNetGetNodeOem( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeOemString</td>
<td>Returns a string detailing the product name and type of the node.</td>
<td>char* ArtNetGetNodeOemString( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>Variable length string containing the product name and type. If the name is unknown, the return string indicates the hexadecimal value of this field.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeVersionInfoH</td>
<td>Return the firmware revision of the node.</td>
<td>BYTE ArtNetGetNodeVersionInfoH( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>This field contains the high byte of the firmware revision of the node. The default value is 0x00.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeVersionInfo</td>
<td>Return the firmware revision of the node.</td>
<td>BYTE ArtNetGetNodeVersionInfo( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>This field contains the low byte of the firmware revision of the node. The default value is 0x00.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetTxPipeNumberPorts</td>
<td>Return the number of DMX512 ports supported by the node.</td>
<td>BYTE ArtNetGetTxPipeNumberPorts( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>The maximum number of ports supported by an Art-Net node is 4. The return value is in the range 1 -&gt; 4. The default value is 0x00. NB. A node with 4 ports would have 4 associated pipes.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>Syntax</td>
<td>Remarks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeStatusIndicators</td>
<td>Returns the state of the node's front panel indicators.</td>
<td>bool ArtNetGetNodeStatusIndicators( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeStatusAuthority</td>
<td>Returns the Programming Authority for the subnet 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.</td>
<td>bool ArtNetGetNodeStatusAuthority( int Node)</td>
<td>Pipe is in the range 0 -&gt; MaxPipe-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeStatusUbeaActive</td>
<td>Returns the User Bios Extension Area (UBEA) status of the node.</td>
<td>bool ArtNetGetNodeStatusUbeaActive( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeUbeaVersion</td>
<td>Returns the firmware version number of the User Bios Extension Area (UBEA) of the node.</td>
<td>BYTE ArtNetGetNodeUbeaVersion( int Node)</td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also:
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetGetNodeStatusRdmCapable</td>
<td>Returns the Remote Device Management (RDM) capabilities of the node.</td>
<td><code>bool ArtNetGetNodeStatusRdmCapable( int Node)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>True if the node is able to support RDM. RDM is a new DMX512-A feature allowing bi-directional communication.</td>
</tr>
<tr>
<td>ArtNetGetNodeStatusRomBoot</td>
<td>Returns the firmware status of the.</td>
<td><code>bool ArtNetGetNodeStatusRomBoot( int Node)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>True if the node is executing original factory programmed firmware. False if the node is executing from non-volatile memory.</td>
</tr>
<tr>
<td>ArtNetGetNodeShortName</td>
<td>Return the short name of the node.</td>
<td><code>char* ArtNetGetNodeShortName( int Node)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>The short name.</td>
</tr>
<tr>
<td>ArtNetGetNodeLongName</td>
<td>Return the long name of the node.</td>
<td><code>char* ArtNetGetNodeLongName( int Node)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>The long name.</td>
</tr>
<tr>
<td>ArtNetGetNodeNodeReport</td>
<td>Return the text string status report of the node.</td>
<td><code>Char* ArtNetGetNodeNodeReport( int Node)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>The Node Report contains human readable status information from the node, such as the results of its power on test.</td>
</tr>
<tr>
<td>ArtNetGetNodeMacroKeys</td>
<td>Return the macro key bit fields of the node</td>
<td><code>BYTE ArtNetGetNodeMacroKeys( int Node)</code></td>
<td>Node is in the range 0 -&gt; MaxNode-1</td>
<td>Byte containing bit fields representing the macro keys.</td>
</tr>
</tbody>
</table>
| **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: |
| **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: |
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
</tr>
</thead>
</table>
| ArtNetGetNodeRxReceived | Return the receive status of the node. | `bool ArtNetGetNodeRxReceived(int Node, int Uni)` | Node is in the range 0 -> MaxNode-1  
Uni is in the range 0 -> 3 | True if data is being received. |
| ArtNetGetNodeRxErrors | Return the receive errors status of the node. | `bool ArtNetGetNodeRxErrors(int Node, int Uni)` | Node is in the range 0 -> MaxNode-1  
Uni is in the range 0 -> 3 | True if data errors are detected. |
| ArtNetGetNodeRxTest | Return the receive test packet status of the node. | `bool ArtNetGetNodeRxTest(int Node, int Uni)` | Node is in the range 0 -> MaxNode-1  
Uni is in the range 0 -> 3 | True if test packets are being received. |
| ArtNetGetNodeRxSip | Return the system information test packet status of the node. | `bool ArtNetGetNodeRxSip(int Node, int Uni)` | Node is in the range 0 -> MaxNode-1  
Uni is in the range 0 -> 3 | True if system information packets are being received. |
| ArtNetGetNodeRxText | Return the receive text packet status of the node. | `bool ArtNetGetNodeRxText(int Node, int Uni)` | Node is in the range 0 -> MaxNode-1  
Uni is in the range 0 -> 3 | True if text packets are being received. |
| ArtNetGetNodeRxDisable | Return the receive enable status of the node. | `bool ArtNetGetNodeRxDisable(int Node, int Uni)` | Node is in the range 0 -> MaxNode-1  
Uni is in the range 0 -> 3 | True if this receiver has been disabled. |
<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetGetNodeTxGood</td>
<td>Return the transmit active status of the node.</td>
<td><code>bool ArtNetGetNodeTxGood(int Node, int Uni)</code></td>
<td>Node is in the range 0 → MaxNode-1 Uni is in the range 0 → 3</td>
<td>True if this transmitter is currently transmitting DMX512.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeTxMerge</td>
<td>Return the transmit merge status of the node.</td>
<td><code>bool ArtNetGetNodeTxMerge(int Node, int Uni)</code></td>
<td>Node is in the range 0 → MaxNode-1 Uni is in the range 0 → 3</td>
<td>True if this transmitter is currently merging data from two sources.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeTxLtp</td>
<td>Return the transmit merge mode of the node.</td>
<td><code>bool ArtNetGetNodeTxLtp(int Node, int Uni)</code></td>
<td>Node is in the range 0 → MaxNode-1 Uni is in the range 0 → 3</td>
<td>True if this transmitter is set to merge in LTP. False if Htp.</td>
<td></td>
</tr>
<tr>
<td>ArtNetGetNodeTxShort</td>
<td>Return the transmit hardware driver status of the node.</td>
<td><code>bool ArtNetGetNodeTxShort(int Node, int Uni)</code></td>
<td>Node is in the range 0 → MaxNode-1 Uni is in the range 0 → 3</td>
<td>True if this transmitter output is shorted.</td>
<td></td>
</tr>
<tr>
<td>ArtNetSetNodeShortName</td>
<td>Sets the short name of a node.</td>
<td><code>void ArtNetSetNodeShortName( int Node char* Name)</code></td>
<td>Node is in the range 0 → MaxNode-1</td>
<td>Void</td>
<td></td>
</tr>
<tr>
<td>ArtNetSetNodeLongName</td>
<td>Sets the long name of the node.</td>
<td><code>void ArtNetSetIpLongName (int Node char* Name)</code></td>
<td>Node is in the range 0 → MaxNode-1</td>
<td>Void</td>
<td></td>
</tr>
<tr>
<td><strong>ArtNetSetNodeSwitches</strong></td>
<td><strong>Purpose:</strong> Sets the front panel switches of the node.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td>void ArtNetSetIpSwitches( int Node, BYTE In0, BYTE In1, BYTE In2, BYTE In3, BYTE Out0, BYTE Out1, BYTE Out2, BYTE Out3, BYTE SubNet, BYTE Video)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
<td>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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
<td>Void</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>See also:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>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:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ArtNetSetIpSwitches(</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0, // NodeLib entry</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x80, // low byte input 0 address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x81,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x82, // NB Bit 7 set to enable change</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x83,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x86, // low byte output 0 address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x87,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x88,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x89,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x81, // hi byte of in &amp; out address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x00 // bit 7 not set, so no affect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>);</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ArtNetSetNodeCommand

**Purpose:**
Sets the long name of the node that matches IP address.

**Syntax:**
```c
void ArtNetSetNodeCommand(int Node, BYTE NewCommand)
```

**Remarks:**
Node is in the range 0 -> MaxNode-1
NewCommand is defined as follows:

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

**Return Value:**
void

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

<table>
<thead>
<tr>
<th>ArtNetResetCounters</th>
<th>Purpose:</th>
<th>Reset the packet counters for both transmit and receive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void ArtNetResetCounters (void)</td>
<td>Remarks:</td>
</tr>
<tr>
<td>Return Value:</td>
<td>void</td>
<td>See also:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ArtNetGetRxPipePacketCount</th>
<th>Purpose:</th>
<th>Return the number of receive packets for this pipe since the last ArtNetResetCounters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>WORD ArtNetGetRxPipePacketCount (void)</td>
<td>Remarks:</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Packet count.</td>
<td>See also: ArtNetResetCounters()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ArtNetGetTxPipePacketCount</th>
<th>Purpose:</th>
<th>Return the number of transmit packets sent via this pipe since the last ArtNetResetCounters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>WORD ArtNetGetTxPipePacketCount (void)</td>
<td>Remarks:</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Packet count.</td>
<td>See also: ArtNetResetCounters()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ArtNetGetRxPipeChannelCount</th>
<th>Purpose:</th>
<th>Return the number of DMX512 channels (slots) received by this pipe.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>WORD ArtNetGetRxPipeChannelCount (void)</td>
<td>Remarks:</td>
</tr>
<tr>
<td>Return Value:</td>
<td>0 if no data received, otherwise in the range 1 -&gt; 512.</td>
<td>See also: ArtNetResetCounters()</td>
</tr>
</tbody>
</table>
### 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'.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Syntax</th>
<th>Remarks</th>
<th>Return Value</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNetShowScopeForm</td>
<td>Displays the 'Scope' Form which contains an 'oscilloscope' and a 'digital meter' for channel levels.</td>
<td>bool ArtNetShowScopeForm(void)</td>
<td></td>
<td>False if not enough resources to create dialogue.</td>
<td></td>
</tr>
<tr>
<td>ArtNetShowAboutBox</td>
<td>Displays the driver about box which among other items contains the credits for all us hard working programmers!</td>
<td>bool ArtNetShowAboutBox(void)</td>
<td></td>
<td>False if not enough resources to create dialogue.</td>
<td></td>
</tr>
<tr>
<td>ArtNetShowNetworkStatusForm</td>
<td>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.</td>
<td>bool ArtNetShowNetworkStatusForm(void)</td>
<td></td>
<td>False if not enough resources to create dialogue.</td>
<td></td>
</tr>
<tr>
<td>ArtNetShowDataMonitorForm</td>
<td>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.</td>
<td>bool ArtNetShowDataMonitorForm(void)</td>
<td></td>
<td>False if not enough resources to create dialogue.</td>
<td></td>
</tr>
</tbody>
</table>
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.

```c
#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\n");
    if(trial(0x278))
        exit(0);
    printf("DMX-Dongle not found at 0x278\n");
    if(trial(0x3bc))
        exit(0);
    printf("DMX-Dongle not found at 0x3bc\n");
    printf("DMX-Dongle not found at 0x3bc\n");
}
```

MS-DOS DEVELOPMENT
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\n",addr);
        printf("Dongle type number = %d\n",get_dongle_type());
        printf("Dongle firmware revision = %d\n",get_firmware());
        printf("Please Wait for Benchmark.\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.\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.

```c
#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) /* read 128 channels of dmx data starting at channel 1 */
    {
        set_rx_header(0);
        reset_rx_base(0);

        rx_dmx_ptr(receive);

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

```c
unsigned char get_rx_header(void)
```

**Remarks:**

**Return Value:** header

**See also:**

**set_tx_header**

**Purpose:** Set new transmit dmx header code.

**Syntax:**

```c
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:**

```c
unsigned char get_tx_header(void)
```

**Remarks:**

**Return Value:** header

**See also:**

**set_tx_break_time**

**Purpose:** Set new transmit break time.

**Syntax:**

```c
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:**

```c
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:**

```c
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
**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**

```c
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 comptability
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 \texttt{rx\_reset\_counters()} was last called.

\texttt{unsigned int rx\_ctrl.ovr\_count}

A 16 bit count of the number of serial overrun errors detected since the function \texttt{rx\_reset\_counters()} was last called.

\texttt{unsigned int rx\_ctrl.hdr\_count}

A 16 bit count of the number of non matching header codes received since the function \texttt{rx\_reset\_counters()} was last called.

\texttt{unsigned int rx\_ctrl.frm\_count}

A 16 bit count of the number of framing errors detected since the function \texttt{rx\_reset\_counters()} was last called. Framing errors exclude line breaks.

\texttt{RX}

typedef struct {
    unsigned char cur\_level[MAX\_CHAN];
    unsigned char min\_level[MAX\_CHAN];
    unsigned char max\_level[MAX\_CHAN];
}RX;

The \texttt{rx} structure contains all channel data received by the dongle. The structure is updated by the function call \texttt{rx\_dmx()}.  

\texttt{unsigned char rx\_cur\_level[]}

An array of 512 entries showing the current channel levels at the last \texttt{rx\_dmx()} call.

\texttt{unsigned char rx\_min\_level[]}

An array of 512 entries showing the minimum level received.

\texttt{unsigned char rx\_max\_level[]}

An array of 512 entries showing the maximum level received.

\texttt{MEM}

\texttt{#define MAX\_MEM 50}
\texttt{#define LEGEND\_LENGTH 35}
\texttt{#define MAX\_CHAN 512}

typedef struct {
    unsigned char level[MAX\_CHAN];
    char legend[LEGEND\_LENGTH+1];
}MEM;

MEM mem[MAX\_MEM];

The \texttt{mem} array is used to build 50 complete dmx memories complete with alphanumeric legends. The \texttt{tx\_memory()} function is used to transmit any one of the memories to the dmx output. The \texttt{mem} array may be accessed directly to set data levels.
### Conversion Tables

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0000 0000</td>
<td>32</td>
<td>20</td>
<td>0010 0000</td>
<td>64</td>
<td>40</td>
<td>0100 0000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>0000 0001</td>
<td>33</td>
<td>21</td>
<td>0010 0001</td>
<td>65</td>
<td>41</td>
<td>0100 0001</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0000 0010</td>
<td>34</td>
<td>22</td>
<td>0010 0010</td>
<td>66</td>
<td>42</td>
<td>0100 0010</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>0000 0011</td>
<td>35</td>
<td>23</td>
<td>0010 0011</td>
<td>67</td>
<td>43</td>
<td>0100 0011</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>0000 0100</td>
<td>36</td>
<td>24</td>
<td>0010 0100</td>
<td>68</td>
<td>44</td>
<td>0100 0100</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>0000 0101</td>
<td>37</td>
<td>25</td>
<td>0010 0101</td>
<td>69</td>
<td>45</td>
<td>0100 0101</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>0000 0110</td>
<td>38</td>
<td>26</td>
<td>0010 0110</td>
<td>70</td>
<td>46</td>
<td>0100 0110</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>0000 0111</td>
<td>39</td>
<td>27</td>
<td>0010 0111</td>
<td>71</td>
<td>47</td>
<td>0100 0111</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>0000 1000</td>
<td>40</td>
<td>28</td>
<td>0010 1000</td>
<td>72</td>
<td>48</td>
<td>0100 1000</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>0000 1001</td>
<td>41</td>
<td>29</td>
<td>0010 1001</td>
<td>73</td>
<td>49</td>
<td>0100 1001</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>0000 1010</td>
<td>42</td>
<td>2A</td>
<td>0010 1010</td>
<td>74</td>
<td>4A</td>
<td>0100 1010</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>0000 1011</td>
<td>43</td>
<td>2B</td>
<td>0010 1011</td>
<td>75</td>
<td>4B</td>
<td>0100 1011</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>0000 1100</td>
<td>44</td>
<td>2C</td>
<td>0010 1100</td>
<td>76</td>
<td>4C</td>
<td>0100 1100</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>0000 1101</td>
<td>45</td>
<td>2D</td>
<td>0010 1101</td>
<td>77</td>
<td>4D</td>
<td>0100 1101</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>0000 1110</td>
<td>46</td>
<td>2E</td>
<td>0010 1110</td>
<td>78</td>
<td>4E</td>
<td>0100 1110</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>0000 1111</td>
<td>47</td>
<td>2F</td>
<td>0010 1111</td>
<td>79</td>
<td>4F</td>
<td>0100 1111</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>0001 0000</td>
<td>48</td>
<td>30</td>
<td>0011 0000</td>
<td>80</td>
<td>50</td>
<td>0101 0000</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>0001 0001</td>
<td>49</td>
<td>31</td>
<td>0011 0001</td>
<td>81</td>
<td>51</td>
<td>0101 0001</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>0001 0010</td>
<td>50</td>
<td>32</td>
<td>0011 0010</td>
<td>82</td>
<td>52</td>
<td>0101 0010</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>0001 0011</td>
<td>51</td>
<td>33</td>
<td>0011 0011</td>
<td>83</td>
<td>53</td>
<td>0101 0011</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>0001 0100</td>
<td>52</td>
<td>34</td>
<td>0011 0100</td>
<td>84</td>
<td>54</td>
<td>0101 0100</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>0001 0101</td>
<td>53</td>
<td>35</td>
<td>0011 0101</td>
<td>85</td>
<td>55</td>
<td>0101 0101</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>0001 0110</td>
<td>54</td>
<td>36</td>
<td>0011 0110</td>
<td>86</td>
<td>56</td>
<td>0101 0110</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>0001 0111</td>
<td>55</td>
<td>37</td>
<td>0011 0111</td>
<td>87</td>
<td>57</td>
<td>0101 0111</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>0001 1000</td>
<td>56</td>
<td>38</td>
<td>0011 1000</td>
<td>88</td>
<td>58</td>
<td>0101 1000</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>0001 1001</td>
<td>57</td>
<td>39</td>
<td>0011 1001</td>
<td>89</td>
<td>59</td>
<td>0101 1001</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>0001 1010</td>
<td>58</td>
<td>3A</td>
<td>0011 1010</td>
<td>90</td>
<td>5A</td>
<td>0101 1010</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>0001 1011</td>
<td>59</td>
<td>3B</td>
<td>0011 1011</td>
<td>91</td>
<td>5B</td>
<td>0101 1011</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>0001 1100</td>
<td>60</td>
<td>3C</td>
<td>0011 1100</td>
<td>92</td>
<td>5C</td>
<td>0101 1100</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>0001 1101</td>
<td>61</td>
<td>3D</td>
<td>0011 1101</td>
<td>93</td>
<td>5D</td>
<td>0101 1101</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>0001 1110</td>
<td>62</td>
<td>3E</td>
<td>0011 1110</td>
<td>94</td>
<td>5E</td>
<td>0101 1110</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>0001 1111</td>
<td>63</td>
<td>3F</td>
<td>0011 1111</td>
<td>95</td>
<td>5F</td>
<td>0101 1111</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Binary</td>
<td>Dec</td>
<td>Hex</td>
<td>Binary</td>
<td>Dec</td>
<td>Hex</td>
<td>Binary</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>--------------</td>
<td>-----</td>
<td>------</td>
<td>--------------</td>
<td>-----</td>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>96</td>
<td>60</td>
<td>0110 0000</td>
<td>128</td>
<td>80</td>
<td>1000 0000</td>
<td>160</td>
<td>A0</td>
<td>1010 0000</td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>0110 0001</td>
<td>129</td>
<td>81</td>
<td>1000 0001</td>
<td>161</td>
<td>A1</td>
<td>1010 0001</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>0110 0010</td>
<td>130</td>
<td>82</td>
<td>1000 0010</td>
<td>162</td>
<td>A2</td>
<td>1010 0010</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>0110 0011</td>
<td>131</td>
<td>83</td>
<td>1000 0011</td>
<td>163</td>
<td>A3</td>
<td>1010 0011</td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>0110 0100</td>
<td>132</td>
<td>84</td>
<td>1000 0100</td>
<td>164</td>
<td>A4</td>
<td>1010 0100</td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>0110 0101</td>
<td>133</td>
<td>85</td>
<td>1000 0101</td>
<td>165</td>
<td>A5</td>
<td>1010 0101</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>0110 0110</td>
<td>134</td>
<td>86</td>
<td>1000 0110</td>
<td>166</td>
<td>A6</td>
<td>1010 0110</td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>0110 0111</td>
<td>135</td>
<td>87</td>
<td>1000 0111</td>
<td>167</td>
<td>A7</td>
<td>1010 0111</td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>0110 1000</td>
<td>136</td>
<td>88</td>
<td>1000 1000</td>
<td>168</td>
<td>A8</td>
<td>1010 1000</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>0110 1001</td>
<td>137</td>
<td>89</td>
<td>1000 1001</td>
<td>169</td>
<td>A9</td>
<td>1010 1001</td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>0110 1010</td>
<td>138</td>
<td>8A</td>
<td>1000 1010</td>
<td>170</td>
<td>AA</td>
<td>1010 1010</td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>0110 1011</td>
<td>139</td>
<td>8B</td>
<td>1000 1011</td>
<td>171</td>
<td>AB</td>
<td>1010 1011</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>0110 1100</td>
<td>140</td>
<td>8C</td>
<td>1000 1100</td>
<td>172</td>
<td>AC</td>
<td>1010 1100</td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>0110 1101</td>
<td>141</td>
<td>8D</td>
<td>1000 1101</td>
<td>173</td>
<td>AD</td>
<td>1010 1101</td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>0110 1110</td>
<td>142</td>
<td>8E</td>
<td>1000 1110</td>
<td>174</td>
<td>AE</td>
<td>1010 1110</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>0110 1111</td>
<td>143</td>
<td>8F</td>
<td>1000 1111</td>
<td>175</td>
<td>AF</td>
<td>1010 1111</td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>0111 0000</td>
<td>144</td>
<td>90</td>
<td>1001 0000</td>
<td>176</td>
<td>B0</td>
<td>1011 0000</td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>0111 0001</td>
<td>145</td>
<td>91</td>
<td>1001 0001</td>
<td>177</td>
<td>B1</td>
<td>1011 0001</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>0111 0010</td>
<td>146</td>
<td>92</td>
<td>1001 0010</td>
<td>178</td>
<td>B2</td>
<td>1011 0010</td>
</tr>
<tr>
<td>115</td>
<td>73</td>
<td>0111 0011</td>
<td>147</td>
<td>93</td>
<td>1001 0011</td>
<td>179</td>
<td>B3</td>
<td>1011 0011</td>
</tr>
<tr>
<td>116</td>
<td>74</td>
<td>0111 0100</td>
<td>148</td>
<td>94</td>
<td>1001 0100</td>
<td>180</td>
<td>B4</td>
<td>1011 0100</td>
</tr>
<tr>
<td>117</td>
<td>75</td>
<td>0111 0101</td>
<td>149</td>
<td>95</td>
<td>1001 0101</td>
<td>181</td>
<td>B5</td>
<td>1011 0101</td>
</tr>
<tr>
<td>118</td>
<td>76</td>
<td>0111 0110</td>
<td>150</td>
<td>96</td>
<td>1001 0110</td>
<td>182</td>
<td>B6</td>
<td>1011 0110</td>
</tr>
<tr>
<td>119</td>
<td>77</td>
<td>0111 0111</td>
<td>151</td>
<td>97</td>
<td>1001 0111</td>
<td>183</td>
<td>B7</td>
<td>1011 0111</td>
</tr>
<tr>
<td>120</td>
<td>78</td>
<td>0111 1000</td>
<td>152</td>
<td>98</td>
<td>1001 1000</td>
<td>184</td>
<td>B8</td>
<td>1011 1000</td>
</tr>
<tr>
<td>121</td>
<td>7A</td>
<td>0111 1010</td>
<td>153</td>
<td>99</td>
<td>1001 1010</td>
<td>185</td>
<td>B9</td>
<td>1011 1010</td>
</tr>
<tr>
<td>122</td>
<td>7B</td>
<td>0111 1011</td>
<td>154</td>
<td>9A</td>
<td>1001 1011</td>
<td>186</td>
<td>BA</td>
<td>1011 1011</td>
</tr>
<tr>
<td>123</td>
<td>7C</td>
<td>0111 1100</td>
<td>155</td>
<td>9B</td>
<td>1001 1100</td>
<td>187</td>
<td>BB</td>
<td>1011 1101</td>
</tr>
<tr>
<td>124</td>
<td>7D</td>
<td>0111 1101</td>
<td>156</td>
<td>9C</td>
<td>1001 1101</td>
<td>188</td>
<td>BC</td>
<td>1011 1100</td>
</tr>
<tr>
<td>125</td>
<td>7E</td>
<td>0111 1110</td>
<td>157</td>
<td>9D</td>
<td>1001 1110</td>
<td>189</td>
<td>BD</td>
<td>1011 1110</td>
</tr>
<tr>
<td>126</td>
<td>7F</td>
<td>0111 1111</td>
<td>158</td>
<td>9E</td>
<td>1001 1110</td>
<td>190</td>
<td>BE</td>
<td>1011 1110</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Binary</td>
<td>Dec</td>
<td>Hex</td>
<td>Binary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>---------</td>
<td>-----</td>
<td>-----</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>C0</td>
<td>1100 0000</td>
<td>224</td>
<td>E0</td>
<td>1110 0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>193</td>
<td>C1</td>
<td>1100 0001</td>
<td>225</td>
<td>E1</td>
<td>1110 0001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>194</td>
<td>C2</td>
<td>1100 0010</td>
<td>226</td>
<td>E2</td>
<td>1110 0010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>C3</td>
<td>1100 0011</td>
<td>227</td>
<td>E3</td>
<td>1110 0011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>196</td>
<td>C4</td>
<td>1100 0100</td>
<td>228</td>
<td>E4</td>
<td>1110 0100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>197</td>
<td>C5</td>
<td>1100 0101</td>
<td>229</td>
<td>E5</td>
<td>1110 0101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>198</td>
<td>C6</td>
<td>1100 0110</td>
<td>230</td>
<td>E6</td>
<td>1110 0110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>199</td>
<td>C7</td>
<td>1100 0111</td>
<td>231</td>
<td>E7</td>
<td>1110 0111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>C8</td>
<td>1100 1000</td>
<td>232</td>
<td>E8</td>
<td>1110 1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>C9</td>
<td>1100 1001</td>
<td>233</td>
<td>E9</td>
<td>1110 1001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>CA</td>
<td>1100 1010</td>
<td>234</td>
<td>EA</td>
<td>1110 1010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>CB</td>
<td>1100 1011</td>
<td>235</td>
<td>EB</td>
<td>1110 1011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>CC</td>
<td>1100 1100</td>
<td>236</td>
<td>EC</td>
<td>1110 1100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>CD</td>
<td>1100 1101</td>
<td>237</td>
<td>ED</td>
<td>1110 1101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>CE</td>
<td>1100 1110</td>
<td>238</td>
<td>EE</td>
<td>1110 1110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>CF</td>
<td>1100 1111</td>
<td>239</td>
<td>EF</td>
<td>1110 1111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>D0</td>
<td>1101 0000</td>
<td>240</td>
<td>F0</td>
<td>1111 0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>D1</td>
<td>1101 0001</td>
<td>241</td>
<td>F1</td>
<td>1111 0001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>D2</td>
<td>1101 0010</td>
<td>242</td>
<td>F2</td>
<td>1111 0010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>D3</td>
<td>1101 0011</td>
<td>243</td>
<td>F3</td>
<td>1111 0011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>D4</td>
<td>1101 0100</td>
<td>244</td>
<td>F4</td>
<td>1111 0100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>D5</td>
<td>1101 0101</td>
<td>245</td>
<td>F5</td>
<td>1111 0101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>D6</td>
<td>1101 0110</td>
<td>246</td>
<td>F6</td>
<td>1111 0110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>D7</td>
<td>1101 0111</td>
<td>247</td>
<td>F7</td>
<td>1111 0111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>D8</td>
<td>1101 1000</td>
<td>248</td>
<td>F8</td>
<td>1111 1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>D9</td>
<td>1101 1001</td>
<td>249</td>
<td>F9</td>
<td>1111 1001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>DA</td>
<td>1101 1010</td>
<td>250</td>
<td>FA</td>
<td>1111 1010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>DB</td>
<td>1101 1011</td>
<td>251</td>
<td>FB</td>
<td>1111 1011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>DC</td>
<td>1101 1100</td>
<td>252</td>
<td>FC</td>
<td>1111 1100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>221</td>
<td>DD</td>
<td>1101 1101</td>
<td>253</td>
<td>FD</td>
<td>1111 1101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>DE</td>
<td>1101 1110</td>
<td>254</td>
<td>FE</td>
<td>1111 1110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>DF</td>
<td>1101 1111</td>
<td>255</td>
<td>FF</td>
<td>1111 1111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ArtNetGetRxPipeRxGood · 37
ArtNetGetRxPipeRxSip · 38
ArtNetGetRxPipeRxTest · 38
ArtNetGetRxPipeRxText · 38
ArtNetGetRxPipeShortName · 40, 46
ArtNetGetRxPipeStatusRdmCapable · 40
ArtNetGetRxPipeStatusRomBoot · 40
ArtNetGetRxPipeStatusUbeaActive · 38, 39
ArtNetGetRxPipeUbeaVersion · 39
ArtNetGetRxPipeUniverse · 34
ArtNetGetRxPipeVersionInfo · 37, 44
ArtNetGetRxPipeVersionInfoH · 37
ArtNetGetRxPipeVideo · 41
ArtNetGetTxPipeEstaMan · 43
ArtNetGetTxPipeEstaManString · 43, 44
ArtNetGetTxPipeIpAddress · 43
ArtNetGetTxPipeLongName · 46
ArtNetGetTxPipeMacroKeys · 46
ArtNetGetTxPipeNodeReport · 46
ArtNetGetTxPipeNumberOfPorts · 44, 49
ArtNetGetTxPipeOem · 44
ArtNetGetTxPipeOemH · 43
ArtNetGetTxPipePacketCount · 58
ArtNetGetTxPipeRemoteKeys · 46
ArtNetGetTxPipeShortName · 45
ArtNetGetTxPipeStatusRdmCapable · 45
ArtNetGetTxPipeStatusRomBoot · 45
ArtNetGetTxPipeStatusUbeaActive · 45
ArtNetGetTxPipeUbeaVersion · 45
ArtNetGetTxPipeUniverse · 42
ArtNetGetTxPipeVersionInfo · 44
ArtNetGetTxPipeVideo · 46
ArtNetInit · 12, 16, 17, 22, 23, 29, 35
ArtNetIsPipe0Dongle · 34
ArtNetIsTxPipeActive · 34, 42
ArtNetReadWrite · 12, 15, 18, 22, 29, 33, 42, 47, 48, 58. See
ArtNetResetCounters · 58
ArtNetSendUdp · 30
ArtNetSetNodeShortName · 54
ArtNetSetCallBackUdpReceivePost · 31
ArtNetSetCallBackUdpReceivePre · 31
ArtNetSetCallBackUdpTransmit · 32
ArtNetSetIpLongName · 54
ArtNetSetIpSwitches · 55
ArtNetSetNodeCommand · 56
ArtNetSetRxPipeRequestUniverse · 33, 34, 35, 36, 37, 43, 44
ArtNetSetTxData · 20, 21
ArtNetSetTxDataGroup · 20, 21
ArtNetSetTxPipeUniverse · 42
ArtNetShowDataMonitorForm · 59
ArtNetShowNetworkStatusForm · 59
ArtNetShowScopeForm · 59
ArtNetSoftEnumerate · 35
ArtNetView · 10

B

bandwidth · 9, 15
Binary · 75, 76, 77
bios.h · 61, 62
Borland · 61
Break · 9, 72

C

conio.h · 61, 62
cype.h · 61, 62

D

DLL · 11, 19, 30
DMX-Dongle II · 1, 9, 12, 14, 16, 18, 58, 59
DMX-Hub · 10, 55
DMX-Workshop · 10
Dong32.vxd · 11
DongleArtNet.dll · 11
DongleArtNet.h · 11, 13, 30
DongleArtNet.lib · 11
DongleArtNet1.sys · 11
DongleArtNet2.sys · 11
DongleArtNet3.sys · 11
DongleGetBreakTime · 24
DongleGetChannelCount · 25
DongleGetFirmwareRevision · 19, 22
DongleGetFramingErrorCode · 27
DongleGetFrequency · 24, 25
DongleGetMabTime · 24
DongleGetOverrunErrorCode · 28
DongleGetPacketCount · 27
DongleGetPeriod · 24, 25
DongleGetPortAddress · 17, 22
DongleGetStartCodeErrorCode · 27
DongleGetTransferCount · 22
DongleGetType · 23
DongleResetRxCounters · 27
DongleSetAux · 26
DongleSetLoopOff · 25, 26
DongleSetLoopOn · 25, 26
DongleSetStartCode · 23, 27
DongleSetTransferCount · 15, 18, 22
DongleSetTxBreakTime · 23
DongleSetTxMabTime · 24
DongleSetTxStartCode · 23
Dos · 61
Email · 82
Enumerate · 35
ESTA · 36, 43, 48

Feats · 9, 10, 16, 20, 22, 27, 29, 33, 42, 47, 58, 59
firmware · 37, 39, 40, 44, 45, 49, 50, 51, 52, 59, 61, 65, 69, 73
Form · 59

GO · 13
Grand-Master Flash! · 10

header · 61, 62, 64, 67, 68, 69, 72, 73, 74
Hex · 75, 76, 77
InstallShield · 11

LEFT · 15
LongName · 30, 51, 54

Mark after Break · 9
MaxNode · 47, 48, 49, 50, 51, 52, 53, 54, 56
MaxPipe · 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46
ME · 1, 9, 11
memory · 16, 17, 40, 45, 51, 61, 64, 67, 74
MENU · 12
Merge · 9, 56
meter · 59

Mnemonic · 56
monitor · 33, 59

name · 27, 30, 36, 40, 43, 44, 45, 46, 48, 49, 51, 54, 56
Net-Link · 10
Network · 35, 59
NT · 1, 9, 11
Oem · 36, 43, 44, 48, 49
oscilloscope · 9, 59
Overview · 11, 61
Packet · 58
parallel · 9, 16
PipeEntry PipeLibrary[MaxPipes]; · 13
PipeLibrary · 13, 14, 18, 20, 21, 33, 34, 35, 42, 43
process.h · 61, 62
Quick Start · 9
RDM · 40, 45, 50
remote · 41, 46, 51
Remote Device Management · 40. See . See RequestUniverse · 33, 34, 35
resources · 59
RIGHT · 15
Scope · 59
SDK · 1, 9, 11
ShortName · 30, 54
Show-Control · 9, 10, 16, 20, 22, 27, 29, 33, 42, 47, 58, 59
source · 9, 61