Table of Contents

DLL and Document History

Version 3.0.0 Jan. 2014
First release of the DALI Access Developer Package, based on DALI USB interface.

Introduction

To be able to use the DALIBusAccess.dll the Lunatone DALI control program **DALI-Cockpit** has to be installed. See the Lunatone homepage www.lunatone.com for the newest version of this software.
The installer for DALI-Cockpit will also install the needed DALIBusServer.exe, the DALIMonitor.exe and make appropriate registry entries so a user application is able to locate the DLL.

The DALIBusAccess.dll and DALIBusServer.exe runs on any 32bit Windows System – that is any Windows version above Windows95. It was tested under Windows98, NT, Windows2000, XP, Vista and Windows 7. The DALI USB is supported on Windows98SE and above.

The DALI Developer Package contains the following files:

-	this documentation	Document in Adobe PDF- format
-	DALIBusAccessAPI.h	C/C++ header file, where all DLL functions are declared
-	DALICommands.h	C- Constant definitions used by the DLL functions
-	DALIdll.cpp, DALIdll.h	A C++ wrapper object for all DLL functions. This source code may also be useful if you need to port to a different programming language.
-	DaliBusAccess.lib	library to statically link to the DLL

To connect the PC to the DALI bus you will need one of the following DALI bus interface devices:

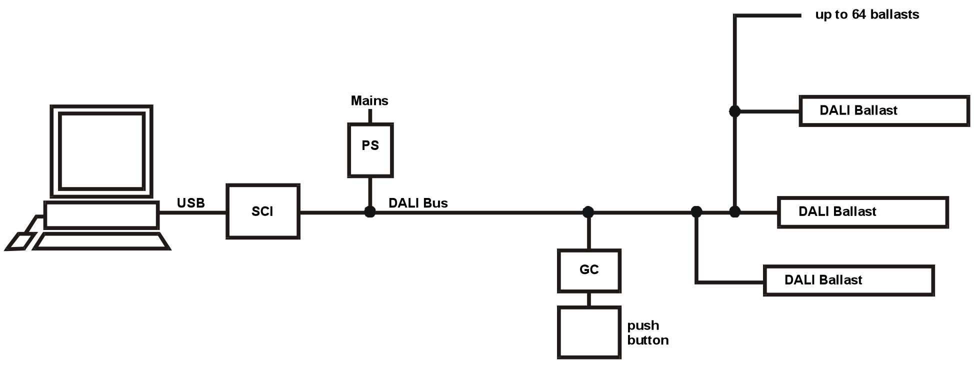
-	DALI SCI2	RS232 - DALI interface module.
-	DALI USB	like the SCI2, but with USB bus connection

Datatypes used in this documentation:

BOOL	a 32 bit integer that may be TRUE (1) or FALSE (0)
int	a 32 bit integer
HDALI	a 32 bit integer
DWORD	a 32bit unsigned integer
HWND	a window handle – declared as 32bit integer in windows.h

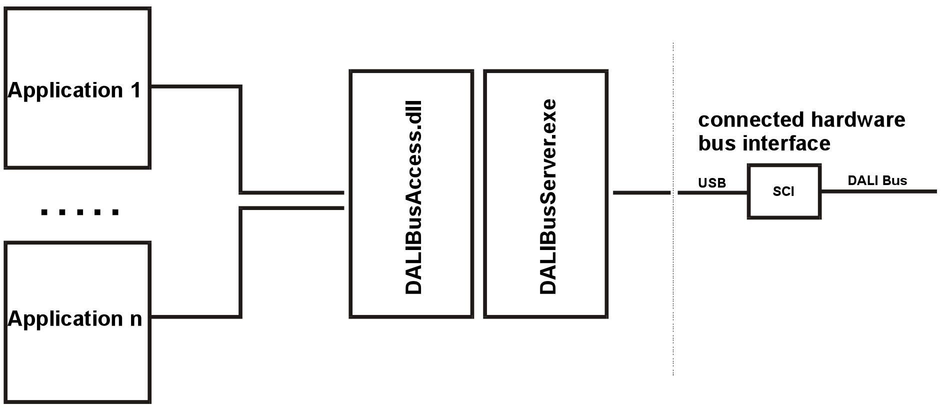
Basic Hardware Installation

Basic hardware installation:



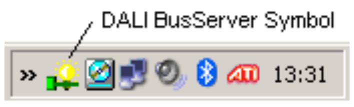
Software Structure

The Software structure is designed such that it is possible for more than one application to connect to the DALI bus interface device. This can bee seen as if the DALI bus was virtually extended into the PC, because all connected applications receive DALI frames from each other and from the physical DALI bus.



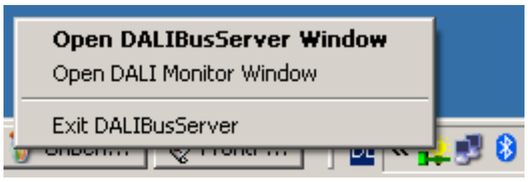
The DaliBusServer.exe application is the nerve center in this concept. It handles the opened DALI devices, the transmitted data and keeps an internal database of connected applications and interface devices.

The DaliBusServer.exe application will be started automatically at the time the first application calls the OpenPort() function of the DLL. The DALIBusServer runs in background and displays a icon in the Windows task bar area:

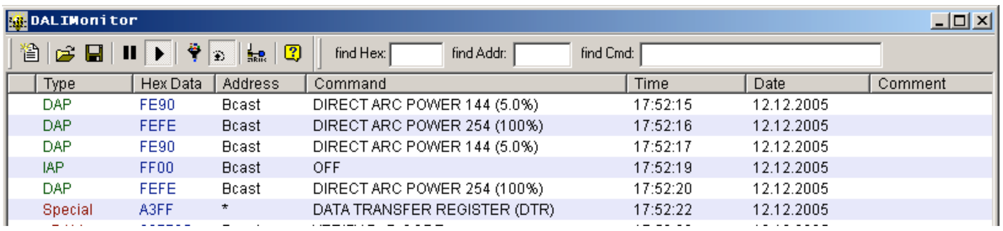


The DALIBusServer.exe will automatically be closed if there is no more device connected after a call to ClosePort().

A right click on the symbol opens the context menu for the DALIBusServer:



This lets you open the BusServer's information window and the DALIMonitor. The Monitor is especially useful for debugging your application and to see the DALI traffic on the bus:



Type	Hex Data	Address	Command	Time	Date	Comment
DAP	FE90	Bcast	DIRECT ARC POWER 144 (5.0%)	17:52:15	12.12.2005	
DAP	FEFE	Bcast	DIRECT ARC POWER 254 (100%)	17:52:16	12.12.2005	
DAP	FE90	Bcast	DIRECT ARC POWER 144 (5.0%)	17:52:17	12.12.2005	
IAP	FF00	Bcast	OFF	17:52:19	12.12.2005	
DAP	FEFE	Bcast	DIRECT ARC POWER 254 (100%)	17:52:20	12.12.2005	
Special	A3FF	*	DATA TRANSFER REGISTER (DTR)	17:52:22	12.12.2005	

If the DALIBusServer is closed while an Application uses the DLL, the result will be an unpredictable behavior of the application. A warning message will be displayed in this case.

How the DALIBusServer.exe and DALIBusAccess.dll are located

To avoid placing the DALIBusServer.exe and DALIBusAcess.dll in the Windows System directory, the installation program writes the registry key

HKEY_CURRENT_USER\Software\Lunatone\DALIBusServer\DALIBusAccessDLLPath

with the location where the DALIBusServer.exe and DALIBusAccess.dll are installed. The provided C++ wrapper object uses this information to locate the DLL – see the source code for details.

If the DALIBusServer.exe, DALIBusAccess.dll and the application are all in the same directory this method is not necessary, so simply copying the 3 files to a directory will work too.

Alternatively you may want to copy the DLL and DaliBusServer.exe into the Windows system directory.

Directly Accessing The DLL Interface

The DLL provides the following interface:

GetDLLVersion()	get DLL or BusServer version number
OpenPort()	open a specific DALI interface device (DALI USB, DALI SCI...)
ClosePort()	close the DALI interface device after use
SendBusData()	send DALI frames on the bus, receive answer
GetDeviceFeatures()	specific to some interface devices
SetDeviceFeatures()	specific to some interface devices
SendGenericData()	specific to some interface devices

This chapter describes the DLL interface functions in detail.
For a quick start look at the examples at the end of this chapter.

extern "C" int WINAPI GetDLLVersion (int iIndex)

Get the DLL or DALIBusServer version number.

iIndex	0	Get DLL version (format x.y).
	1	Get DLL version (format x.y.z).
	2	Get DALIBusServer version (format x.y.z).

Return Value:

Version	the Version is coded like this: format x.y: (x * 10) + y, e.g. Version 2.1 = 21 format x.y.z: (x * 100) + (y * 10) + z, e.g. Version 2.1.1 = 211

extern "C" HDALI WINAPI OpenPort (DWORD **dwPort*, int **iDevice*, HWND *hWnd*)

dwPort

For the devices connected to the serial port (DD_NORMAL ... DD_DALISCI2) this is the COM port number 1..127 for Win98 and 1..255 for winNT and XP.

For USB devices (DD_DALIUSB) *dwPort* selects which USB device to use (see table below). If only one DALI USB is connected simply pass USB_PORT_BASE to *dwPort*. This will open the first DALI USB that was found.

dwPort	
0	reserved
0x01...0xFF	serial port number (COM1 ... COM255)
0x100...0x3FF	reserved
0x400...0x4FF	defined as USB_PORT_BASE ... USB_PORT_ITERATE_MAX in DALIBusAccessAPI.h. Use this to iterate through all USB devices connected to the PC.
0x500...0x00FFFFFF	directly open USB device with serial No. 0.... 16776715. *dwPort*= USB_PORT_DEV_BASE+ SerialNumber

iDevice

Opens the device *iDevice* using the given port *dwPort* if not already open. If *iDevice*= DD_TESTPORT, the function just tests the availability of the given port.

iDevice	DD_TESTPORT (0)	just test the availability of port *dwPort*; does not actually open the port.
	DD_AUTODEVICE (1)	test the availability of *dwPort* and check for a connected device – opens the port and returns the device ID if successful.
	DD_DALISCI (8)	try to locate a DALI SCI on COM [*dwPort*]
	DD_ DALISCI2 (9)	try to locate a DALI SCI2 on COM [*dwPort*]
	DD_DALIUSB (11)	try to locate a DALI SCI USB on the USB bus *dwPort* should then be >= USB_PORT_BASE

to be continued – see DALIAccessAPI.h for a complete list of devices

Multiple applications (processes) may call OpenPort(). An internal process counter is incremented for every new process that calls OpenPort(). Calls to ClosePort() decrement this process counter – **therefore ist is essential that calls to OpenPort() and ClosePort() are always paired.**
A call to ClosePort() will not actually close the port until the last process has called ClosePort().

Calling OpenPort() with *iDevice* set to DD_TESTPORT may be useful to create a list of available ports:

COMx

To test the availability of a serial COM port *dwPort* has to be in the range [1...255].

USB

If you want to test for the availability of a USB DALI device set *dwPort* to USB_ID_BASE (declared as 1024 in DALIAccessAPI.h). and call OpenPort() with *iDevice* set to DD_TESTPORT. Repeat this, incrementing *dwPort* with every call, until an error is returned. The number of successful iterations equals the available devices on the USB port.

When OpenPort() returns the actual port number is copied into *dwPort*. Once the actual port number for a specific device is known is can always be opened using this value for *dwPort*. The returned value is: USB_PORT_DEV_BASE+ SerialNumber. This means that if a DALI USB was opened with *dwPort*= USB_PORT_BASE, after the function returns *dwPort* will hold USB_PORT_DEV_BASE+ SerialNumber or with other words: if the serial number in known you can always open this DALI USB by setting *dwPort* to USB_PORT_DEV_BASE+ SerialNumber.

In the case of *iDevice* is set to DD_TESTPORT the port is tested but not actually opened.
If *iDevice*= DD_AUTODEVICE, the function tries to open the given port *dwPort* and returns the connected bus interface device type, DD_PORTAVAIL if no such device was found or DD_PORTNOTAVAIL if the given port could not be opened.

hWnd

Received DALI data will be reported to your application via a notification message to the window *hWnd* if not NULL. *hWnd* specifies a Windows window handle of type HWND.
If the monitoring feature is not needed set hWnd to NULL.

Notifications are sent in the following form:

```
long SendMessage( hWnd, RWM_DALIDATA, wDataType, lData);
```

where `RWM_DALIDATA` is defined in DALIAccessAPI.h as follows:

```
     static UINT RWM_DALIDATA = RegisterWindowMessage("DALI_DATA-
           {610C388E-A776-4996-87EB-FC87513107D5}");
```

Using a registered window message ensures the uniqueness of the message ID.

`wDataType` may be one of the following values (defined in DALIBusAccessAPI.h):

DI_8BIT	lData holds 8bit DALI data, DALI_ERROR, DALI_TRASH or DALI_NO
DI_16BIT	lData holds 16bit DALI data
DI_24BIT	lData holds 24bit DALI data
DI_extDSI	lData holds 16bit extended DSI data
DI_DSI	lData holds 8bit DSI data

Return Value:

> 0 valid handle to a device. This handle must be used in subsequent calls to ClosePort(), SendBusData(), GetDeviceFeatures() and SetDeviceFeatures().

`*dwPort` receives the actual port identifier (COM port number for serial port devices, device serial number for USB and TCP/IP devices)

`*iDeviceType` receives the device identifier for the connected device

<= 0 error:

DD_PORTNOTAVAIL (-1)	the port *iPort* could not be opened
DD_PORTAVAIL (-2)	the port *iPort* could be opened, but there was no supported device connected
DD_NOSERVER (-3)	OpenPort() failed, because there is no DALIBusServer running or could not be started.

extern "C" int WINAPI ClosePort(HDALI hDeviceHandle)

Closes the port corresponding to *hDeviceHandle* (returned from OpenPort( )).
The port is not actually closed if there are other processes that still use the port. (see OpenPort())

Return Value:	**1**	the device and port has been actually closed
	0	the device is closed for this application, but other processes prevent closing the port because they still use this device.
	DALI_ERROR (-4)	either *hDeviceHandle* is not a valid handle (not returned by OpenPort(), the port has already been closed, or an internal error has occured.

extern "C" int SendBusData(HDALI hDeviceHandle, DWORD dwBusCommand, DWORD dwBusData, DWORD dwFlags, WORD wBusAddress, DWORD dwAdditionalData, DWORD dwAdditionalAddress)

This function implements the complete DALI functionality. All valid DALI commands are defined in DALICommands.h and may be passed as *dwBusCommand*. Some commands need additional data(e.g. DirectArcPowerControl) passed as *dwBusData*. See DALICommands.h for a short description. More detailed information is found in the “DALI draft”.

The function returns after the DALI command has been sent and a valid DALI answer has been received (if any). So a call to this function may take up to 40ms to complete. The function automatically blocks other processes from writing to the DALI bus simultaneously.

hDeviceHandle handle as returned by OpenPort().

wBusCommand CMD_xxx as defined in DALICommands.h

dwBusData Additional data needed for some CMD_xxx (see DALICommands.h and the DALI draft)

wFlags **ADDR_DEFAULT**, **ADDR_BROADCAST**: sends commands to all devices on the bus using DALI Broadcast. *wBusAddress* is not used in this case and should be set to zero.
ADDR_GROUP sends to a DALI Group 0..15 (*wBusAddress*= Group)
ADDR_ADDR sends to a DALI Short Address 0..63 (*wBusAddress*= Address)

additional Flags that may be OR'ed to the above addressing flags:
BDF_NOAUTODTR By default the DTR is set with the data byte given in *dwBusData* for commands that use data from the DTR. You can

overwrite this default behaviour when setting this flag. Your application is then responsible for a correct setting of the DTR register.

BDF_NOCMDREPEAT	If this flag is set, the automatic repetition of commands (as defined in the DALI draft) is suppressed. Your application is then responsible for a command repetition within the specified 100ms.
BDF_PRIORITY_X	Some DALI devices support priority controlled sending of packets. Set the packet priority here (valid range depends on device used, basically 0-4, normal priority is 2)
BDF_REPEATTIME	unused, just for compatibility
BDF_SETDTR	force the setting of DTR. The DTR value is passed in the low byte of dwAdditionalData.
BDF_SETDEVTYPE	force the setting of the Dali device type before the command. The device type is passed in dwAdditionalData.
BDF_SENDTWICE	send the command twice.
BDF_ACTUALLEVELDTR	send 'store actual level to DTR' before the command.

wBusAddress

Short address or group address as specified by the flags in *wFlags*.

dwAdditionalData

for commands < 900 this is used as the device type in combination with BDF_SETDEVTYPE.
For the "raw" commands 950...990 this has the following meaning:
0x0000TTDD: TT= device type, DD= DTR value.
Only used in combination with BDF_SETDTR or BDF_SETDEVTYPE.

dwAdditionalAddress

unused, just for compatibility

Return value:

DALI_ERROR (-4)	general error (invalid handle, out of memory,...)
DALI_TRASH (-3)	DALI error (not a valid DALI frame – may occur if 2 or more DALI- devices answer simultaneously)
DALI_NO (-2)	DALI answer NO
DALI_OK (-1)	command sent OK, but this command does not have an DALI answer
>= 0	valid DALI answer byte (e.g. answer to a query command)

Sample Code

Examples on how to open a bus interface device:

- for a code sample on how to dynamically load the DLL see the C++ wrapper object DALIdll.cpp. Alternatively you may statically link to the DLL by including DaliBusAccess.lib to your project.

- test for the availability and version number of the DLL:

```
int iVersion= GetDLLVersion( 1);
// now do something with iVersion; e.g. display it

int iBusServerVersion= GetDLLVersion( 2);
// now do something with iBusServerVersion; e.g. display it
```

- open the DALI interface device connected to a serial port or USB port:

```
DWORD dwPort= USB_PORT_BASE;       // e.g.: open a DALI USB
int iDevice= DD_DALIUSB;

HDALI hDev= OpenPort( &dwPort, &iDevice, NULL);
if((hDev > 0) && (iDevice == DD_DALIUSB))
 {
   // successfully opened a DALI USB
   // now ready to use the port...
   SendBusData(hDev, CMD_OFF, 0, 0, 0, 0, 0)
 }
```

- If OpenPort() returned a valid device handle (positive number) it is now possible to send out DALI commands on the DALI bus by using SendBusData(). See examples on the next page.

- When you finished using the interface device (or when your application terminates) close it using:

```
ClosePort( hDev);
```

Example on how to open a DALI SCI on the serial port COM2:

```
DWORD dwPort= 2;        // use COM port 2 as an example
int iDevice= DD_AUTODEVICE;

HDALI hDev= m_DaliBus.OpenPort(&dwPort, &iDevice, NULL);
if( hDev > 0)
 {
   // iDevice now holds the device type of the connected
   // DALI interface device
   //  - DD_DALISCI
   //  - DD_DALISCI2 and so on
   if( iDevice == DD_DALISCI)
      SendBusData(hDev, CMD_OFF, 0, 0, 0, 0, 0)
 }
```

Handling multiple DALI USB connected to the PC

Every DALI USB has an unique serial number. This serial number is returned by a call to the dll's OpenPort() function and may be used to open exactly this DALI USB in subsequent calls to OpenPort().

1) If only one DALI USB is connected to the PC just pass `USB_PORT_BASE` as dwPort – this simply opens the first DALI USB in the device list:

```
DWORD dwPort= USB_PORT_BASE;
int iDevice= DD_DALIUSB;

hDev= OpenPort(&dwPort, &iDevice, NULL);
```

2) If the DALI USB's serial number is unknown, a list of all connected DALI USB can be generated by calling OpenPort() in a loop until the device is not DD_DALIUSB. For every iteration in the loop the parameter dwPort has to be incremented.
Note that `USB_PORT_BASE (0x400)` has to be used as the base for the iteration. After the serial number is known use `USB_PORT_DEV_BASE (0x500)` as shown in step 3).

```
#define MAX_DEVS     20       // just an example (127 devices max)
DWORD dwSerial[MAX_DEVS];
DWORD dwPort;
int iDevice= DD_DALIUSB;
HDALI hDev= 1;

for( int i= 0; (i < MAX_DEVS) && (iDevice == DD_DALIUSB); i++)
{
   iDevice= DD_TESTPORT;
   dwPort= USB_PORT_BASE+ i;
   OpenPort(&dwPort, &iDevice, NULL);

   dwSerial[i]= dwPort- USB_PORT_DEV_BASE;
   // since we have set iDevice to DD_TESTPORT the port has not
   // actually opened, so there is no need to close it using
   // ClosePort(). After we have the list, we actually
   // open the wanted device in the next step.
}
```

3) If the serial number is known, open the DALI USB like this:

```
DWORD dwPort= USB_PORT_DEV_BASE+ SerialNumber;
 // open a DALI USB
 // with this serial
 // number
int iDevice= DD_DALIUSB;

hDev= OpenPort(&dwPort, &iDevice, NULL);
if( (hDev > 0) && (iDevice == DD_DALIUSB))
 {
   // successfully opened a DALI USB
}
```

`USB_PORT_DEV_BASE` is defined in DaliBusAccessAPI.h (0x500).

Examples on how to use SendBusData( ):

1) send Broadcast UP

```
SendBusData( hDev, CMD_UP, 0, ADDR_BROADCAST, 0, 0, 0)    or:
SendBusData( hDev, CMD_UP, 0, 0, 0, 0, 0)
```

2) send RECALL_MAX to address 5

```
SendBusData( hDev, CMD_RECALL_MAX, 0, ADDR_ADDR, 5, 0, 0)
```

3) send direct arc power 180 to group 3

```
SendBusData( hDev, CMD_DIRECT_LIGHT_CONTROL, 180, ADDR_GROUP, 3, 0, 0)
```

4) send Broadcast light level 70%

```
SendBusData( hDev, CMD_DIRECT_LIGHT_CONTROL_PERCENT, 7000, 0, 0, 0, 0)
```

5) Broadcast store DTR= 200 as scene 3

```
SendBusData( hDev, CMD_STORE_SCENE3, 200, 0, 0, 0, 0)
```

6) Broadcast store actual level as scene 3

```
SendBusData( hDev, CMD_STORE_SCENE3, 0,
             BDF_NOAUTODTR|BDF_ACTUALLEVELDTR, 0, 0, 0)
```

7) Query control gear at address 3

```
iAnswer= SendBusData( hDev, CMD_QUERY_CONTROL_GEAR, 0, ADDR_ADDR, 3,
                      0, 0)
```

8) send INITIALISE

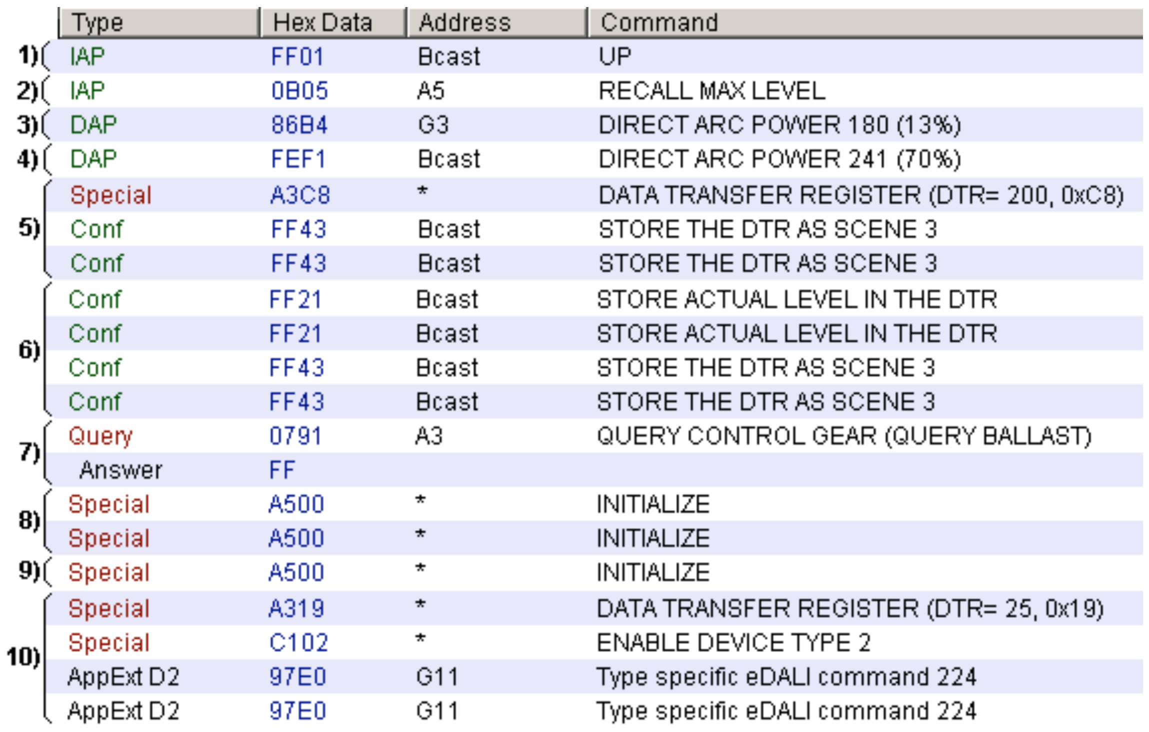
```
SendBusData( hDev, CMD_INITIALISE, 0, 0, 0, 0, 0)
```

9) send INITIALISE (suppress automatic repetition)

```
SendBusData( hDev, CMD_INITIALISE, 0, BDF_NOCMDREPEAT, 0, 0, 0)
```

10) send application extended command 224 to device type 2, group 11, set DTR with 25, repeat command

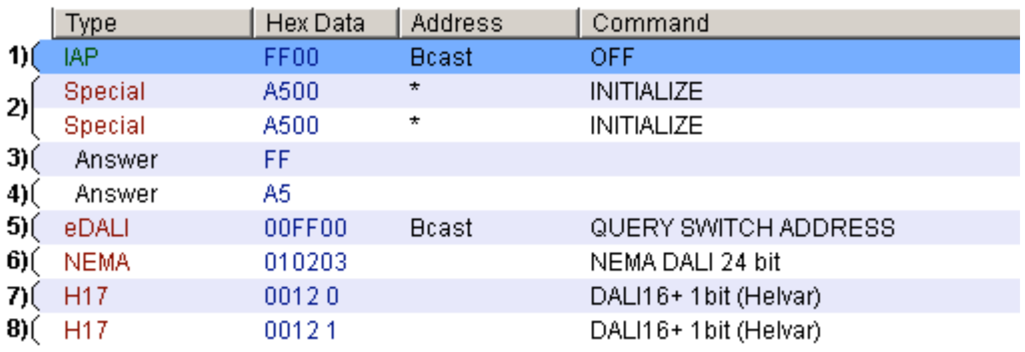
```
SendBusData( hDev, 224, 25,
             ADDR_GROUP|BDF_SETDEVTYPE|BDF_SETDTR|BDF_REPEAT,
             11, 2, 0)
```



	Type	Hex Data	Address	Command
1)	IAP	FF01	Bcast	UP
2)	IAP	0B05	A5	RECALL MAX LEVEL
3)	DAP	86B4	G3	DIRECT ARC POWER 180 (13%)
4)	DAP	FEF1	Bcast	DIRECT ARC POWER 241 (70%)
5)	Special	A3C8	*	DATA TRANSFER REGISTER (DTR= 200, 0xC8)
	Conf	FF43	Bcast	STORE THE DTR AS SCENE 3
	Conf	FF43	Bcast	STORE THE DTR AS SCENE 3
6)	Conf	FF21	Bcast	STORE ACTUAL LEVEL IN THE DTR
	Conf	FF21	Bcast	STORE ACTUAL LEVEL IN THE DTR
	Conf	FF43	Bcast	STORE THE DTR AS SCENE 3
	Conf	FF43	Bcast	STORE THE DTR AS SCENE 3
7)	Query	0791	A3	QUERY CONTROL GEAR (QUERY BALLAST)
	Answer	FF		
8)	Special	A500	*	INITIALIZE
	Special	A500	*	INITIALIZE
9)	Special	A500	*	INITIALIZE
10)	Special	A319	*	DATA TRANSFER REGISTER (DTR= 25, 0x19)
	Special	C102	*	ENABLE DEVICE TYPE 2
	AppExt D2	97E0	G11	Type specific eDALI command 224
	AppExt D2	97E0	G11	Type specific eDALI command 224

Send raw DALI frames with SendBusData( ):

1) send a 16bit DALI frame (e.g. 0xFF00 = Broadcast OFF)
```
    SendBusData( hDev, CMD_SEND_DALI16bit, 0xFF00, 0, 0, 0, 0)
```
2) send a 16bit DALI frame with repetition (e.g. 0xA500 = Initialise)
```
    SendBusData( hDev, CMD_SEND_DALI16bit, 0xA500, BDF_SENDTWICE, 0, 0, 0)
```
3) send a 8bit DALI frame (e.g. 0xFF = YES)
```
    SendBusData( hDev, CMD_SEND_DALI8bit, 0xFF, 0, 0, 0, 0)
```
4) send a 8bit DALI frame (e.g. 0xA5)
```
    SendBusData( hDev, CMD_SEND_DALI8bit, 0xA5, 0, 0, 0, 0)
```
5) send a 25bit Tridonic.Atco eDALI frame (e.g. 0x00FF00)
```
    SendBusData( hDev, CMD_SEND_eDALI, 0x00FF00, 0, 0, 0, 0)
```
6) send a 24bit NEMA DALI frame (e.g. 0x010203)
```
    SendBusData( hDev, CMD_SEND_DALI24bit, 0x010203, 0, 0, 0, 0)
```
7) send a 17bit Helvar DALI frame, 17th bit low (e.g. 0x0012)
```
    SendBusData( hDev, CMD_SEND_DALI17bit, 0x0012, 0, 0, 0, 0)
```
8) send a 17bit Helvar DALI frame, 17th bit high (e.g. 0x0012)
```
    SendBusData( hDev, CMD_SEND_DALI17bit, 0x010012, 0, 0, 0, 0)
```



	Type	Hex Data	Address	Command
1)	IAP	FF00	Bcast	OFF
2)	Special	A500	*	INITIALIZE
	Special	A500	*	INITIALIZE
3)	Answer	FF		
4)	Answer	A5		
5)	eDALI	00FF00	Bcast	QUERY SWITCH ADDRESS
6)	NEMA	010203		NEMA DALI 24 bit
7)	H17	0012 0		DALI16+ 1bit (Helvar)
8)	H17	0012 1		DALI16+ 1bit (Helvar)

The C++ DLL Wrapper Class

For a fast implementation of DALI functionality into your C++ application it’s probably the best to use the supplied CDALIdll wrapper object. It implements a thin wrapper over the DaliBusAccess.dll and provides an easy to use interface.

It also implements the DLL location method as described in the chapter "Software Structure".

The wrapper class provides the following member functions:

GetDLLVersion()	get DLL or BusServer version number
GetDLLVersionString()	get DLL or BusServer version number formatted as a string
GetInterfaceName()	get the interface name formatted as a string (e.g. DALI USB)
OpenPort()	open a specific DALI interface device (DALI USB, DALI SCI...)
ClosePort()	close the DALI interface device after use
TestPort()	find all connected DALI interface devices
IsOpen()	TRUE if the port has been opened
GetPort()	returns the device handle used for the DLL interface functions
GetDevType()	returns the device type (e.g. DD_DALIUSB)
SendDALIData()	send DALI frames on the bus, receive answer
SetDLLPath()	may be used to overwrite the default DLL path
SendGenericData()	specific to some interface devices

For more detailed information on the interface functions see the source code for the wrapper class (DALIdll.h, DALIdll.cpp) and the description on the DLL interface functions.
You will also find more information by a look at the sample code (DaliBusAccessTest.sln)

On the following pages you will find a quick start guide on how to use the C++ wrapper class.

C++ Quick Start Guide

include DALIdll.h in your source file, where the CDALIdll object should be implemented

```
#include "DALIdll.h"
```

- declare an object of type CDALIdll

```
CDALIdll   m_DALIBus;
```

- test for the availability and version number of the DLL:

```
CString Str;
if( m_DALIBus.GetDLLVersionString( &Str))
 {
   // display version string in Str
 }
```

- open the DALI interface device connected to a serial port or USB port by a call to the OpenPort() member function:

```
// for example: open a DALI USB
int iDevice;

iDevice= m_DaliBus.OpenPort(USB_PORT_BASE, DD_DALIUSB);
if( iDevice == DD_DALIUSB)
 {
   // successfully opened a DALI USB

   // now ready to use the port...
   m_DaliBus.SendDALIData(CMD_OFF)
 }
```

- If OpenPort() returned a valid device type (positive number) it is now possible to send out DALI commands on the DALI bus by using SendDALIData().

```
// e.g.: send OFF command to all ballasts:
 m_DaliBus.SendDALIData(CMD_OFF);

// recall scene 2 on ballast with address 20:
 m_DaliBus.SendDALIData(CMD_RECALL_SCENE, 2, ADDR_ADDR, 20);
```

- When you finished using the port (or when your application terminates) close it using:

```
m_DaliBus.ClosePort();
```

Example on how to open a DALI SCI on the serial port COM2:

```
DWORD dwPort= 2;        // use COM port 2 as an example

int iDevice= m_DaliBus.OpenPort(dwPort);
if( iDevice == DD_DALISCI)
 {
   // now ready to use the port...
   m_DaliBus.SendDALIData(CMD_OFF)
 }
```

Handling multiple DALI USB connected to the PC

Every DALI USB has an unique serial number. This serial number is returned by a call to the TestPort() or OpenPort() function and may be used to open exactly this DALI USB in subsequent calls to OpenPort().

1) If only one DALI USB is connected to the PC just pass `USB_PORT_BASE` as dwPort – this simply opens the first DALI USB in the device list:

```
int iDevice= m_DaliBus.OpenPort(USB_PORT_BASE, DD_DALIUSB);
```

2) If the DALI USB's serial number is unknown, a list of all connected DALI USB can be generated by calling TestPort() in a loop until the result is other than DD_DALIUSB. For every iteration in the loop the parameter dwPort has to be incremented.
Note that `USB_PORT_BASE (0x400)` has to be used as the base for the iteration. After the serial number is known use `USB_PORT_DEV_BASE (0x500)` as shown in step 3).

```
#define MAX_DEVS     20       // just an example - 127 devs max.
DWORD dwSerial[MAX_DEVS];
DWORD dwPort= USB_PORT_BASE;  // 0x400, defined in DaliBusAccessAPI.h
int iDevice= DD_DALIUSB;

for( int i= 0; (i < MAX_DEVS) && (iDevice == DD_DALIUSB); i++)
{
   iDevice= m_DaliBus.TestPort(dwPort+ i, &dwSerial[i]);
   // when using TestPort, the port has not actually
   // be opened so there is no need to close it using
   // ClosePort()
}
```

3) If the serial number is known, open the DALI USB like this:
(note that `USB_PORT_DEV_BASE (0x500)` has to be used as the base for dwPort)

```
DWORD dwPort= USB_PORT_DEV_BASE+ SerialNumber;
 // open a DALI USB
 // with this serial
 // number
int iDevice;

iDevice= m_DaliBus.OpenPort(dwPort, DD_DALIUSB);
if( iDevice == DD_DALIUSB)
 {
   // successfully opened a DALI USB
 }
```

`USB_PORT_DEV_BASE` is defined in DaliBusAccessAPI.h (0x500).

How to implement a DALI monitor

If you use Visual C++ and MFC you can easily implement a message handler for receiving DALI frames in a window of your application as follows:

in the header file declare:

```
....
//}}AFX_MSG
afx_msg LRESULT OnReceivedDALIData( UINT wParam, LONG lParam);
```

Note that the declaration is outside the //{{AFX_MSG ..... //}}AFX_MSG section, otherwise class wizard will delete the entry the next time it makes a change.

In the cpp source file implement the message handler:

```
....
//}}AFX_MSG_MAP
ON_REGISTERED_MESSAGE(RWM_DALIDATA, OnReceivedDALIData)
....

// example implementation:

LRESULT ClassName::OnReceivedDALIData( UINT wParam, LONG lParam)
{
      switch( wParam)
      {
              case DI_8BIT:
                      if( lParam >= 0)
                      {
                         printf (“DALI 8BIT: 0x%02x\n", (lParam & 0xFF));
                      }
                      else
                      {
                         switch( lParam)
                         {
                              case DALI_NO: printf (“DALI NO\n”); break;
                              case DALI_TRASH: printf (“TRASH\n”); break;
                              default: printf (“ERROR\n”); break;
                         }
                      }
                 break;
              case (char)DI_16BIT:
                      printf (“DALI 16BIT: 0x%04x\n", (lParam & 0xFFFF));
                 break;
              case (char)DI_24BIT:
                      printf (“DALI 24BIT: 0x%06x\n", (lParam & 0xFFFFFF));
                 break;
              case (char)DI_DSI:
                      printf (“DALI DSI: 0x%02x\n", (lParam & 0xFF));
                 break;
              case (char)DI_extDSI:
                      printf (“DALI extDSI: 0x%04x\n", (lParam & 0xFFFF));
                 break;
              default:
                      printf (“unknown data: 0x%08x\n", lParam);
                 break;
      }
      // return a value < 0 if no further processing is required
      // return [0...255] as a answer to this cmd (only for DI_16BIT, DI_24BIT)
  return -1;
}
```

Open the DALI USB and pass the window handle in the call to OpenPort():

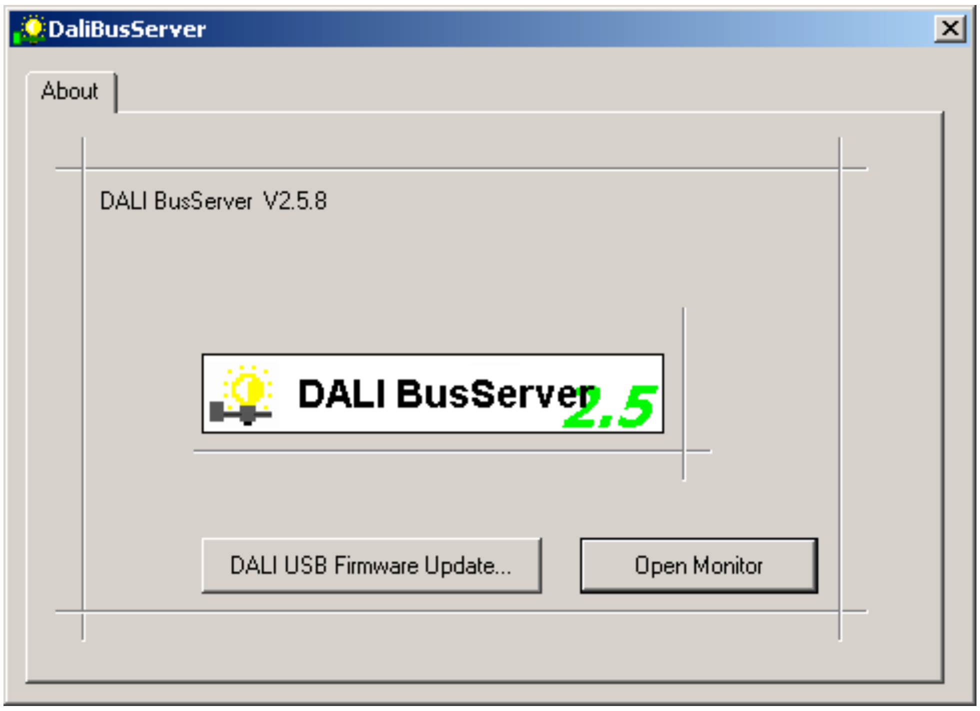
```
 iDevice= m_DaliBus.OpenPort (dwPort, DD_DALIUSB, m_hWnd);
```

Automatic Update for the DALI USB

When a DALI USB is opened by an application (call to OpenPort()) the firmware version is checked and an automatic update of the DALI USB is started if the version number does not match.
This update normally takes about 10..15 seconds to complete.

In rare cases the update could fail and leave the DALI USB in bootloader mode. In bootloader mode the LED's on the DALI USB are switched OFF after you connect it to the USB, in normal mode one of the two LED's is constantly ON, the other indicates DALI bus activity.

In case the BusServer detects a DALI USB in bootloader mode it displays the "DALI USB Firmware Update" button:



By clicking on this button you may force an update manually.