DLN-Series Interface Adapters. Programmer's Reference Manual.

DLN-Series Interface Adapters. Programmer's Reference Manual.

Table of Contents

List of Figures

List of Tables

Revision History

Introduction

Chapter 1. Communication with device

1.1. Messages

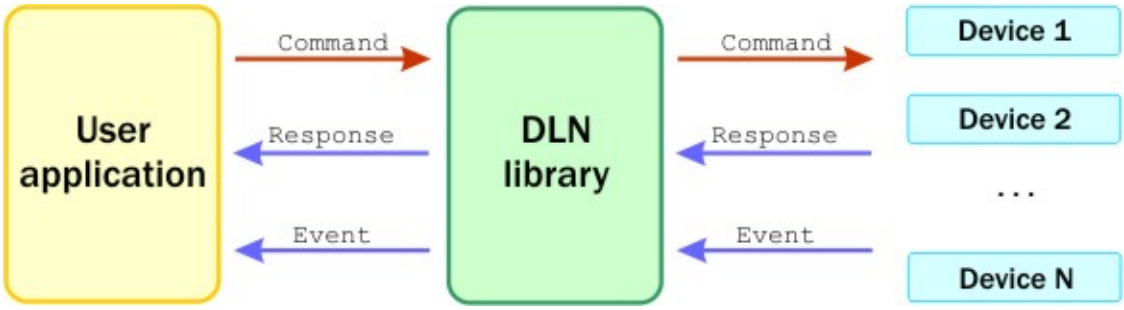
The communication with a device is performed by the use of messages. A message is a packet of data that is sent from the library to a device and vice versa. The DLN adapters utilize three types of messages:

- commands;
- responses;
- events.

Here is a short comparison to make things more logical and simple.

Table 1.1.

	Name	Sender	Recipient	Description
Command	DLN_DO_SOME_ACTION_CMD	User application	Device	Contains an instruction to a device
Response	DLN_DO_SOME_ACTION_RSP	Device	User application	Contains the result of the command execution
Event	DLN_SOMETHING_CHANGED_EV	Device	User application	Contains information about some changes that took place



Commands are sent from the user application to a device. They contain some instructions to the device. You may instruct the device to perform an action (e.g. to change voltage on a pin) or to configure the settings of the device. Each command has corresponding response.

A response is sent from the device to the user application after a command execution. A response always returns the result of the command execution. If the command was successfully executed, the response informs the user application about this. If it is impossible to complete the command, the response returns the error code. Some commands request specific data (e.g. the serial number of a device or the total number of connected adapters). In this case the response returns the requested data in addition to the result of the command execution.

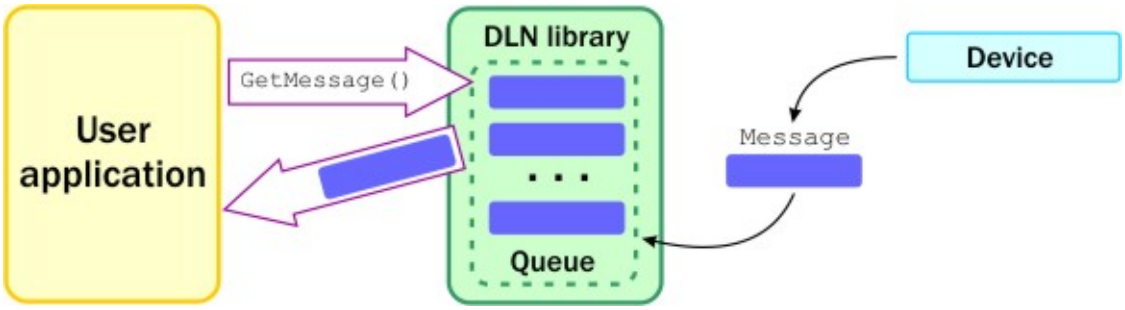
Events are sent from the device to the user application. They contain information about some changes that have taken place. A user can predefine the condition of an event generation. For example, an event may be generated when a new device is connected or when voltage changes on an input pin.

All the messages are transferred through the DLN library.

The messages (commands, responses and events) are delivered with the help of three functions:

- DlnSendMessage() - sends a specified message (an asynchronous command) to the device;
- DlnGetMessage() - retrieves messages (responses and events) sent by the device;
- DlnTransaction() - sends a synchronous command, waits for a response and returns the response details.

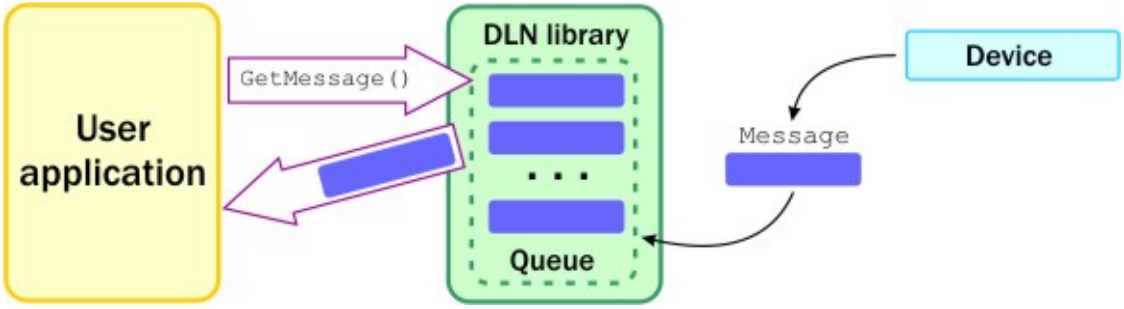
Messages sent by the device (responses and events) are pushed into the DLN library message queue. The user may call the DlnGetMessage() function to get the message from the queue.



The DlnGetMessage() function removes the message from the queue and passes the message details to the user application.

1.2. Notifications

The DLN library may notify the user application when new messages arrive from the device.



There are 4 different types of notifications:

- callback function;
- event object;
- window message;
- thread message.

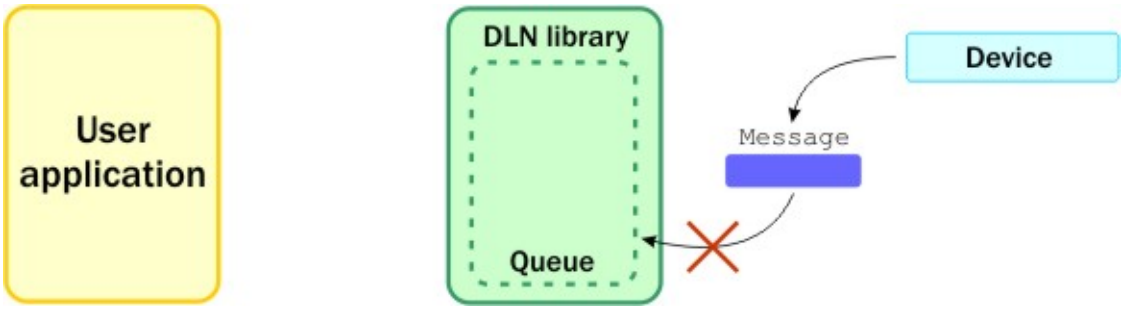
You can configure the same notification settings for all the messages. To do so, call the DlnRegisterNotification() function and specify `HDLN_ALL_DEVICES (0)` value as a handle. In this case the DLN library will notify the user application about messages from all devices.

The DLN library may notify the application about messages from a specific device. To configure such notification settings, call the DlnRegisterNotification() function and specify the handle of the device. Streams (like devices) have their own handles. So, you may configure the notification settings for a specific stream as well.

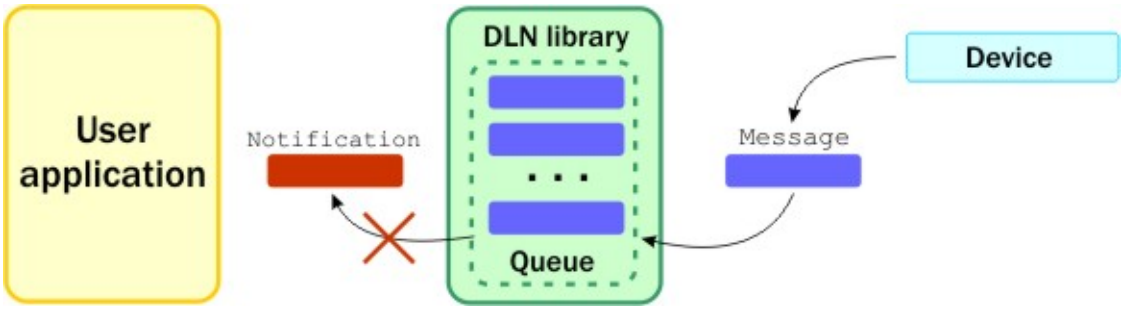
You can use the DlnRegisterNotification() function several times, specifying various notification settings for different devices (streams). For example, if you have 4 devices, you may register certain notification settings for one device and different settings for other devices. When a device sends a message, the library checks the notification settings for current device. If the library finds such settings, the notification is generated. If there are no settings for current device, the library checks the notification settings for all devices. If there are no such settings either, the notification isn't generated and the message isn't pushed into the queue.

Sometimes it is useful when messages aren't pushed into the queue. It is most convenient for those who use only synchronous communication. During the synchronous communication the application doesn't call the DlnGetMessage() function. Thus the messages aren't removed from the queue. It leads to memory

leak and eventually to memory overflow. If you don't want messages to be enqueued, you shouldn't register any notification settings.



If you want the messages to be enqueued without notification, do the following. Call the DlnRegisterNotification() function and specify `DLN_NOTIFICATION_TYPE_NO_NOTIFICATION` value as the notification type. In this case the messages will be pushed into the queue without notification to the user application. The messages can be obtained with the help of the DlnGetMessage() function.



To unregister the notification settings call the DlnUnregisterNotification() function.

You can register and unregister notification settings with the help of commands instead of functions. Send DLN_REGISTER_NOTIFICATION command to register the settings. To unregister the notification settings send the DLN_UNREGISTER_NOTIFICATION command.

1.3. Device Opening & Identification

Using the DLN library a user can operate several devices at once. The number of available devices can be retrieved using the DlnLedGetCount() function. In case a user works with several servers, all devices connected to those servers become available.

A device can be opened using the DlnOpenDevice() function. In case there is a single device connected to a host, we can open it, specifying 0 as the first parameter.

In case there are several devices connected to a host, this function distinguishes them using a number, randomly assigned when a device is connected. Such a number cannot be used to open a specific device.

A user can use call the DlnOpenDeviceBySn() or DlnOpenDeviceById() function to open a specific device.

Serial Number

A serial number (SN) is factory assigned and cannot be changed. The SN can be used to open a device using the DlnOpenDeviceBySn() function. Once the device is open, a user can retrieve the device SN using the DlnGetDeviceSn() function.

Identification number

An identification (ID) number can be assigned to each DLN-series adapter by a user. In case there are several devices connected to a host, their ID numbers can be used for easy identification.

An ID number can be assigned using a DlnSetDeviceId() function. Once the ID number is set, it can be used to open the device using the DlnOpenDeviceById() function. The ID number of the opened device can be retrieved using the DlnGetDeviceId() function.

The Device ID application shows how to set or retrieve the ID number of the device.

1.4. Device Pin Configuration

1.5. Structures and Types

1.5.1. DLN_MSG_HEADER

The message header is the first field of each message, sent from a host to a device or vice versa. It is used to identify and route the message correctly.

The DLN_MSG_HEADER structure is used to describe a message (command, response or event).

```
typedef struct
{
   uint16_t size;
   uint16_t msgId;
   uint16_t echoCounter;
   HDLN handle;
} __PACKED_ATTR DLN_MSG_HEADER;
```

Members:

size
Defines the size of the message.

msgId
The message ID, defining the message.

echoCounter
If case of a command/response pair the echo counter is used to establish a one-one link between them. In case the message is an event, this is a freerunning counter. It shows the number of events sent by the device. The counter is reset when its value reaches 0xFFFF (65536).

handle
A handle to the DLN-series adapter. In case of an event, it contains the handle of the device, that generated the event. For a command, the handle is defined by a user in order to route it to a specific device. The HDLN_ALL_DEVICES constant can be used to send the command to all devices.

1.5.2. DLN_PIN_CFG

The DLN_PIN_CFG structure is used to store the configuration of a single DLN-series adapter pin.

```
typedef struct
{
   DLN_MODULE module;
   DLN_PIN_ROLE role;
} __PACKED_ATTR DLN_PIN_CFG;
```

Members:

module
A module, to which the pin is connected. The following values are available:

- DLN_MODULE(0x00) or DLN_MODULE_GENERIC - Generic module;
- DLN_MODULE(0x01) or DLN_MODULE_GPIO - GPIO module;

- `DLN_MODULE(0x02)` or `DLN_MODULE_SPI` - SPI Module;
- `DLN_MODULE(0x03)` or `DLN_MODULE_I2C` - I2C module;
- `DLN_MODULE(0x04)` or `DLN_MODULE_I2S` - I2S module;
- `DLN_MODULE(0x05)` or `DLN_MODULE_PWM` - PWM module;
- `DLN_MODULE(0x06)` or `DLN_MODULE_FREQ` - Frequency counter module;
- `DLN_MODULE(0x07)` or `DLN_MODULE_ADC` - ADC module;
- `DLN_MODULE(0x08)` or `DLN_MODULE_LED` - LED module;

role
A role performed by the pin.

1.5.3. DLN_VERSION

The `DLN_VERSION` structure is used to store the DLN-series adapter and software version data.

See. DlnGetVersion() function and Get version demo application.

```
typedef struct
{
   uint32_t hardwareType;
   uint32_t hardwareVersion;
   uint32_t firmwareVersion;
   uint32_t serverVersion;
   uint32_t libraryVersion;
} __PACKED_ATTR DLN_VERSION;
```

Members:

hardwareType
A type of the device (e.g. DLN-4).

hardwareVersion
A version of the hardware, used in he device.

firmwareVersion
A version of the firmware, installed in the device.

serverVersion
A version of the DLN server.

libraryVersion
A version of the DLN-library.

1.5.4. DLN_NOTIFICATION

Notification settings are passed to DlnRegisterNotification() function and DLN_REGISTER_NOTIFICATION_CMD command as DLN_NOTIFICATION structure:

```
typedef struct _DLN_NOTIFICATION
{
        uint16_t type.
```

```
        union
        {
            PDLN_CALLBACK callback;
#ifdef WIN32
            HANDLE event;
            struct
            {
                HWND handle;
                UINT message;
            } windowMessage;
            struct
            {
                DWORD thread;
                UINT message;
            } threadMessage;
#endif // WIN32
#ifdef DLN_QT_INTERFACE
            struct
            {
                const QObject* receiver;
                const char* method;

            } qtSignal;
#endif //  DLN_QT_INTERFACE
            uint8_t reserved[64];
        };

} __PACKED_ATTR DLN_NOTIFICATION;
```

1.5.5. Notification Types

There are 4 different types of notifications: callback function, event object, window message, and thread message. There are 5 different constants to specify the notification type:

```
#define DLN_NOTIFICATION_TYPE_NO_NOTIFICATION ((DLN_NOTIFICATION_TYPE)0x00)
#define DLN_NOTIFICATION_TYPE_CALLBACK 0x01
#define DLN_NOTIFICATION_TYPE_EVENT_OBJECT 0x02
#define DLN_NOTIFICATION_TYPE_WINDOW_MESSAGE 0x03
#define DLN_NOTIFICATION_TYPE_THREAD_MESSAGE 0x04
```

1.5.5.1. DLN_NOTIFICATION_TYPE_NO_NOTIFICATION (0x00)

In this case the library does not notify the application. However, the messages are pushed into the queue. You can call the DlnGetMessage() function to check whether the new messages have arrived and get the message details.

1.5.5.2. DLN_NOTIFICATION_TYPE_CALLBACK 0x01

The library calls the specified callback function to notify the application. You can specify the address of a callback function in callback member of the DLN_NOTIFICATION structure. The callback function is called in the context of the internal DLN library thread. If you use resources which are also available for other threads, apply the appropriate synchronization.

1.5.5.3. DLN_NOTIFICATION_TYPE_EVENT_OBJECT 0x02

The library uses an event object to notify the application. Specify valid handle to the event object in event member of the DLN_NOTIFICATION structure. As there is no possibility to pass parameters, handle to the

device that generated the event is not sent. Specify `HDLN_ALL_DEVICES (0)` constant as the device handle when you call the DlnGetMessage() function.

1.5.5.4. DLN_NOTIFICATION_TYPE_WINDOW_MESSAGE 0x03

The library sends a window message to notify the application. Specify valid window handle in window message member of the DLN_NOTIFICATION structure. Specify message ID (chosen during the initialization) in window message member of the DLN_NOTIFICATION structure. If a notification has been generated by a device, the device handle is returned in the window message. If a notification is sent by the library, the window message returns `HDLN_DLL_NOTIFICATION` constant instead of the device handle.

1.5.5.5. DLN_NOTIFICATION_TYPE_THREAD_MESSAGE 0x04

The library sends a thread message to notify the application. Specify valid thread identifier in thread message member of the DLN_NOTIFICATION structure. Specify message ID (chosen during the initialization) in thread message member of the DLN_NOTIFICATION structure. If a notification has been generated by a device, the device handle is returned in the thread message. If a notification is sent by the library, the thread message returns `HDLN_DLL_NOTIFICATION` constant instead of device handle.

1.6. Functions

This section describes the generic functions. They are used to control and monitor the DLN-series adapter.

Actual control of a device is performed by use of commands and responses. Each function utilizes respective commands and responses. You can send such commands directly if necessary.

- DlnRegisterNotification() - registers the notification settings between user applications and the DLN library;
- DlnUnregisterNotification() - unregisters notification setting between user applications and the DLN library;
- DlnConnect() - establishes the connection to the DLN server;
- DlnDisconnect() - closes the connection to the specified DLN server;
- DlnDisconnectAll() - closes connections to all servers at once;
- DlnGetDeviceCount() - retrieves the total number of DLN-devices available;
- DlnOpenDevice() - opens the specified device;
- DlnOpenDeviceBySn() - opens a device, specified by its serial number;
- DlnOpenDeviceById() - opens a device, specified by its ID number;
- DlnCloseHandle() - closes the handle to an opened DLN-series adapter (stream);
- DlnCloseAllHandles() - closes handles to all opened DLN-series adapters and streams;
- DlnGetVersion() - retrieves the DLN-series adapter version data;
- DlnGetDeviceSn() - retrieves a device serial number;
- DlnSetDeviceId() - sets a new ID to the DLN-series device;
- DlnGetDeviceId() - retrieves the device ID number;
- DlnSendMessage() - sends a specified message to the DLN-series adapter;
- DlnGetMessage() - retrieves messages sent by the device;

- DlnTransaction() - sends a synchronous command and returns the response details.
- DlnGetPinCfg() - retrieves current configuration of the specified DLN-series adapter pin.

1.6.1. DlnRegisterNotification()

```
DLN_RESULT DlnRegisterNotification(
   HDLN handle,
   DLN_NOTIFICATION notification
);
```

The `DlnRegisterNotification()` function registers notification settings. See "Notifications" for details.

Parameters:

handle
A handle to the DLN-series adapter. You may specify either the handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value. In the latter case the notification setting will be applied for all the devices (streams).

notification
Defines the notification settings. The settings are passed as the DLN_NOTIFICATION structure.

This function is defined in the `dln_generic.h` file.

1.6.2. DlnUnregisterNotification()

```
DLN_RESULT DlnUnregisterNotification(
   HDLN handle
);
```

The `DlnUnregisterNotification()` function unregisters notification settings.

Parameters:

handle
A handle to the DLN-series adapter (stream). The notification settings for the device (stream) will be unregistered . You may specify either the handle to a specific device (stream) or the `HDLN_ALL_DEVICES`. In the latter case the notification setting will be unregistered for all the devices (streams).

This function is defined in the `dln_generic.h` file.

1.6.3. DlnConnect()

```
DLN_RESULT DlnConnect(
   const char* host,
   uint16_t port
);
```

The `DlnConnect()` function establishes the connection to the DLN server.

Parameters:

host
A server to establish the connection to. This value can be a URL (e.g. www.diolan.com), or an IP address (e.g. 127.0.0.1). In order to connect to a server, launched on the same computer, you can use the predefined "localhost" value as a host name.

port
A port number of the DLN server.

This function is defined in the `dln_generic.h` file.

1.6.4. DlnDisconnect()

```
DLN_RESULT DlnDisconnect(
   const char* host,
   uint16_t port
);
```

The `DlnDisconnect()` function closes the connection to the specified DLN server.

Parameters:

host
A server to close the connection to. This value can be a URL (e.g. www.diolan.com), or an IP address (e.g. 127.0.0.1). In order to close the connection to a server, launched on the same computer, you can use the predefined "localhost" value as a host name.

port
A port of the DLN server.

This function is defined in the `dln_generic.h` file.

1.6.5. DlnDisconnectAll()

```
DLN_RESULT DlnDisconnectAll(
);
```

The `DlnDisconnectAll()` function closes connections to all servers at once.

This function is defined in the `dln_generic.h` file.

1.6.6. DlnGetDeviceCount()

```
DLN_RESULT DlnGetDeviceCount(
   uint32_t* deviceCount
);
```

The `DlnGetDeviceCount()` function retrieves the total number of DLN-devices available. In case there are several servers connected, this function will return the total number of DLN adapters, connected to all servers.

Parameters:

deviceCount
A pointer to an unsigned 32-bit integer. This integer will be filled with the total number of available DLN-series adapters.

This function is defined in the `dln_generic.h` file.

1.6.7. DlnOpenDevice()

```
DLN_RESULT DlnOpenDevice(
   uint32_t deviceNumber,
   HDLN* deviceHandle
);
```

The `DlnOpenDevice()` function opens the specified device. This function uses an index number of the device. This number is randomly system-assigned to each connected device. It cannot be used to identify the device. If you need to open a specific device, use DlnOpenDeviceBySn() or DlnOpenDeviceById().

Parameters:

deviceNumber
A number of the device.

deviceHandle
A pointer to the variable that receives the device handle after the function execution.

This function is defined in the `dln_generic.h` file.

1.6.8. DlnOpenDeviceBySn()

```
DLN_RESULT DlnOpenDeviceBySn(
   uint32_t sn,
HDLN* deviceHandle
);
```

The `DlnOpenDeviceBySn()` function opens a specified device. The device is defined by its serial number.

A device serial number is factory-assigned and cannot be changed.

Parameters:

sn
A serial number of the DLN-series adapter.

deviceHandle
A pointer to the variable that receives the device handle after the function execution.

This function is defined in the `dln_generic.h` file.

1.6.9. DlnOpenDeviceById()

```
DLN_RESULT DlnOpenDeviceById(
   uint32_t id,
```

```
   HDLN* deviceHandle
);
```

The `DlnOpenDeviceById()` function opens a specified device. The device is defined by its ID number.

The device ID can be changed by a user using the DlnSetDeviceId() function.

Parameters:

id
An ID of the DLN-series adapter.

deviceHandle
A pointer to the variable that receives the device handle after the function execution.

This function is defined in the `dln_generic.h` file.

1.6.10. DlnCloseHandle()

```
DLN_RESULT DlnCloseHandle(
   HDLN handle
);
```

The `DlnCloseHandle()` function closes the handle to an opened DLN-series adapter (stream).

Parameters:

handle
A handle to the DLN-series adapter.

This function is defined in the `dln_generic.h` file.

1.6.11. DlnCloseAllHandles()

```
DLN_RESULT DlnCloseAllHandles(
);
```

The `DlnCloseAllHandles()` function closes handles to all opened DLN-series adapters and streams.

This function is defined in the `dln_generic.h` file.

1.6.12. DlnGetVersion()

```
DLN_RESULT DlnGetVersion(
   HDLN handle,
   DLN_VERSION* version
);
```

The `DlnGetVersion()` function retrieves the following data about the DLN-series adapter:

- **Hardware type** - the type of the device (e.g. DLN-4);
- **Hardware version** - the version of the hardware, used in the device;
- **Firmware version** - the version of the firmware, installed in the device;

- **Server version** - the version of the server;
- **Library version** - the version of the DLN-library.

Parameters:

handle
A handle to the DLN-series device.

version
A pointer to a DLN_VERSION structure that receives version information after the function execution.

This function is defined in the `dln_generic.h` file.

1.6.13. DlnGetDeviceSn()

```
DLN_RESULT DlnGetDeviceSn(
   HDLN handle,
   uint32_t* sn
);
```

The `DlnGetDeviceSn()` function retrieves the device serial number. A serial number is factory-assigned and cannot be changed.

Parameters:

handle
A handle to the DLN-series adapter.

sn
A pointer to the variable that receives the device serial number after the function execution.

This function is defined in the `dln_generic.h` file.

1.6.14. DlnSetDeviceId()

```
DLN_RESULT DlnSetDeviceId(
   HDLN handle,
   uint32_t id
);
```

The `DlnSetDeviceId()` function sets a new ID number to the DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

id
An ID number to be set.

This function is defined in the `dln_generic.h` file.

1.6.15. DlnGetDeviceId()

```
DLN_RESULT DlnGetDeviceId(
   HDLN handle,
   uint32_t* id
);
```

The `DlnGetDeviceId()` function retrieves the device ID number.

The device ID number can be changed by a user using the DlnSetDeviceId() function.

Parameters:

handle
A handle to the DLN-series adapter.

id
A pointer to an unsigned 32-bit integer. This integer will be filled with the ID number after the function execution.

This function is defined in the `dln_generic.h` file.

1.6.16. DlnSendMessage()

```
DLN_RESULT DlnSendMessage(
   void* message
);
```

The `DlnSendMessage()` function sends a specified message (an asynchronous command) to the device.

Parameters:

message
A pointer to a variable that contains a message to be sent.

This function is defined in the `dln.h` file.

1.6.17. DlnGetMessage()

```
DLN_RESULT DlnGetMessage(
   HDLN handle,
   void* messageBuffer,
   uint16_t messageSize
);
```

The `DlnGetMessage()` function retrieves a message (response or event) sent by the device.

Parameters:

handle
A handle to the DLN-series adapter.

messageBuffer
A pointer to the buffer that receives the message information.

messageSize
The maximum number of bytes to be retrieved.

This function is defined in the `dln.h` file.

1.6.18. DlnTransaction()

```
DLN_RESULT DlnTransaction(
   void* command,
   void* responseBuffer,
   uint16_t responseBufferSize
);
```

The `DlnTransaction()` function sends a synchronous command, waits for a response and returns the response details.

Parameters:

command
A pointer to a variable that contains a command to be sent.

responseBuffer
A pointer to the buffer that receives the response information.

responseBufferSize
The maximum number of bytes to be retrieved.

This function is defined in the `dln.h` file.

1.6.19. DlnGetPinCfg()

```
DLN_RESULT DlnGetPinCfg(
    HDLN handle,
    uint16_t pin,
    DLN_PIN_CFG* cfg
);
```

The `DlnGetPinCfg()` function retrieves current configuration of the specified DLN-series adapter pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A pin to get the configuration from.

config
A pointer to the DLN_PIN_CONFIG structure which will be filled with the configuration after function execution.

This function is defined in the `dln_generic.h` file.

1.7. Commands and Responses

1.7.1. DLN_REGISTER_NOTIFICATION

DLN_REGISTER_NOTIFICATION Command

Go to response

The **DLN_REGISTER_NOTIFICATION** command registers notification settings.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_NOTIFICATION notification;
} __PACKED_ATTR DLN_REGISTER_NOTIFICATION_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_REGISTER_NOTIFICATION_CMD structure.
- **msgId** - Defines the message. For the DLN_REGISTER_NOTIFICATION command, it must be set to `0x0000`. You can use the `DLN_MSG_ID_REGISTER_NOTIFICATION` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

notification
Defines the notification settings. See DLN_NOTIFICATION structure for additional info.

DLN_REGISTER_NOTIFICATION Response

Go to command

The adapter sends the **DLN_REGISTER_NOTIFICATION** response after the command execution. The response will notify you whether the settings were successfully registered.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_REGISTER_NOTIFICATION_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_REGISTER_NOTIFICATION_RSP structure.
- **msgId** - Defines the message. For the DLN_REGISTER_NOTIFICATION response it is set to `0x0000`. The `DLN_MSG_ID_REGISTER_NOTIFICATION` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.

- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

1.7.2. DLN_UNREGISTER_NOTIFICATION

DLN_UNREGISTER_NOTIFICATION Command

Go to response

The **DLN_UNREGISTER_NOTIFICATION** command unregisters notification settings.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_UNREGISTER_NOTIFICATION_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_UNREGISTER_NOTIFICATION_CMD structure.
- **msgId** - Defines the message. For the DLN_UNREGISTER_NOTIFICATION command it must be set to `0x0001`. You can use the `DLN_MSG_ID_UNREGISTER_NOTIFICATION` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

DLN_UNREGISTER_NOTIFICATION Response

Go to command

The adapter sends the **DLN_UNREGISTER_NOTIFICATION** response after the command execution. The response will notify you whether the settings were successfully unregistered.

```
typedef struct
{
    DLN_MSG_HEADER header;
    DLN_RESULT result;
} __PACKED_ATTR DLN_UNREGISTER_NOTIFICATION_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_UNREGISTER_NOTIFICATION_RSP structure.
- **msgId** - Defines the message. For the DLN_UNREGISTER_NOTIFICATION response it is set to `0x0001`. The `DLN_MSG_ID_UNREGISTER_NOTIFICATION` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

1.7.3. DLN_CONNECT

DLN_CONNECT Command

Go to response

The **DLN_CONNECT** command establishes the connection to the DLN server.

```
typedef struct
{
   DLN_MSG_HEADER header;
   char host[DLN_MAX_HOST_LENGTH+1];
   uint16_t port;
} __PACKED_ATTR DLN_CONNECT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_CONNECT_CMD structure.
- **msgId** - Defines the message. For the DLN_CONNECT command it must be set to `0x0010`. You can use the `DLN_MSG_ID_CONNECT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. For the DLN_CONNECT command the `HDLN_ALL_DEVICES` value must be used.

host
A server to establish the connection to. This value can be a URL (e.g. www.diolan.com), or an IP address (e.g. 127.0.0.1). In order to connect to a server, launched on the same computer, you can use the predefined "localhost" value as a host name.

port
A port number of the DLN server.

DLN_CONNECT Response

Go to command

The adapter sends the **DLN_CONNECT** response after the command execution. The `result` field informs a user if the connection has been successfully established.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_CONNECT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_CONNECT_RSP structure.
- **msgId** - Defines the message. For the DLN_CONNECT response it is set to `0x0010`. The `DLN_MSG_ID_CONNECT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. For the DLN_CONNECT response the `HDLN_ALL_DEVICES` value is used.

result
Contains the result of the command execution. The following values are available:

1.7.4. DLN_DISCONNECT

DLN_DISCONNECT Command

Go to response

The **DLN_DISCONNECT** command closes the connection to the specified DLN server.

```
typedef struct
{
   DLN_MSG_HEADER header;
   char host[DLN_MAX_HOST_LENGTH+1];
   uint16_t port;
} __PACKED_ATTR DLN_DISCONNECT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_DISCONNECT_CMD structure.
- **msgId** - Defines the message. For the DLN_DISCONNECT command it must be set to `0x0011`. You can use the `DLN_MSG_ID_DISCONNECT` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. For the DLN_DISCONNECT command the `HDLN_ALL_DEVICES` value must be used.

host
A server to close the connection to. This value can be a URL (e.g. www.diolan.com), or an IP address (e.g. 127.0.0.1). In order to close the connection to a server, launched on the same computer, you can use the predefined "localhost" value as a host name.

port
A port number of the DLN server.

DLN_DISCONNECT Response

The adapter sends the **DLN_DISCONNECT** response after the command execution. The `result` field informs a user if the connection has been successfully closed.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_DISCONNECT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_DISCONNECT_RSP structure.
- **msgId** - Defines the message. For the DLN_DISCONNECT response it is set to `0x0011`. The `DLN_MSG_ID_DISCONNECT` constant can be used.
- **echoCounter** -Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. For the DLN_DISCONNECT response the `HDLN_ALL_DEVICES` value is used.

result
Contains the result of the command execution. The following values are available:

1.7.5. DLN_DISCONNECT_ALL

DISCONNECT_ALL Command

Go to response

The **DLN_DISCONNECT_ALL** command closes connections to all servers at once.

```
typedef struct
{
DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_DISCONNECT_ALL_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_DISCONNECT_ALL_CMD structure.
- **msgId** - Defines the message. For the DLN_DISCONNECT_ALL command it must be set to `0x0012`. You can use the `DLN_MSG_ID_DISCONNECT_ALL` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. For the DLN_DISCONNECT_ALL command the `HDLN_ALL_DEVICES` value must be used.

DISCONNECT_ALL Response

Go to command

The adapter sends the **DLN_DISCONNECT** response after the command execution. The `result` field informs a user if all the connections were successfully closed.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_DISCONNECT_ALL_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_DISCONNECT_ALL_RSP structure.
- **msgId** - Defines the message. For the DLN_DISCONNECT_ALL response it is set to `0x0012`. The `DLN_MSG_ID_DISCONNECT_ALL` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. For the DLN_DISCONNECT_ALL response the `HDLN_ALL_DEVICES` value is used.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - connections to all servers were successfully closed;
- `DLN_RES_NOT_CONNECTED` - no connections were present during the command execution.

1.7.6. DLN_GET_DEVICE_COUNT

GET_DEVICE_COUNT Command

Go to response

The **DLN_GET_DEVICE_COUNT** command retrieves the total number of DLN-devices available.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t filter;
   uint32_t hardwareType;
   uint32_t sn;
   uint32_t id;
} __PACKED_ATTR DLN_GET_DEVICE_COUNT_CMD;
```

Parameters:

header

Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GET_DEVICE_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_GET_DEVICE_COUNT command it must be set to `0x0020`. You can use the `DLN_MSG_ID_GET_DEVICE_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. For the DLN_GET_DEVICE_COUNT command the `HDLN_ALL_DEVICES` value must be used.

filter

A bit field that defines how the device will be identified for counting. Each bit corresponds to a DLN-series adapter identification, namely:

Bit	Identification
0	Hardware type
1	Device serial number
2	Device ID number

Only the identifications with corresponding mask bits set to 1 will be taken into account. The identifications with mask bits set to 0 will be ignored.

A user can set several mask bits to 1. This way only devices with all identifications matching the specified ones will be counted.

For example, if we need to count the devices with the specific hardware type and ID number, we should set the `filter` parameter as follows:

0	1	0

You can use the following constants:

- `DLN_DEVICE_FILTER_NUMBER` - the `number` parameter;

- DLN_DEVICE_FILTER_HW_TYPE - the hardwareType parameter;
- DLN_DEVICE_FILTER_SN - the sn parameter;
- DLN_DEVICE_FILTER_ID - the id parameter;

hardwareType
A type of the DLN-series adapter.

sn
A serial number of the DLN-series adapter.

id
An ID number of the DLN-series adapter.

DLN_GET_DEVICE_COUNT Response

Go to command

The adapter sends the **DLN_GET_DEVICE_COUNT** response after the command execution. The response will contain the number of connected devices.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t deviceCount;
} __PACKED_ATTR DLN_GET_DEVICE_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GET_DEVICE_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_GET_DEVICE_COUNT response it is set to 0x0020. The DLN_MSG_ID_GET_DEVICE_COUNT constant can be used.
- **echoCounter** - Links a command to a response. The echoCounter value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. For the DLN_GET_DEVICE_COUNT response the HDLN_ALL_DEVICES value is used.

result
Contains the result of the command execution. The following values are available:

- DLN_RES_SUCCESS - the number of the connected devices has been successfully retrieved.

deviceCount
The number of connected devices.

1.7.7. DLN_OPEN_DEVICE

DLN_OPEN_DEVICE Command

Go to response

The **DLN_OPEN_DEVICE** command opens the specified device.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t filter;
   uint32_t number;
   uint32_t hardwareType;
   uint32_t sn;
   uint32_t id;
} __PACKED_ATTR DLN_OPEN_DEVICE_CMD;
```

Parameters

header

Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields

- **size** - The size of the message. Must be equal to the size of the DLN_OPEN_DEVICE_CMD structure.
- **msgId** - Defines the message. For the DLN_OPEN_DEVICE command it must be set to `0x0021`. You can use the `DLN_MSG_ID_OPEN_DEVICE` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. For the DLN_OPEN_DEVICE command the `HDLN_ALL_DEVICES` value must be used.

filter

A bit field that defines how the device will be identified for opening. Each bit corresponds to a DLN-series adapter identification, namely:

Bit	Identification
0	Device number
1	Hardware type
2	Device serial number
3	Device ID number

Only the identifications with corresponding mask bits set to 1 will be taken into account. The identifications with mask bits set to 0 will be ignored.

A user can set several mask bits to 1. This way only a device with all identifications matching the specified ones will be opened.

For example, if we need to open the device with the specific hardware type and ID number, we should set the `filter` parameter as follows:

1	0	1	0

You can use the following constants:

- `DLN_DEVICE_FILTER_NUMBER` - the `number` parameter;
- `DLN_DEVICE_FILTER_HW_TYPE` - the `hardwareType` parameter;
- `DLN_DEVICE_FILTER_SN` - the `sn` parameter;

- `DLN_DEVICE_FILTER_ID` - the `id` parameter;

number
A number of the connected device. Used for DlnOpenDevice() function.

hardwareType
A type of the DLN-series adapter.

sn
A serial number of the DLN-series adapter. Used for DlnOpenDeviceBySn() function.

id
An ID number of the DLN-series adapter. Used for DlnOpenDeviceById() function.

DLN_OPEN_DEVICE Response

Go to command

The adapter sends the **DLN_OPEN_DEVICE** response after the command execution. The response notifies you if the device has been successfully opened.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   HDLN-series adapterHandle;
} __PACKED_ATTR DLN_OPEN_DEVICE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. Is equal to the size of the DLN_OPEN_DEVICE_RSP structure.
- **msgId** - Defines the message. For the DLN_OPEN_DEVICE response it is set to `0x0021`. The `DLN_MSG_ID_CONNECT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the device was successfully opened. The `deviceHandle` parameter contains a valid handle.
- `DLN_RES_NOT_CONNECTED` - The library was not connected to any server (See DlnConnect() function).
- `DLN_RES_MEMORY_ERROR` - Not enough memory to process this command.
- `DLN_RES_HARDWARE_NOT_FOUND` - The number of available devices is less than `deviceNumber+1`
- `DLN_RES_DEVICE_REMOVED` - The device was disconnected while opening.

deviceHandle
A handle to the DLN-series adapter. For the DLN_OPEN_DEVICE response the HDLN_ALL_DEVICES value is used.

1.7.8. DLN_CLOSE_HANDLE

DLN_CLOSE_HANDLE Command

Go to response

The **DLN_CLOSE_HANDLE** command closes the handle to the opened DLN-series adapter (stream).

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_CLOSE_HANDLE_CMD;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_CLOSE_HANDLE_CMD structure.
- **msgId** - Defines the message. For the DLN_CLOSE_HANDLE command it must be set to 0x0023. You can use the DLN_MSG_ID_CLOSE_HANDLE constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same echoCounter value. This field can be any numerical value from 0 to 0xFFFF (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the HDLN_ALL_DEVICES value.

DLN_CLOSE_HANDLE Response

Go to command

The adapter sends the **DLN_CLOSE_HANDLE** response after the command execution. The result field informs a user if the connection has been successfully closed.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_CLOSE_HANDLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_CLOSE_HANDLE_RSP structure.

- **msgId** - Defines the message. For the DLN_CLOSE_HANDLE response it is set to `0x0023`. The `DLN_MSG_ID_CLOSE_HANDLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the handle to the device has been successfully closed.

1.7.9. DLN_CLOSE_ALL_HANDLES

DLN_CLOSE_ALL_HANDLES Command

Go to response

The **DLN_CLOSE_HANDLE** command closes all handles to opened DLN-series adapters (stream).

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_CLOSE_ALL_HANDLES_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_CLOSE_ALL_HANDLES_CMD structure.
- **msgId** - Defines the message. For the DLN_CLOSE_ALL_HANDLES command it must be set to `0x0024`. You can use the `DLN_MSG_ID_CLOSE_ALL_HANDLES` constant.
- **echoCounter** - Can be used to link a command to a response. The response to the command will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. For the DLN_CLOSE_ALL_HANDLES command the `HDLN_ALL_DEVICES` value must be used.

DLN_CLOSE_ALL_HANDLES Response

Go to command

The adapter sends the **DLN_CLOSE_ALL_HANDLES** response after the command execution. The `result` field informs a user if all connections were successfully closed.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
```

```
} __PACKED_ATTR DLN_CLOSE_ALL_HANDLES_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_CLOSE_ALL_HANDLES_RSP structure.
- **msgId** - Defines the message. For the DLN_CLOSE_ALL_HANDLES response it is set to `0x0024`. The `DLN_MSG_ID_CLOSE_ALL_HANDLES` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. For the DLN_CLOSE_ALL_HANDLES response the `HDLN_ALL_DEVICES` value is used.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the handles to all devices were successfully closed.

1.7.10. DLN_GET_VER

DLN_GET_VER Command

Go to response

The **DLN_GET_VER** command retrieves the following data about the DLN adapter:

- **Hardware type** - The type of the device (e.g. DLN-4);
- **Hardware version** - The version of the hardware, used in the device;
- **Firmware version** - The version of the firmware, installed in the device;
- **Server version** - The version of the server;
- **Library version** - The version of the DLN-library.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_GET_VER_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GET_VER_CMD structure.
- **msgId** - Defines the message. For the DLN_GET_VER command it must be set to `0x0030`. You can also use the `DLN_MSG_ID_GET_VER` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

DLN_GET_VER Response

Go to command

The adapter sends the **DLN_GET_VER** response after the command execution. The response will contain the retrieved information.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   DLN_VERSION version;
} __PACKED_ATTR DLN_GET_VER_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GET_VER_RSP structure.
- **msgId** - Defines the message. For the DLN_GET_VER response it is set to `0x0030`. The `DLN_MSG_ID_GET_VER` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the information has been successfully retrieved.

version
The DLN_VERSION structure, containing the version information.

1.7.11. DLN_GET_DEVICE_SN

GET_DEVICE_SN Command

Go to response

The **DLN_GET_DEVICE_SN** command retrieves a device serial number. The serial number is factory-assigned and cannot be changed.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_GET_DEVICE_SN_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GET_DEVICE_SN_CMD structure.
- **msgId** - Defines the message. For the DLN_GET_DEVICE_SN command it must be set to `0x0031`. You can use the `DLN_MSG_ID_GET_DEVICE_SN` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

DLN_GET_DEVICE_SN Response

Go to command

The adapter sends the **DLN_GET_DEVICE_SN** response after the command execution. The response will contain the device serial number.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t sn;
} __PACKED_ATTR DLN_GET_DEVICE_SN_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GET_DEVICE_SN_RSP structure.
- **msgId** - Defines the message. For the DLN_GET_DEVICE_SN response it is set to `0x0031`. The `DLN_MSG_ID_GET_DEVICE_SN` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the device serial number has been successfully retrieved.

sn
A device serial number

1.7.12. DLN_SET_DEVICE_ID

DLN_SET_DEVICE_ID Command

Go to response

The **DLN_SET_DEVICE_ID** command sets a new ID to the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint32_t id;
} __PACKED_ATTR DLN_SET_DEVICE_ID_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SET_DEVICE_ID_CMD structure.
- **msgId** - Defines the message. For the DLN_SET_DEVICE_ID command it must be set to `0x0032`. You can use the `DLN_MSG_ID_SET_DEVICE_ID` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

id
A new device ID to be set.

DLN_SET_DEVICE_ID Response

Go to command

The adapter sends the **DLN_SET_DEVICE_ID** response after the command execution. The `result` field informs a user if the device ID has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SET_DEVICE_ID_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SET_DEVICE_ID_RSP structure.

- **msgId** - Defines the message. For the DLN_SET_DEVICE_ID response it is set to `0x0032`. The `DLN_MSG_ID_SET_DEVICE_ID` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the device ID was successfully set.

1.7.13. DLN_GET_DEVICE_ID

DLN_GET_DEVICE_ID Command

Go to response

The **DLN_GET_DEVICE_ID** command retrieves the device ID number.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_GET_DEVICE_ID_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GET_DEVICE_ID_CMD structure.
- **msgId** - Defines the message. For the DLN_GET_DEVICE_ID command it must be set to `0x0033`. You can use the `DLN_MSG_ID_GET_DEVICE_ID` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

DLN_GET_DEVICE_ID Response

Go to command

The adapter sends the **DLN_GET_DEVICE_ID** response after the command execution. The response will contain the retrieved device ID number.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t id;
```

```
} __PACKED_ATTR DLN_GET_DEVICE_ID_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GET_DEVICE_ID_RSP structure.
- **msgId** - Defines the message. For the DLN_GET_DEVICE_ID response it is set to `0x0033`. The `DLN_MSG_ID_GET_DEVICE_ID` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the device ID was successfully retrieved.

id
A device ID number.

1.7.14. DLN_GET_PIN_CFG

DLN_GET_PIN_CFG Command

Go to response

The **DLN_GET_PIN_CFG** command retrieves current configuration of the DLN-series adapter pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t pin;
} __PACKED_ATTR DLN_GET_PIN_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GET_PIN_CFG structure.
- **msgId** - Defines the message. For the DLN_GET_PIN_CFG command it must be set to `0x0040`. You can use the `DLN_MSG_ID_GET_PIN_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to get the configuration from.

DLN_GET_PIN_CFG Response

Go to command

The adapter sends the **DLN_GET_PIN_CFG** response after the command execution. The response contains current configuration of the specified DLN-series adapter pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   DLN_PIN_CFG cfg;
} __PACKED_ATTR DLN_GET_PIN_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GET_PIN_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_PIN_CFG response it is set to `0x0040`. The `DLN_MSG_ID_GET_PIN_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the pin cfg was successfully retrieved.

cfg
Contains the pin configuration DLN_PIN_CONFIG

If the pin is currently not in use, the module parameter will be set to `0x00` or `DLN_MODULE_GENERIC` while the `role` parameter will be set to `DLN_PIN_ROLE_NOT_IN_USE`

1.8. Events

1.8.1. DLN_CONNECTION_LOST

The **DLN_CONNECTION_LOST** event is generated when connection to a server is lost. The event contains the host address and port number of the server.

```
typedef struct
{
   DLN_MSG_HEADER header;
   char host[DLN_MAX_HOST_LENGTH+1];
   uint16_t port;
} __PACKED_ATTR DLN_CONNECTION_LOST_EV;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The event header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_CONNECTION_LOST_EV structure.
- **msgId** - Defines the message. For the DLN_CONNECTION_LOST event it is set to `0x000F`. The `DLN_MSG_ID_CONNECTION_LOST_EV` constant can be used.
- **echoCounter** - A freerunning counter. It shows the number of events sent by the device. The `echoCounter` is reset when its value reaches 0xFFFF (65536).
- **handle** - A handle to the DLN-series adapter. For the DLN_CONNECTION_LOST event the `HDLN_ALL_DEVICES` value is used.

host
A server, the connection to which was lost. This value can be a URL (e.g. www.diolan.com), an IP address (e.g. 127.0.0.1) or the predefined `"localhost"` value.

port
A port number of the DLN server.

1.8.2. DLN_DEVICE_REMOVED

The `DLN_DEVICE_REMOVED` event is generated when a device is removed from a server. The event contains the handle to the removed device.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_DEVICE_REMOVED_EV;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The event header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_DEVICE_REMOVED_EV structure.
- **msgId** - Defines the message. For the DLN_DEVICE_REMOVED event it is set to `0x002E`. The `DLN_MSG_ID_DEVICE_REMOVED_EV` constant can be used.
- **echoCounter** - A freerunning counter. It shows the number of events sent by the device. The `echoCounter` is reset when its value reaches 0xFFFF (65536).
- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

1.8.3. DLN_DEVICE_ADDED

The `DLN_DEVICE_ADDED` event is generated when a device is added to a server. The event contains the type of the device and its identification.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint32_t hardwareType;
   uint32_t id;
   uint32_t sn;
} __PACKED_ATTR DLN_DEVICE_ADDED_EV;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The event header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_DEVICE_ADDED_EV structure.
- **msgId** - Defines the message. For the DLN_DEVICE_ADDED event it is set to `0x002F`. The `DLN_MSG_ID_DEVICE_ADDED_EV` constant can be used.
- **echoCounter** - A freerunning counter. It shows the number of events sent by the device. The `echoCounter` is reset when its value reaches 0xFFFF (65536).
- **handle** - A handle to the DLN-series adapter. Can be either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

hardwareType
A type of the DLN-series adapter

id
An ID number of the DLN-series adapter.

sn
A serial number of the DLN-series adapter.

Chapter 2. GPIO Module

GPIO module allows you to use the adapter pins as general purpose input/output lines. DLN-4 adapter has 48 pins grouped together in 6 ports. You can individually configure each pin as an input or as an output.

Each I/O line of the DLN adapter features:

- A debounce filter providing rejection of unwanted pulses from key or push button operations;
- Open drain functionality;
- Control of the embedded pull-up resistors of the I/O line;

A DLN-series adapter can be configured to send DLN_GPIO_EVENT events. They are generated when the level on an I/O line meets specified requirements. A user can set the events to be recurrent in order to continuously monitor the level on the I/O lines.

2.1. Digital Outputs

Each of the adapter pins can be configured as a general purpose digital output.

This can be done using DlnGpioPortSetCfg() function.

You must specify the corresponding `handle` and `port` values, as well as configure `validFields`, `mask` and `config` parameters.

In order to configure pin direction, a user must set the first bit in a `validFields` parameter to 1 (you can use a `DLN_GPIO_FIELD_ISOUTPUT` constant). This way the corresponding configuration field (`isOutput`) becomes valid.

The `config` parameter contains the DLN_GPIO_PORT_CONFIG structure, which is filled with new configuration values to be set. Its second field - `isOutput` - triggers the pin direction and must be set to 1 for output.

The `mask` parameter in this case is used to define the I/O lines, whose I/O configuration must be triggered. Set the bits, corresponding to the pins to be affected to 1.

If the pin is not configured as a GPIO - line, you must set the `isEnabled` parameter, as well as zeroth bit in the `validFields` parameter to 1.

The third field of the structure - `outputValue` - is used to set an output level of an I/O line and must be set to 0 or 1 for logic 0 and logic 1 respectively. In order to change a port's output value, a user must set the second bit in the `validFields` parameter to 1 (you can use the `DLN_GPIO_FIELD_OUTPUTVAL` constant). This way we can simultaneously connect a pin to GPIO module, configure it as output and set its output value.

We can further change the output values of digital outputs using the DlnGpioPortSetOutVal() function.

Digital outputs can function in the Open Drain mode. A user can also enable embedded Pull-Up resistors for each of the digital outputs.

The current level on each I/O line can be determined using the DlnGpioPortGetVal() function. This function returns levels on I/O lines regardless of their configuration, whether they are configured as GPIO inputs or outputs or even not connected to GPIO module at all. Another function - DlnGpioPortGetOutVal() retrieves pin's output values. If the pin is configured as digital output, this value matches the one returned by the DlnGpioPortGetVal() function. If the pin is not configured as digital output, the value is taken from an internal latch buffer.

2.2. Digital Inputs

Each of the adapter pins can be configured as a general purpose digital input.

This can be done using the DlnGpioPortSetCfg() function.

You must specify the corresponding `handle` and `port` values, as well as configure `validFields`, `mask` and `config` parameters.

In order to configure pin direction, a user must set the first bit in the `validFields` parameter to 1 (you can use the `DLN_GPIO_FIELD_ISOUTPUT` constant). This way the corresponding configuration field (`isOutput`) becomes valid.

The `config` parameter contains the DLN_GPIO_PORT_CONFIG structure, which is filled with new configuration values to be set. Its second field - `isOutput` - triggers the pin direction and must be set to 0 for input.

The `mask` parameter is used to define the I/O lines, whose I/O configuration must be triggered. Set the bits, corresponding to the pins to be affected to 1.

In order to use a pin as a digital input, it must be connected to GPIO module. To do this, you must first set the zeroth bit in the `validFields` parameter to 1. Then you must set all the corresponding bits from the `isEnabled` parameter to 1.

A user can enable embedded Pull-Up resistors and the Debounce Filter for each of the digital inputs.

The current level on each I/O line can be determined using the DlnGpioPortGetVal() function. This function returns the level on an I/O line regardless of its configuration, whether it is configured as a GPIO input or output or even not connected to the GPIO module at all.

Digital inputs can be configured to send events on an I/O line level change. The DLN-adapter can also generate periodical events containing current state of the line. The period can be defined using the `eventPeriod` parameter of the DLN_GPIO_PORT_CONFIG structure. This way a user can continuously monitor the input value on the line. For details see "Digital Input Events".

2.3. Digital Input Events

A DLN-series adapter can be configured to send DLN_GPIO_EVENT events. They are generated when the level on an I/O meets specified requirements. Digital inputs events are configured using the DLN_GPIO_PORT_CONFIG structure. The `eventType` parameter defines conditions for event generation. For some of the event types the `eventPeriod` parameter can be defined. In this case the device will send recurrent events each `eventPeriod` milliseconds.

There are five types of events:

- DLN_GPIO_EVENT_NONE - no events are generated;
- DLN_GPIO_EVENT_CHANGE - events are generated when the level on the digital input line changes;
- DLN_GPIO_EVENT_LEVEL_HIGH - events are generated when the high level is detected on the digital input line;
- DLN_GPIO_EVENT_LEVEL_LOW - events are generated when the low level is detected on the digital input line;
- DLN_GPIO_EVENT_ALWAYS - events are generated continuously, regardless of the signal level.

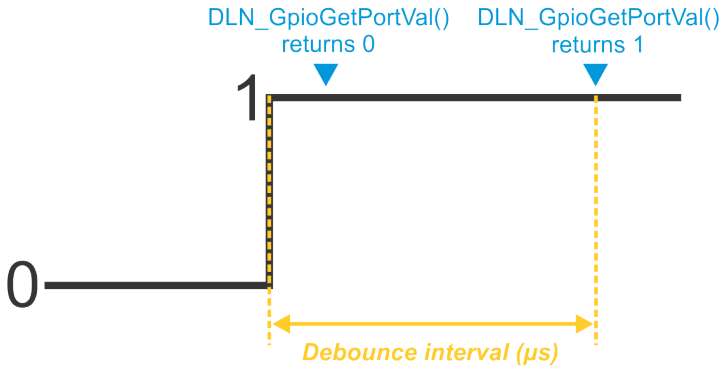
2.3.1. DLN_GPIO_EVENT_NONE

No events are generated.

In this case the `eventPeriod` value must be set to 0.

The current level on a digital input can be retrieved using the DlnGpioPortGetVal() function.

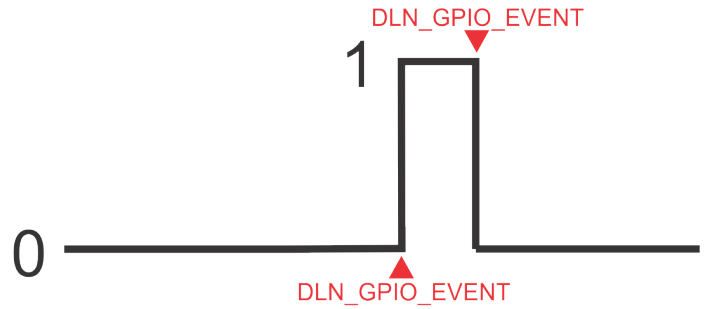
We can enable the debounce filter for the current pin. The debounce filter affects the values, returned by the DlnGpioPortGetVal() function. If we call the DlnGpioPortGetVal() function at the point where the level on the I/O line changes, it will return the previous value. The new value is returned only after it has been stable for a predefined period of time.



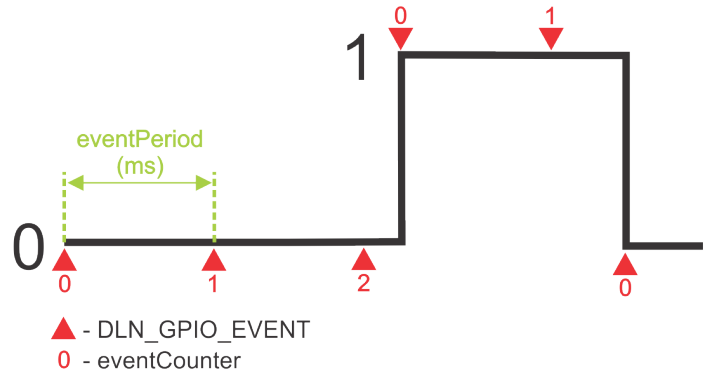
2.3.2. DLN_GPIO_EVENT_CHANGE

Events are generated when the level on the digital input line changes.

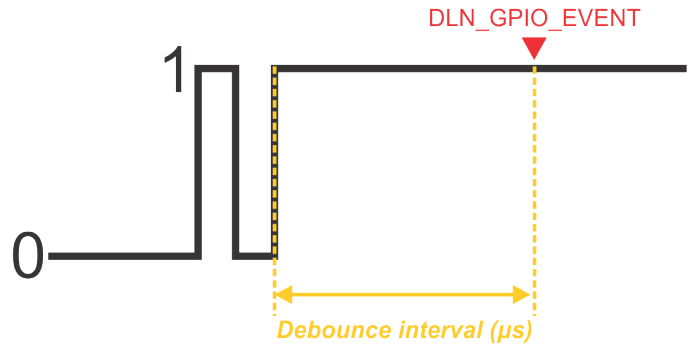
If the `eventPeriod` parameter is 0, a single event is generated, when the level on the line changes.



If a user defines the non-zero `eventPeriod` parameter, the device will send recurrent events every `eventPeriod` milliseconds. The eventCount parameter contains a number of events, sent after the previous level change. When the level on the digital input changes, the device sends an event immediately. It doesn't wait for an `eventPeriod` since the previous event to pass. The eventCount parameter is reset to 0. Further recurrent event counting will be started from this point.



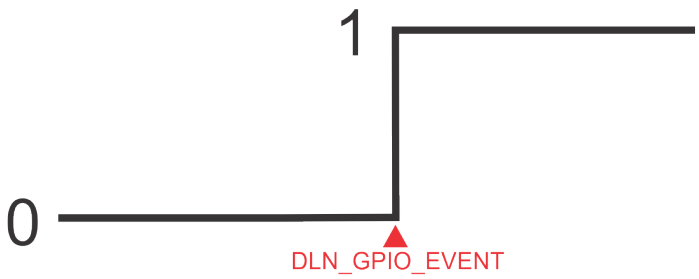
If the `isDebounceEnabled` parameter is set, the changes on the line are registered only after the signal has been stable for a predefined period of time.



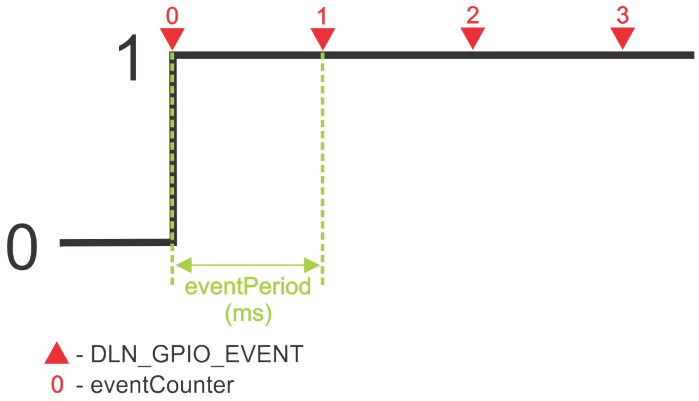
2.3.3. DLN_GPIO_EVENT_LEVEL_HIGH

Events are generated when the high level (logical 1) is detected on the digital input line.

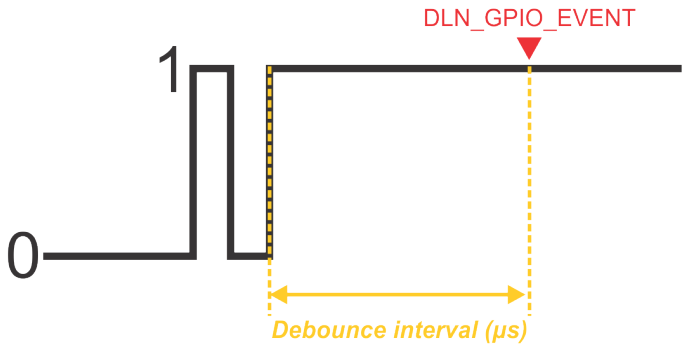
If the `eventPeriod` parameter is 0, a single event is generated, when the level on the line changes from logical 0 to logical 1.



If the `eventPeriod` is non-zero, the device will send recurrent events every `eventPeriod` milliseconds, while high level is present on the line. The DLN_GPIO_EVENT structure contains the eventCount parameter. It contains a number of events, sent after the previous level change.



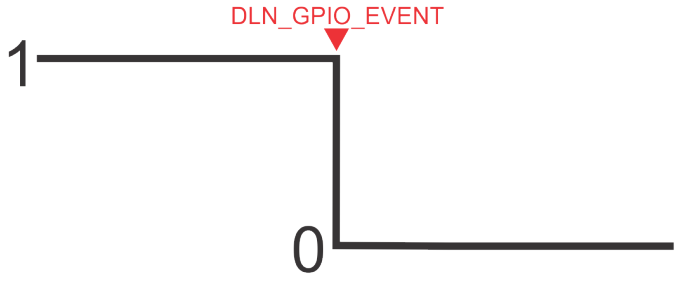
If the `isDebounceEnabled` parameter is set, the changes on the line are registered only after the signal has been stable for a predefined period of time.



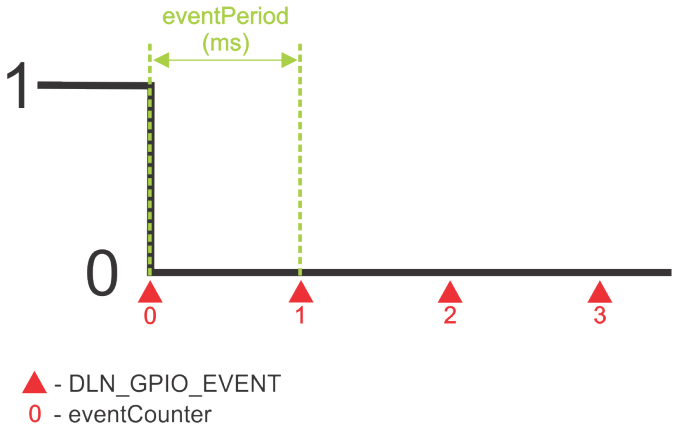
2.3.4. DLN_GPIO_EVENT_LEVEL_LOW

Events are generated when the low level (logical 0) is detected on the digital input line.

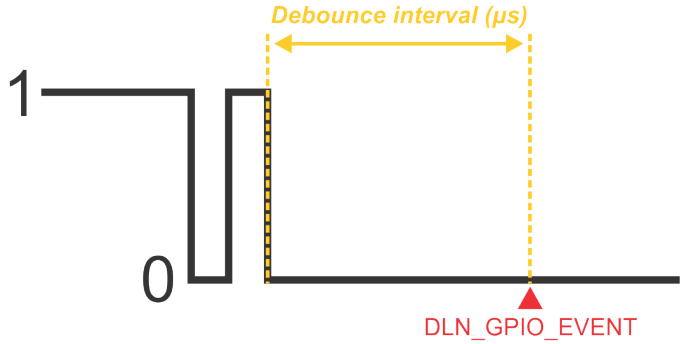
If the `eventPeriod` parameter is 0, a single event is generated, when the level on the line changes from logical 1 to logical 0.



If the `eventPeriod` is non-zero, the device will send recurrent events every `eventPeriod` milliseconds, while low level is present on the line. The DLN_GPIO_EVENT structure contains the eventCount parameter. It contains a number of events, sent after the previous level change.



If the `isDebounceEnabled` parameter is set, the changes on the line are registered only after the signal has been stable for a predefined period of time.

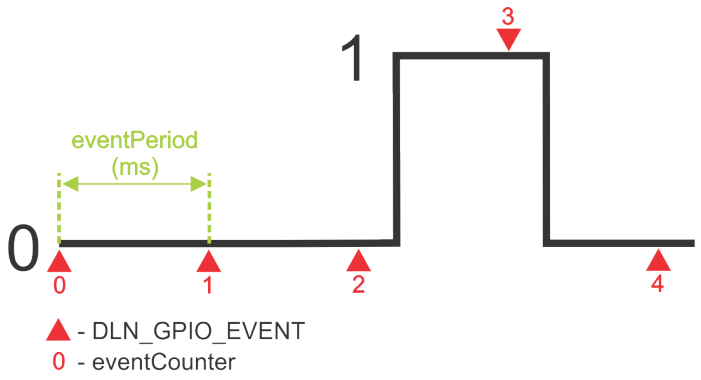


2.3.5. DLN_GPIO_EVENT_ALWAYS

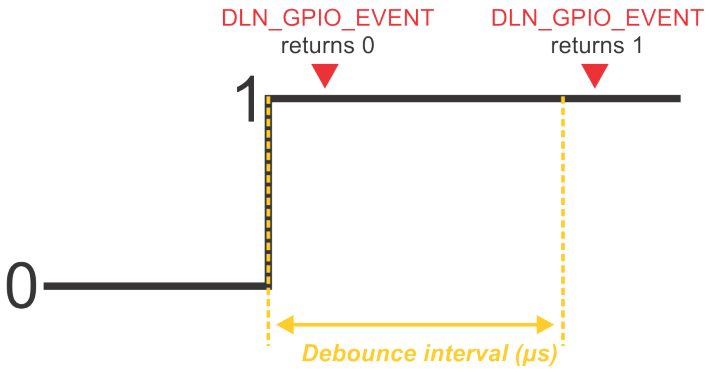
Events are generated continuously, regardless of the signal level. A user must define the non-zero `eventPeriod` parameter. The device sends the DLN_GPIO_EVENT event every `eventPeriod` milliseconds. The event contains the current value on the input line.

You can find out the number of events, sent after the configuration setting, from the eventCount parameter.

The level change does not trigger an additional event generation. The eventCount parameter is not reset to 0. A user can learn about the level change from the next scheduled event.



If the `isDebounceEnabled` parameter is enabled, the changes on the line are registered only after the signal has been stable for a predefined period of time. Therefore, if a recurrent event is sent at the point where the level on the I/O line changes, it will return the previous value. The new value is returned only after it has been stable for a predefined period of time.

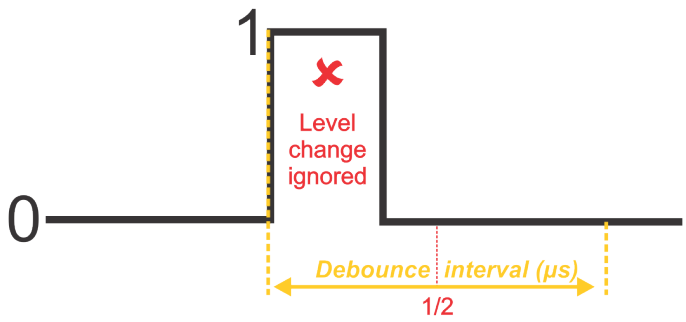


2.4. Debounce Filter

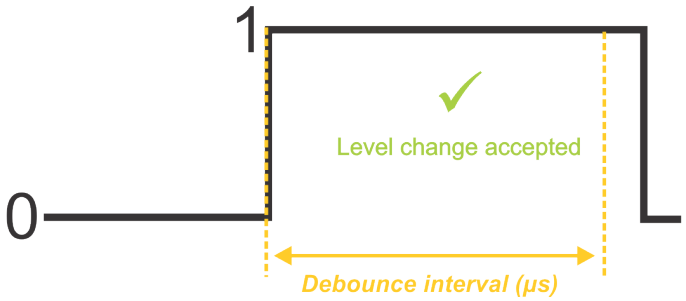
Contact bounce may cause sending numerous events. To avoid it, debounce filter is implemented in the adapter. The new value is only accepted if it is stable for a predefined period of time. This time period

(**Debounce interval**) is defined with DLN_GPIO_SET_DEBOUNCE command in µs. This time period is equal for all pins. The debounce filter can be switched on/off independently for each I/O line.

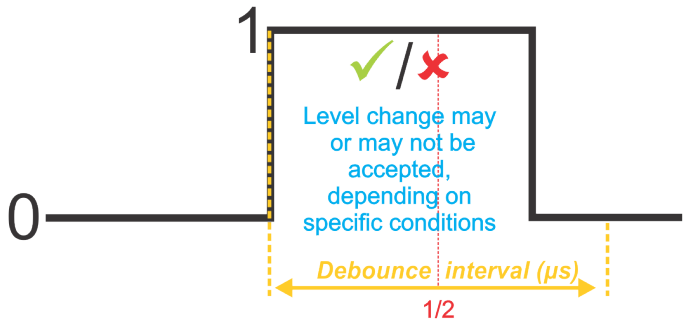
When the debouncing filter is enabled, a pulse with duration of less than 1/2 of the specified debounce interval is automatically rejected.



A pulse with a duration of 1 or more debounce intervals is accepted, and a respective event is generated.



For pulse durations between 1/2 and 1 debounce interval, the pulse may or may not be taken into account, depending on the precise timing of its occurrence.



Thus for a pulse to be visible it must exceed 1 debounce interval, whereas for a pulse to be reliably filtered out, its duration must be less than 1/2 of selected time period.

Debounce filters can be enabled or disabled using DlnGpioPortSetCfg() function.

To do this you must specify the corresponding `handle` and `port` values, as well as configure `validFields`, `mask` and `config` parameters.

In order to trigger debounce filters, a user must set the fifth bit in a `validFields` parameter to 1 (you can use the `DLN_GPIO_FIELD_ISDEBOUNCEEN` constant). This way the corresponding configuration field (`isDebounceEnabled`) becomes valid.

The `mask` parameter is used to define the I/O lines, whose debounce filter settings must be changed. Set the bits, corresponding to the pins that should be affected to 1.

The `config` parameter contains the DLN_GPIO_PORT_CONFIG structure, which is filled with new configuration values to be set. Its sixth field - `isDebounceEnabled` - triggers the debounce filter and must be set to 0 or 1 in order to disable or enable it respectively.

Another way to enable debounce filtering for a single pin is calling the DlnGpioPinDebounceEnable() function. The DlnGpioPinDebounceDisable() function disables the debounce filter, while DlnGpioPinDebounceIsEnabled() retrieves its current state.

By default all the debounce filters are disabled.

2.5. Open Drain Mode

Each I/O line can be independently programmed in open drain mode. This feature permits several outputs to be connected on a single I/O line. An open drain terminal is connected to ground in the low voltage (logic 0) state, but has high impedance in the logic 1 state. This prohibits current flow, but as a result, such a device requires an external (or embedded) pull-up resistor connected to the positive voltage rail (logic 1). If all outputs attached to the line are in the high-impedance (i.e., logic 1) state, the pull-up resistor will hold the wire in a high voltage state. If 1 or more of the device outputs are in the ground (i.e., logic 0) state, they will sink current and pull the line voltage near ground.

Open Drain can be enabled or disabled using the DlnGpioPortSetCfg() function.

To do this you must specify the corresponding `handle` and `port` values, as well as configure `validFields`, `mask` and `config` parameters.

In order to trigger the open drain mode, a user must set the third bit in a `validFields` parameter to 1 (you can use a `DLN_GPIO_FIELD_ISOPENDRAIN` constant). This way the corresponding configuration field (`isOpenDrain`) becomes valid.

The `mask` parameter is used to define the I/O lines, whose open drain setting must be changed. Set the bits, corresponding to the pins that should be affected to 1.

The `config` parameter contains the DLN_GPIO_PORT_CONFIG structure, which is filled with new configuration values to be set. Its fourth field - `isOpenDrain` - triggers open drain mode and must be set to 0 or 1 in order to disable or enable it respectively.

The DlnGpioPinOpendrainEnable() function activates Open Drain mode for a single pin. In order to disable Open Drain mode for a single pin call the DlnGpioPinOpendrainDisable() function. Current state of the mode can be retrieved using the DlnGpioPinOpendrainIsEnabled()

After reset, the open drain mode is disabled on all pins.

2.6. Pull-up Resistors

Each I/O line is equipped with an embedded pull-up resistor. Pull-up resistors are used to ensure that inputs to I/O lines settle at expected levels, if external devices are disconnected or high-impedance.

A pull-up resistor can be enabled or disabled using a DlnGpioPortSetCfg() function.

To do this you must specify the corresponding `handle` and `port` values, as well as configure `validFields`, `mask` and `config` parameters.

In order to trigger pull up resistors, a user must set the fourth bit in a `validFields` parameter to 1 (you can use a `DLN_GPIO_FIELDISPULLUPEN` constant). This way the corresponding configuration field (`isPullUpEnabled`) becomes valid.

The `mask` parameter is used to define the pins, whose pull-up resistors must be triggered. Set the bits, corresponding to the pins that should be affected to 1.

The `config` parameter contains the DLN_GPIO_PORT_CONFIG structure, which is filled with new configuration values to be set. Its fifth field - `isPullUpEnabled` - triggers pull-up resistors and must be set to 0 or 1 in order to disable or enable them respectively.

A pull-up resistor can be enabled for a single pin using the DlnGpioPinPullupEnable() function and disabled using the DlnGpioPinPullupDisable(). In order to retrieve current state of the pull-up resistor, call the DlnGpioPinPullupIsEnabled().

By default all the pull-up resistors are enabled.

2.7. Structures

2.7.1. DLN_GPIO_PORT_CONFIG

```
typedef struct
{
   DLN_GPIO_PIN_CONFIG pins[8];
} __PACKED_ATTR DLN_GPIO_PORT_CONFIG;
```

This structure is used to store configuration of GPIO pins from a single port.

The configuration of each pin is set individually, using the DlnGpioPinSetConfig() function.

Members:

pins
An 8-element array. Each of its elements corresponds to DLN_GPIO_PIN_CONFIG structure for each pin of the port.

2.7.2. DLN_GPIO_PIN_CONFIG

```
typedef struct
{
   uint16_t cfg;
   uint8_t eventType;
   uint16_t eventPeriod;
} __PACKED_ATTR DLN_GPIO_PIN_CONFIG;
```

This structure is used to store configuration of a single GPIO pin.

Members:

cfg
A bit field, consisting of 16 bits. Each of the bits `0`-`5` corresponds to a specific parameter, that defines the pin configuration. The bits `6` and `7` are reserved. You can also use special constants, defined in the dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00005.html] file for each of the bits.

Bit	Value	Description	Constant
0	1	The pin is configured as a general purpose input/output.	DLN_GPIO_ENABLED
0	0	The pin is not configured as a general purpose input/output.	DLN_GPIO_DISABLED
1	1	The pin is configured as output.	DLN_GPIO_OUTPUT
1	0	The pin is configured as input.	DLN_GPIO_INPUT
2	1	The output value on the pin is 1.	DLN_GPIO_OUTPUT_VAL_1
2	0	The output value on the pin is 0.	DLN_GPIO_OUTPUT_VAL_0
3	1	The output is open drain.	DLN_GPIO_OPEN_DRAIN_ENABLED
3	0	The output is push pull.	DLN_GPIO_OPEN_DRAIN_DISABLED
4	1	The pull-up resistor is on.	DLN_GPIO_PULLUP_ENABLED
4	0	The pull-up resistor is off.	DLN_GPIO_PULLUP_DISABLED
5	1	The debounce filter is switched on.	DLN_GPIO_DEBOUNCE_ENABLED
5	0	The debounce filter is switched off.	DLN_GPIO_DEBOUNCE_DISABLED
6		Reserved.	
7		Reserved.	
8	1	Events are enabled.	DLN_GPIO_EVENT_ENABLED
8	0	Events are disabled.	DLN_GPIO_EVENT_DISABLED
9	1	Events are periodic.	DLN_GPIO_EVENT_PERIODIC
9	0	A single event is generated.	DLN_GPIO_EVENT_SINGLE
10		Reserved.	
11		Reserved.	
12		Reserved.	
13		Reserved.	
14		Reserved.	
15		Reserved.	

Configuration parameters:

DLN_GPIO_ENABLE_BIT

The zeroth bit in the `cfg` parameter. It defines whether the pin is configured as general purpose input/output or not. The following values are available.

- `1` – `DLN_GPIO_ENABLED` - the pin is configured as general purpose I/O;
- `0` – `DLN_GPIO_DISABLED` the pin is disconnected from the GPIO module.

DLN_GPIO_OUTPUT_BIT

The first bit in the `cfg` parameter. It defines whether the pin is configured as input or as output. The following values are available

- `1` – `DLN_GPIO_OUTPUT` - the pin is configured as output;
- `0` – `DLN_GPIO_INPUT` - the pin is configured as input.

DLN_GPIO_OUTPUT_VAL_BIT

The second bit in the `cfg` parameter. It defines the output value on the pin. If the pin is an output, the value is applied immediately. If the pin is an input, the value is stored in a latch. This value will be applied when the pin becomes an output. The following values are available:

- `1` – `DLN_GPIO_OUTPUT_VAL_1`the pin outputs high level (logic 1);
- `0` – `DLN_GPIO_OUTPUT_VAL_0`the pin outputs low level (logic 0).

DLN_GPIO_OPEN_DRAIN_BIT
The third bit in the `cfg` parameter. Defines whether the output is push pull or open drain. If the pin is an output, the parameter is applied immediately. If the pin is an input, the parameter is stored in a latch. This parameter will be applied when the pin becomes an output. The following values are available:

- `1` – `DLN_GPIO_OPEN_DRAIN_ENABLED` - the pin is an open drain output;
- `0` – `DLN_GPIO_OPEN_DRAIN_DISABLED` - the pin is a push pull output.

DLN_GPIO_PULL_UP_BIT
The fourth bit in the `cfg` parameter. Defines whether the pull-up resistor is on or off. The following values are available:

- `1` – `DLN_GPIO_PULLUP_ENABLED` - pull-up on the pin is switched on;
- `0` – `DLN_GPIO_PULLUP_DISABLED` - pull-up on the pin is switched off.

DLN_GPIO_DEBOUNCE_BIT
The fifth bit in the `cfg` parameter. Defines whether the debounce filter is switched on or not. The following values are available:

- `1` – `DLN_GPIO_DEBOUNCE_ENABLED` - debounce filter is applied to the pin;
- `0` – `DLN_GPIO_DEBOUNCE_DISABLED` - debounce filter is not applied to the pin.

The debounce duration is set with DLN_GPIO_SET_DEBOUNCE command.

DLN_GPIO_EVENT_TYPE_BIT
The eighth bit in the `cfg` parameter. Defines whether event generation on the pin is enabled as well as the condition of event generation for the pin. The following values are available:

- `1` – `DLN_GPIO_EVENT_ENABLED` - events are generated on the pin;
- `0` – `DLN_GPIO_EVENT_DISABLED` - no events are generated on the pin.

For more detailed information about the event generation see GPIO Event

DLN_GPIO_EVENT_PERIOD_BIT
The ninth bit in the `cfg` parameter. Defines whether the DLN_GPIO_CONDIION_MET event is single or periodic. The following values are available:

- `1` – `DLN_GPIO_EVENT_PERIODIC` - the events are generated periodically;
- `0` – `DLN_GPIO_EVENT_SINGLE` - a single event is generated.

eventType
Defines the condition of event generation for the pin. The following values are available:

- DLN_GPIO_EVENT_NONE – no events are sent for current pin;
- DLN_GPIO_EVENT_CHANGE – an event is sent when the input value on the pin changes;
- DLN_GPIO_EVENT_LEVEL_HIGH – events are generated when the high level (logical 1) is detected on the digital input line;
- DLN_GPIO_EVENT_LEVEL_LOW – events are generated the event is sent when the low level (logical 0) is detected on the digital input line;
- DLN_GPIO_EVENT_ALWAYS – the events are sent periodically with predefined repeat interval.

The non-zero interval must be specified for this event type.

For more detailed information about the event generation see GPIO Event

eventPeriod

Defines the repeat interval for DLN_GPIO_CONDIION_MET_EV event generation on the pin. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the device will send a single event when the level on the line changes to meet the specified conditions.

2.8. Functions

This section describes the GPIO functions. They are used to control and monitor the GPIO module of a DLN-series adapter.

Actual control of the device is performed by use of commands and responses. Each function utilizes respective commands and responses. You can send such commands directly if necessary.

- DlnGpioGetPortCount() - retrieves the total number of the GPIO ports available in the DLN-series adapter;
- DlnGpioGetPinCount() - retrieves the total number of GPIO pins available in the DLN-series device;
- DlnGpioPinSetCfg() - changes the configuration of a single GPIO pin;
- DlnGpioPinGetCfg() - retrieves current configuration of the specified GPIO pin;
- DlnGpioPortGetCfg() - retrieves current configuration of the GPIO pins from the specified port;
- DlnGpioPortSetCfg() - changes the configuration of the GPIO pins from the specified port;
- DlnGpioPortGetVal() - retrieves the current level on the I/O lines from the specified port;
- DlnGpioPortSetOutVal() - sets output values for the I/O lines from GPIO the specified port;
- DlnGpioPortGetOutVal() - retrieves current output values for the GPIO pins from the specified port.
- DlnGpioSetDebounce() - Specifies the minimum duration of the pulse to be registered (the Debounce interval).
- DlnGpioGetDebounce() - Retrieves the current setting of the minimum duration of the pulse to be registered (the Debounce interval).
- DlnGpioPinGetOutVal() - Retrieves the pin output value, stored in the internal latch.
- DlnGpioPinSetOutVal() - Sets the output value for the specified GPIO pin.
- DlnGpioPinGetVal() - Retrieves the current value on the specified GPIO pin.
- DlnGpioPinEnable() - Configures a pin as general purpose input/output.
- DlnGpioPinDisable() - Disables a pin as general purpose input/output.
- DlnGpioPinIsEnabled() - Informs whether a pin is currently configured as general purpose input/output.
- DlnGpioPinSetDirection() - Configures a pin as input or as output.
- DlnGpioPinGetDirection() - Retrieves current direction of a pin.
- DlnGpioPinOpendrainEnable() - Enables Open Drain mode for the specified pin.
- DlnGpioPinOpendrainDisable() - Disables Open Drain mode for the specified pin.
- DlnGpioPinOpendrainIsEnabled() - Informs whether the pin output is currently configured as push pull or Open Drain.

- DlnGpioPinPullupEnable() - Activates an embedded pull-up resistor for the specified pin.
- DlnGpioPinPullupDisable() - Deactivates an embedded pull-up resistor for the specified pin.
- DlnGpioPinPullupIsEnabled() - Informs whether an embedded pull-up resistor is enabled for the specified pin.
- DlnGpioPinDebounceEnable() - Enables debounce filtering for the specified pin.
- DlnGpioPinDebounceDisable() - Disables debounce filtering for the specified pin.
- DlnGpioPinDebounceIsEnabled() - Informs whether the debounce filtering is currently enabled for the specified pin.
- DlnGpioPinSetEventCfg() - Configures the event generation conditions for the specified pin.
- DlnGpioPinGetEventCfg() - Retrieves current event generation conditions for the specified pin.

2.8.1. DlnGpioGetPortCount()

The `DlnGpioGetPortCount()` function retrieves the total number of the GPIO ports available in your DLN-series adapter.

```
DLN_RESULT DlnGpioGetPortCount(
   HDLN handle,
   uint8_t* count
);
```

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. This integer will be filled with the number of available ports after function execution.

Return Values:

- `DLN_RES_SUCCESS` - the GPIO port count has been successfully retrieved.

Table 2.1. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Total number of GPIO ports	6	6	4
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.2. DlnGpioGetPinCount()

```
DLN_RESULT DlnGpioGetPinCount(
   HDLN handle,
   uint16_t* count
);
```

The `DlnGpioGetPinCount()` function retrieves the total number of GPIO pins available in the DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 16-bit integer. This integer will be filled with the number of available pins after the function execution.

Return Values:

- `DLN_RES_SUCCESS` - GPIO pin count has been successfully retrieved.

Table 2.2. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Total number of GPIO pins	48	48	32
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.3. DlnGpioPinSetCfg()

```
DLN_RESULT DlnGpioPinSetCfg(
   HDLN handle,
   uint16_t pin,
   uint16_t validFields,
   DLN_GPIO_PIN_CONFIG config
);
```

The `DlnGpioPinSetCfg()` function changes the configuration of a single GPIO pin.

With this function you can either reconfigure the pin entirely or change only some of its parameters. This is what the `validFields` parameter is for.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A pin to be configured.

validFields
A bit field that defines the configuration parameters to be updated by this function. Each of the 16 validFields bits corresponds to a specific parameter in the DLN_GPIO_PIN_CONFIG structure. If you set the bit to 1, the new configuration parameter will be applied. If you set the bit to 0, the configuration parameter will remain unchanged regardless of its value in the DLN_GPIO_PIN_CONFIG structure. A user can also configure the pin parameters, using the constants, defined in the dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00005.html] file. If several constant are used, they should be separated with "| " (binary "or").

Several bits are reserved for future use and must be set to 0.

Bit	Corresponds to	Constant
0	Bit 0 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_ENABLE_BIT
1	Bit 1 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_OUTPUT_BIT
2	Bit 2 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_OUTPUT_VAL_BIT
3	Bit 3 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_OPEN_DRAIN_BIT
4	Bit 4 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_PULL_UP_BIT
5	Bit 5 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_DEBOUNCE_BIT
6	reserved	
7	reserved	
8	DLN_GPIO_PIN_CONFIG::eventType	DLN_GPIO_EVENT_TYPE_BIT
9	DLN_GPIO_PIN_CONFIG::eventPeriod	DLN_GPIO_EVENT_PERIOD_BIT
10	reserved	
11	reserved	
12	reserved	
13	reserved	
14	reserved	
15	reserved	

In order to include a configuration field in the operation, set the corresponding bit to 1. If we set a bit to 0, the field will be ignored.

For example, if we only need to change isOutput and eventType settings, our `validFields` byte should look like this:

0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0

A user can configure the pin using the constants, defined in the dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00005.html] file. In this case, the `validFields` byte should look like this:

```
validFields = DLN_GPIO_OUTPUT_BIT | DLN_GPIO_EVENT_TYPE_BIT;
```

config

A configuration to be set. See "DLN_GPIO_PIN_CONFIG" structure for details.

Return Values:

- `DLN_RES_SUCCESS` - the new configuration has been successfully applied;
- `DLN_RES_INVALID_HANDLE` - the specified handle is not valid;
- `DLN_RES_CONNECTION_LOST` - the connection to the DLN server was interrupted;
- `DLN_RES_INVALID_PIN_NUMBER` - the number of the pin is out of range. Use DlnGpioGetPinCount() function to get the available number of pins for your DLN-series adapter;
- `DLN_RES_NON_ZERO_RESERVED_BIT` - one or more of the reserved bits in `validFields` or `config` parameters are set to 1.

Table 2.3. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.4. DlnGpioPinGetCfg()

```
DLN_RESULT DlnGpioPinGetCfg(
   HDLN handle,
   uint16_t pin,
   DLN_GPIO_PIN_CONFIG* config
);
```

The `DlnGpioPinGetCfg()` function retrieves current configuration of the specified GPIO pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A pin to get configuration for.

config
A pointer to the DLN_GPIO_PIN_CONFIG structure which will be filled with the configuration after function execution.

Return values:

- `DLN_RES_SUCCESS` - the GPIO pin configuration was successfully retrieved;
- `DLN_RES_INVALID_HANDLE` - the specified handle is not valid;
- `DLN_RES_CONNECTION_LOST` - connection to the DLN server was interrupted;
- `DLN_RES_INVALID_PIN_NUMBER` - wrong pin number was specified.

Table 2.4. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.5. DlnGpioPortGetCfg()

```
DLN_RESULT DlnGpioPortGetCfg(
   HDLN handle,
   uint8_t port,
   DLN_GPIO_PORT_CONFIG* config
);
```

The `DlnGpioPortGetCfg()` function retrieves current configuration of the GPIO pins from the specified port.

Parameters:

handle
A handle to the DLN-series adapter.

port
A port to get configuration for.

config
A pointer to the DLN_GPIO_PORT_CONFIG structure, which will be filled with the configuration after function execution.

Return Values:

- `DLN_RES_SUCCESS` - the GPIO port configuration has been successfully retrieved.

Table 2.5. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.6. DlnGpioPortSetCfg()

```
DLN_RESULT DlnGpioPortSetCfg(
   HDLN handle,
   uint8_t port,
   uint8_t mask,
   uint16_t validFields,
   DLN_GPIO_PORT_CONFIG config
   uint16_t* conflictedPin
);
```

The `DlnGpioPortSetCfg()` function changes the configuration of the GPIO pins from the specified port and sets the conditions of the DLN_GPIO_CONDITION_MET_EV event generation. Each pin is configured individually via the DLN_GPIO_PIN_CONFIG structure.

The `mask` parameter allows a user to reconfigure all of the pins from the specified port or only some of them. This parameter is a byte value. Each of the eight bits, contained in the byte, corresponds to a pin of the port. The new configuration will be applied only to the pins with their `mask` bits set to 1. The configuration of pins with `mask` bits set to 0 will remain unchanged.

For example, if we only need to reconfigure the first and the fourth pin, we should set the `mask` byte as follows:

0	0	0	1	0	0	1	0

The `validFields` parameter is a bit field that defines the configuration parameters to be updated by this function. Each of the 16 validFields bits corresponds to a specific parameter in the DLN_GPIO_PIN_CONFIG structure. If you set the bit to 1, the new configuration parameter will be applied. If you set the bit to 0, the configuration parameter will remain unchanged regardless of its value in the DLN_GPIO_PIN_CONFIG structure. See DlnGpioPinSetCfg() for details.

Parameters:

handle
A handle to the DLN-series adapter.

port
A port, whose I/O lines are to be configured.

mask
A bit field that defines, which I/O lines are to be configured.

validFields
A bit field that defines valid DLN_GPIO_PIN_CONFIG fields.

config
A configuration to be set. See "DLN_GPIO_PORT_CONFIG structure" for details.

conflictedPin
A number of the conflicted pin, if any. The pins are numbered throughout the whole device. Numbers 0 to 7 belong to port **A**, 8 to 15 belong to port **B** etc.

Warning

In case there are several conflicted pins, only the number of the first one will be returned. As soon as a user fixes the problem, they should use the DlnGpioPortSetCfg() function once again, to see if there are any more conflicting pins.

Return Values

- `DLN_RES_SUCCESS` - the GPIO port configuration has been successfully set.

Table 2.6. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.7. DlnGpioPortGetVal()

```
DLN_RESULT DlnGpioPortGetVal(
   HDLN handle,
   uint8_t port,
   uint8_t* values
);
```

The `DlnGpioPortGetVal()` function retrieves current level on the I/O lines from the specified port. This function returns the level on an I/O line regardless of its configuration, whether it is configured as a GPIO input or output or even not connected to the GPIO module at all.

Parameters:

handle
A handle to the DLN-series adapter.

port
A port to get pin values from.

values
A pointer to an unsigned 8-bit integer. This integer will be filled with the pin values after function execution.

Return Values

- `DLN_RES_SUCCESS` - the GPIO port values have been successfully retrieved.

Table 2.7. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.8. DlnGpioPortSetOutVal()

```
DLN_RESULT DlnGpioPortSetOutVal(
   HDLN handle,
   uint8_t port,
   uint8_t mask,
   uint8_t values
);
```

The `DlnGpioPortSetOutVal()` function sets output values for I/O lines from the specified GPIO port. If a pin is an output, the value is applied immediately. If a pin is an input, the value is stored in an internal latch buffer. This value will be applied when the pin becomes output.

The `mask` parameter allows a user to change the output value for all of the pins from the port or only some of them. This parameter is a byte value. Each of the eight bits, contained in the byte, corresponds to a pin of the port. The new output values will be applied only to the pins with their mask bits set to 1. The output values of the pins with mask bits set to 0 will remain unchanged.

Parameters:

handle
A handle to the DLN-series adapter.

port
A port, whose output values are to be changed.

mask
A bit field that defines I/O lines, whose output values are to be changed.

values
Values to be set.

Return Values:

- `DLN_RES_SUCCESS` - pin output values for the specified GPIO port have been successfully applied.

Table 2.8. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.9. DlnGpioPortGetOutVal()

```
DLN_RESULT DlnGpioPortGetOutVal(
   HDLN handle,
```

```
    uint8_t port,
    uint8_t* values
);
```

The `DlnGpioPortGetOutVal()` retrieves current output values for the GPIO pins from the specified port. This function returns values, stored in an internal latch buffer. If the pin is configured as an output, the returned value matches the current I/O line level.

Parameters:

handle
A handle to the DLN-series adapter.

port
A port to get pin output values from.

values
A pointer to an unsigned 8-bit integer. This integer will be filled with pin output values after function execution.

Return Values:

- `DLN_RES_SUCCESS` - pin output values for the specified GPIO port have been successfully retrieved.

Table 2.9. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.10. DlnGpioSetDebounce()

```
DLN_RESULT DlnGpioSetDebounce(
    HDLN handle,
    uint32_t* duration
);
```

The `DlnGpioSetDebounce()` function specifies the minimum duration of the pulse to be registered (the Debounce interval). The duration is specified in µs from 1µs up to 4,294,967,295µs (~1h 10m). This value will be approximated (increased) as the closest debounce duration supported by the adapter. See Debounce Filter for details.

Parameters:

handle
A handle to the DLN-series adapter.

duration
A pointer to an unsigned 32-bit integer. This integer will be filled with the value, approximated (increased) as the closest debounce duration supported by the adapter after the function execution.

See Debounce Filter for details.

Return Values:

- `DLN_RES_SUCCESS` - the debounce interval has been successfully retrieved.

- `DLN_RES_COMMAND_NOT_SUPPORTED` - the adapter does not support debounce filtering.

Table 2.10. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Debounce interval range	1µs to 4,294,967,295µs	1µs to 4,294,967,295µs	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.11. DlnGpioGetDebounce()

```
DLN_RESULT DlnGpioGetDebounce(
   HDLN handle,
   uint32_t* duration
);
```

The `DlnGpioGetDebounce()` function retrieves the current setting of the minimum duration of the pulse to be registered (the Debounce interval).

Parameters:

handle
A handle to the DLN-series adapter.

duration
A pointer to an unsigned 32-bit integer. This integer will be filled with the debounce interval value after the function execution.

Return Values:

- `DLN_RES_SUCCESS` - the debounce interval has been successfully applied.
- `DLN_RES_COMMAND_NOT_SUPPORTED` - the adapter does not support debounce filtering.

Table 2.11. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Debounce interval range	1µs to 4,294,967,295µs	1µs to 4,294,967,295µs	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.12. DlnGpioPinGetOutVal()

```
DLN_RESULT DlnGpioPinGetOutVal(
   HDLN handle,
   uint8_t pin,
   uint8_t* value
);
```

The `DlnGpioPinGetOutVal()` function retrieves the pin output value, set using the DlnGpioPinSetOutVal() function and stored in the internal latch. If the pin is not an output, the value is taken from the latch.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A pin to get the output value from.

value
A pointer to an unsigned 8-bit integer. This integer will be filled with the pin output value after the function execution.

Return Values:

- `DLN_RES_SUCCESS` - the pin output value has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.12. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.13. DlnGpioPinSetOutVal()

```
DLN_RESULT DlnGpioPinSetOutVal(
   HDLN handle,
   uint8_t pin,
   uint8_t value
);
```

The `DlnGpioPinSetOutVal()` function sets the output value for the specified GPIO pin. If the pin is an output, the value is applied immediately. If the pins is an input, the value is stored in the internal latch. This value will be applied when the pin becomes output.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A pin to be configured.

value
A pin output value to be set.

Return Values:

- `DLN_RES_SUCCESS` - the pin output value has been successfully applied.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.13. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.14. DlnGpioPinGetVal()

```
DLN_RESULT DlnGpioPinGetVal(
   HDLN handle,
   uint8_t pin,
   uint8_t* value
);
```

The `DlnGpioPinGetVal()` function retrieves the current value on the specified GPIO pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A pin to get the value from.

value
A pointer to an unsigned 8-bit integer. This integer will be filled with the pin value after the function execution.

Return Values:

- `DLN_RES_SUCCESS` - the current pin value has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_PIN_NOT_CONNECTED_TO_MODULE` - the pin is not assigned to the GPIO module of the adapter and its value cannot be retrieved.

Table 2.14. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.15. DlnGpioPinEnable()

```
DLN_RESULT DlnGpioPinEnable(
   HDLN handle,
   uint8_t pin
);
```

The `DlnGpioPinEnable()` function configures a pin as general purpose input/output.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

Return Values:

- `DLN_RES_SUCCESS` - the pin has been successfully enabled.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_PIN_IN_USE` - the pin is assigned to another module of the adapter and cannot be enabled as GPIO.

Table 2.15. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.16. DlnGpioPinDisable()

```
DLN_RESULT DlnGpioPinDisable(
   HDLN handle,
   uint8_t pin
);
```

The `DlnGpioPinEnable()` function disables a pin as general purpose input/output.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

Return Values:

- `DLN_RES_SUCCESS` - the pin has been successfully disabled.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_PIN_IN_USE` - the pin is not assigned to GPIO module of the adapter and cannot be disabled as GPIO.

Table 2.16. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.17. DlnGpioPinIsEnabled()

```
DLN_RESULT DlnGpioPinIsEnabled(
   HDLN handle,
   uint8_t pin,
   uint8_t* enabled
);
```

The `DlnGpioPinIsEnabled()` function informs whether the pin is currently configured as general purpose input/output.

Parameters:

header
A handle to the DLN-series adapter.

pin
A number of the pin to retrieve information about.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with current pin configuration after the function execution. The following values are available:

- `1` - The pin is configured as GPIO.
- `0` - The pin is NOT configured as GPIO.

Return Values:

- `DLN_RES_SUCCESS` - the pin information has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.17. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.18. DlnGpioPinSetDirection()

```
DLN_RESULT DlnGpioPinSetDirection(
   HDLN handle,
   uint8_t pin,
   uint8_t direction
);
```

The `DlnGpioPinSetDirection()` function configures a pin as input or as output.

Parameters:

handle
A handle to the DLN-series adapter

pin
A number of the GPIO pin to be configured.

direction
Direction of a pin. Set to 0 for input or 1 for output.

Return Values:

- `DLN_RES_SUCCESS` - the pin direction has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.18. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.19. DlnGpioPinGetDirection()

```
DLN_RESULT DlnGpioPinGetDirection(
   HDLN handle,
   uint8_t pin,
   uint8_t* direction
);
```

The `DlnGpioPinGetDirection()` function retrieves current direction of a GPIO pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the GPIO pin to retrieve the information from.

direction
A pointer to an unsigned 8-bit integer. The integer will be filled with the pin current direction after the function execution. Possible values:

- `1` - The pin is an output.
- `0` - The pin is an input.

Return Values:

- `DLN_RES_SUCCESS` - the pin direction has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.19. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.20. DlnGpioPinOpendrainEnable()

```
DLN_RESULT DlnGpioPinOpendrainEnable(
   HDLN handle,
   uint8_t pin
);
```

The `DlnGpioPinOpendrainEnable()` function enables Open Drain mode for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

Return Values:

- `DLN_RES_SUCCESS` - the open drain mode has been successfully enabled.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.20. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Open drain capable pins	48	48	2 (Open drain only)
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.21. DlnGpioPinOpendrainDisable()

```
DLN_RESULT DlnGpioPinOpendrainDisable(
   HDLN handle,
   uint8_t pin
);
```

The `DlnGpioPinOpendrainDisable()` disables Open Drain mode for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

Return Values:

- `DLN_RES_SUCCESS` - the open drain mode has been successfully disabled.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.21. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Open drain capable pins	48	48	2 (Open drain only)
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.22. DlnGpioPinOpendrainIsEnabled()

```
DLN_RESULT DlnGpioPinOpendrainIsEnabled(
   HDLN handle,
   uint8_t pin,
   uint8_t* enabled
);
```

The `DlnGpioPinOpendrainIsEnabled()` function informs whether the pin output is currently configured as push pull or Open Drain.

Parameters:

header
A handle to the DLN-series adapter.

pin
A number of the pin to retrieve information about.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with current pin configuration after the function execution. The following values are available:

- `1` - The output is open drain.
- `0` - The output is push pull.

Return Values:

- `DLN_RES_SUCCESS` - the open drain mode has been successfully enabled.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.22. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Open drain capable pins	48	48	2 (Open drain only)
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.23. DlnGpioPinPullupEnable()

```
DLN_RESULT DlnGpioPinPullupEnable(
   HDLN handle,
   uint8_t pin
```

```
);
```

The `DlnGpioPinPullupEnable()` function activates an embedded pull-up resistor for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to enable a pull-up resistor for.

Return Values:

- `DLN_RES_SUCCESS` - the pull-up resistor has been successfully activated.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.23. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Pull-up resistors	48	48	30
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.24. DlnGpioPinPullupDisable()

```
DLN_RESULT DlnGpioPinPullupDisable(
   HDLN handle,
   uint8_t pin
);
```

The `DlnGpioPinPullupDisable()` deactivates an embedded pull-up resistor for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to disable a pull-up resistor for.

Return Values:

- `DLN_RES_SUCCESS` - the pull-up resistor has been successfully deactivated.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.24. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Pull-up resistors	48	48	30
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.25. DlnGpioPinPullupIsEnabled

```
DLN_RESULT DlnGpioPinPullupIsEnabled(
   HDLN handle,
   uint8_t pin,
   uint8_t* enabled
);
```

The `DlnGpioPinPullupIsEnabled()` function informs whether an embedded pull-up resistor is enabled for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with current state of a pull-up resistor after the function execution. The following values are available:

- `1` - The pull-up resistor is enabled.
- `0` - The pull-up resistor is disabled.

Return Values:

- `DLN_RES_SUCCESS` - the pull-up resistor has been successfully activated.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.25. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Pull-up resistors	48	48	30
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.26. DlnGpioPinDebounceEnable()

```
DLN_RESULT DlnGpioPinDebounceEnable(
   HDLN handle,
   uint8_t pin
);
```

The `DlnGpioPinDebounceEnable()` function enables debounce filtering for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

Return Values:

- `DLN_RES_SUCCESS` - the debounce filter has been successfully activated.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_COMMAND_NOT_SUPPORTED` - debounce filtering is not supported by the adapter.

Table 2.26. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.27. DlnGpioPinDebounceDisable()

```
DLN_RESULT DlnGpioPinDebounceDisable(
   HDLN handle,
   uint8_t pin
);
```

The `DlnGpioPinDebounceDisable()` disables debounce filtering for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

Return Values:

- `DLN_RES_SUCCESS` - the debounce filter has been successfully deactivated.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_COMMAND_NOT_SUPPORTED` - debounce filtering is not supported by the adapter.

Table 2.27. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.28. DlnGpioPinDebounceIsEnabled()

```
DLN_RESULT DlnGpioPinOpendrainIsEnabled(
   HDLN handle,
   uint8_t pin,
```

```
    uint8_t* enabled
);
```

The `DlnGpioPinDebounceIsEnabled()` function informs whether the debounce filtering is currently enabled for the specified pin.

Parameters:

header
A handle to the DLN-series adapter.

pin
A number of the pin to retrieve information about.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with current pin configuration after the function execution. The following values are available:

- `1` - Debounce filtering is enabled.
- `0` - Debounce filtering is disabled.

Return Values:

- `DLN_RES_SUCCESS` - the debounce filter state has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_COMMAND_NOT_SUPPORTED` - debounce filtering is not supported by the adapter.

Table 2.28. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.29. DlnGpioPinSetEventCfg()

```
DLN_RESULT DlnGpioPinSetEventCfg(
    HDLN handle,
    uint8_t pin,
    uint8_t eventType,
    uint16_t eventPeriod
);
```

The `DlnGpioPinSetEventCfg()` function configures the event generation conditions for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to configure the event generation conditions for.

eventType
Defines the condition for event generation for the pin. The following values are available:

- `0` or DLN_GPIO_EVENT_NONE - no events are generated for the current pin;
- `1` or DLN_GPIO_EVENT_CHANGE - events are generated when the level on the digital input line changes;
- `2` or DLN_GPIO_EVENT_LEVEL_HIGH - events are generated when high level (logical 1) is detected on the digital input line;
- `3` or DLN_GPIO_EVENT_LEVEL_LOW - events are generated when low level (logical 1) is detected on the digital input line;
- `4` or DLN_GPIO_EVENT_LEVEL_ALWAYS - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

For more detailed information see GPIO Events

eventPeriod
Defines the repeat interval for DLN_GPIO_CONDIION_MET_EV event generation on the pin. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the device will send a single event when the level on the line changes to meet the specified conditions.

Return Values:

- `DLN_RES_SUCCESS` - the event generation conditions were successfully configured.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_INVALID_EVENT_TYPE` - invalid event type was specified.
- `DLN_RES_INVALID_EVENT_PERIOD` - invalid event period was specified.

Table 2.29. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.8.30. DlnGpioPinGetEventCfg()

```
DLN_RESULT DlnGpioPinGetEventCfg(
   HDLN handle,
   uint8_t pin,
   uint8_t* eventType,
   uint16_t* eventPeriod
                       );
```

The `DlnGpioPinGetEventCfg()` function obtains current event generation conditions for the specified pin.

Parameters:

handle
A handle to the DLN-series adapter.

pin
A number of the pin to retrieve the information from.

eventType
A pointer to an unsigned 8-bit integer. The integer will be filled with currently set event generation conditions for the pin after the function execution.

eventPeriod
A pointer to an unsigned 8-bit integer. The integer will be filled with current event repeat interval after the function execution.

Return Values:

- `DLN_RES_SUCCESS` - the event generation conditions were successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Table 2.30. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9. Commands and Responses

2.9.1. DLN_GPIO_GET_PORT_COUNT

DLN_GPIO_GET_PORT_COUNT Command

Go to Response

The **DLN_GPIO_GET_PORT_COUNT** command is used to retrieve the number of GPIO ports available in your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_GPIO_GET_PORT_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_GET_PORT_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_GET_PORT_COUNT command it must be set to `0x0100`. You can use the `DLN_MSG_ID_GPIO_GET_PORT_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_GPIO_GET_PORT_COUNT Response

Go to Command

The adapter sends the **DLN_GPIO_GET_PORT_COUNT** response after the command execution. The response contains the available number of GPIO ports.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_GPIO_GET_PORT_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_GET_PORT_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_GET_PORT_COUNT response it is set to `0x0100`. The `DLN_MSG_ID_GPIO_GET_PORT_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of the GPIO ports has been successfully obtained.

count
Contains the available number of GPIO ports.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Total number of GPIO ports	6	6	4
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.2. DLN_GPIO_GET_PIN_COUNT

DLN_GPIO_GET_PIN_COUNT Command

Go to Response

The **DLN_GPIO_GET_PIN_COUNT** command is used to retrieve the total number of GPIO pins available in the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_GPIO_GET_PIN_COUNT_CMD;
```

Parameters:

header

Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_GET_PIN_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_GET_PIN_COUNT command it must be set to `0x0101`. You can use the `DLN_MSG_ID_GPIO_GET_PIN_COUNT` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_GPIO_GET_PIN_COUNT Response

Go to Command

The adapter sends the `DLN_GET_PIN_COUNT` response after the command execution. The response contains the available number of GPIO pins.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t count;
} __PACKED_ATTR DLN_GPIO_GET_PIN_COUNT_RSP;
```

Parameters:

header

Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_GET_PIN_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_GET_PIN_COUNT response it is set to `0x0101`. The `DLN_MSG_ID_GPIO_GET_PIN_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result

Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of pins has been successfully obtained.

count
Contains the available number of pins.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Total number of GPIO pins	48	48	32
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.3. DLN_GPIO_PIN_SET_CFG

DLN_GPIO_PIN_SET_CFG Command

Go to response

The **DLN_GPIO_PIN_SET_CFG** command is used to change the configuration of a single GPIO pin and set the conditions of DLN_GPIO_EVENT generation.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t pin;
   uint16_t validFields;
   DLN_GPIO_PIN_CONFIG config;
} __PACKED_ATTR DLN_GPIO_PIN_SET_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_SET_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_CFG command it must be set to `0x0109`. You can also use the `DLN_MSG_ID_GPIO_PIN_SET_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to be configured.

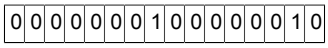
validFields
A bit field that defines the configuration parameters to be updated by this function. Each of the 16 validFields bits corresponds to a specific parameter in the DLN_GPIO_PIN_CONFIG structure. If you set the bit to 1, the new configuration parameter will be applied. If you set the bit to 0, the configuration parameter will remain unchanged regardless of its value in the DLN_GPIO_PIN_CONFIG structure. A user can also configure the pin parameters, using the constants, defined in the dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00005.html] file. If several constant are used, they should be separated with "| " (binary "or").

Several bits are reserved for future use and must be set to 0.

Bit	Corresponds to	Constant
0	Bit 0 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_ENABLE_BIT
1	Bit 1 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_OUTPUT_BIT
2	Bit 2 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_OUTPUT_VAL_BIT
3	Bit 3 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_OPEN_DRAIN_BIT
4	Bit 4 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_PULL_UP_BIT
5	Bit 5 of DLN_GPIO_PIN_CONFIG::cfg	DLN_GPIO_DEBOUNCE_BIT
6	reserved	
7	reserved	
8	DLN_GPIO_PIN_CONFIG::eventType	DLN_GPIO_EVENT_TYPE_BIT
9	DLN_GPIO_PIN_CONFIG::eventPeriod	DLN_GPIO_EVENT_PERIOD_BITT
10	reserved	
11	reserved	
12	reserved	
13	reserved	
14	reserved	
15	reserved	

In order to include a configuration field in the operation, set the corresponding bit to 1. If we set a bit to 0, the field will be ignored.

For example, if we only need to change isOutput and eventType settings, our `validFields` byte should look like this:



A user can configure the pin using the constants, defined in the dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00005.html] file. In this case, the `validFields` byte should look like this:

```
validFields = DLN_GPIO_OUTPUT_BIT | DLN_GPIO_EVENT_TYPE_BIT;
```

config
A configuration to be set. See "DLN_GPIO_PIN_CONFIG" structure for details.

DLN_GPIO_PIN_SET_CFG Response

Go to command

The adapter sends the **DLN_GPIO_PIN_SET_CFG** response after the command execution. The `result` field informs a user if the settings were successfully configured.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint_16t conflict
} __PACKED_ATTR DLN_GPIO_PIN_SET_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_SET_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_CFG response it is set to `0x0109`. The `DLN_MSG_ID_GPIO_PIN_SET_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the new configuration has been successfully applied;
- `DLN_RES_INVALID_HANDLE` - the specified handle is not valid;
- `DLN_RES_CONNECTION_LOST` - the connection to the DLN server was interrupted;
- `DLN_RES_INVALID_PIN_NUMBER` - the number of the pin is out of range. Use DlnGpioGetPinCount() function to get the number of pins available for the current DLN adapter;
- `DLN_RES_NON_ZERO_RESERVED_BIT` - one or more of the reserved bits in `validFields` or in config parameters are set to 1.

conflict
Contains the set of unsupported bits, which intersect with the `validFields` parameter.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.4. DLN_GPIO_PIN_GET_CFG

DLN_GPIO_PIN_GET_CFG Command

Go to response

The **DLN_GPIO_PIN_GET_CFG** command is used to retrieve the current configuration settings of a single GPIO pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_GET_CFG_CMD;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_GET_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_CFG command it must be set to `0x010A`. You can also use the `DLN_MSG_ID_GPIO_PIN_GET_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to get the configuration from.

DLN_GPIO_PIN_GET_CFG Response

Go to command

The adapter sends the **DLN_GPIO_PIN_GET_CFG** response after the command execution. The response contains current configuration of the specified GPIO pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   DLN_GPIO_PIN_CONFIG config;
} __PACKED_ATTR DLN_GPIO_PIN_GET_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_GET_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_CFG response it is set to `0x010A`. The `DLN_MSG_ID_GPIO_PIN_GET_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the configuration was successfully retrieved;
- `DLN_RES_INVALID_HANDLE` - the specified handle is not valid;
- `DLN_RES_CONNECTION_LOST` - the connection to the DLN server was interrupted;
- `DLN_RES_INVALID_PIN_NUMBER` - the specified pin number is not valid.

config
Contains the pin configuration DLN_GPIO_PIN_CONFIG

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.5. DLN_GPIO_PORT_SET_CFG

DLN_GPIO_PORT_SET_CFG Command

Go to response

The **DLN_GPIO_PORT_SET_CFG** command is used to change the configuration of the GPIO pins from the specified port and set the conditions of the DLN_GPIO_CONDITION_MET_EV event generation. Each pin is configured individually via the DLN_GPIO_PIN_CONFIG structure.

The `mask` parameter allows a user to reconfigure all of the pins from the specified port or only some of them. This parameter is a byte value. Each of the eight bits, contained in the byte, corresponds to a pin of the port. The new configuration will be applied only to the pins with their `mask` bits set to 1. The configuration of pins with `mask` bits set to 0 will remain unchanged.

For example, if we only need to reconfigure the first and the fourth pin, we should set the `mask` byte as follows:

0	0	0	1	0	0	1	0

The `validFields` parameter is a bit field that defines the configuration parameters to be updated by this function. Each of the 16 validFields bits corresponds to a specific parameter in the DLN_GPIO_PIN_CONFIG structure. If you set the bit to 1, the new configuration parameter will be applied. If you set the bit to 0, the configuration parameter will remain unchanged regardless of its value in the DLN_GPIO_PIN_CONFIG structure. See DlnGpioPinSetCfg() for details.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t mask;
   uint16_t validFields;
   DLN_GPIO_PORT_CONFIG config;
} __PACKED_ATTR DLN_GPIO_PORT_SET_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PORT_SET_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_SET_CFG command it must be set to `0x0102`. You can also use the `DLN_MSG_ID_GPIO_PORT_SET_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
A port whose pins will be configured.

mask
A bit field that defines, which I/O lines are to be configured. Each of the eight bits, contained in the byte, corresponds to a pin of the port. The new configuration will be applied only to the pins with their mask bits set to 1. The configuration of pins with mask bits set to 0 will remain unchanged.

validFields
A bit field that defines valid DLN_GPIO_PIN_CONFIG fields. See DlnGpioPinSetCfg() for details.

config
A configuration to be set. See DLN_GPIO_PORT_CONFIG structure for details.

DLN_GPIO_PORT_SET_CFG_RSP Response

Go to command

The adapter sends the **DLN_GPIO_PORT_SET_CFG** response after the command execution. The `result` field informs a user if the settings were successfully applied.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t result;
   uint16_t conflictedPin;
} DLN_GPIO_PORT_SET_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PORT_SET_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_SET_CFG response it is set to `0x0102`. The `DLN_MSG_ID_GPIO_PORT_SET_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` – settings were successfully configured;
- `DLN_RES_INVALID_COMMAND_SIZE` - the size, specified in the `size` field does not correspond to the actual size of the command;
- `DLN_RES_INVALID_PORT_NUMBER` - invalid port number specified;
- `DLN_RES_INVALID_EVENT_TYPE` - invalid event type specified;
- `DLN_RES_GPIO_PIN_IN_USE` – pin is used by another module. The number of the pin is contained in the `conflictedPin` field;

- `DLN_RES_PIN_NOT_CONNECTED_TO_MODULE` - pin is not connected to the GPIO module. The number of the pin is contained in the `conflictedPin` field;

conflictedPin
A number of the conflicted pin, if any. The pins are numbered throughout the whole device. Numbers 0 to 7 belong to port A, 8 to 15 belong to port B etc.

> **Warning**
>
> In case there are several conflicted pins, only the number of the first one will be returned. As soon as a user fixes the problem, they should send the DLN_GPIO_PORT_SET_CFG command once again, to see if there are any more conflicting pins.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.6. DLN_GPIO_PORT_GET_CFG

DLN_GPIO_PORT_GET_CFG Command

Go to response

The **DLN_GPIO_PORT_GET_CFG** command is used to retrieve the current configuration settings of a GPIO port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_GPIO_PORT_GET_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PORT_GET_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_GET_CFG command it must be set to `0x0103`. You can also use the `DLN_MSG_ID_GPIO_PORT_GET_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response to the command will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A port to get configuration from.

DLN_GPIO_PORT_GET_CFG Response

Go to command

The adapter sends the DLN_GPIO_PORT_GET_CFG response after the command execution. The response will contain the configuration settings.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t result;
   DLN_GPIO_PORT_CONFIG config;
} __PACKED_ATTR DLN_GPIO_PORT_GET_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PORT_GET_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_GET_CFG response it is set to 0x0103. The DLN_MSG_ID_GPIO_PORT_GET_CFG constant can be used.
- **echoCounter** - Links a command to a response. The echoCounter value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- DLN_RES_SUCCESS – configuration settings were successfully retrieved.

config
The port configuration DLN_GPIO_PORT_CONFIG.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.7. DLN_GPIO_SET_DEBOUNCE

DLN_GPIO_SET_DEBOUNCE Command

Go to response

The **DLN_GPIO_SET_DEBOUNCE** command is used to specify the minimum duration of the pulse to be registered (the Debounce interval).

```
typedef struct
{
```

```
    DLN_MSG_HEADER header;
    uint32_t duration;
} DLN_GPIO_SET_DEBOUNCE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_SET_DEBOUNCE_CMD structure.
- **msgId** - Defines the message. Must be set to `0x0104`. You can also use the `DLN_MSG_ID_GPIO_SET_DEBOUNCE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

duration
Minimum duration of the pulse to be registered. The duration is specified in µs from 1µs up to 4,294,967,295µs (~1h 10m). This value will be approximated as the closest debounce duration supported by the adapter. See Debounce Filter for details.

DLN_GPIO_SET_DEBOUNCE Response

Go to command

The adapter sends the **DLN_GPIO_SET_DEBOUNCE** response after the command execution. The response will contain the minimum pulse duration approximated as the closest to user-defined value.

```
typedef struct
{
    DLN_MSG_HEADER header;
    uint8_t result;
    uint32_t duration;
} DLN_GPIO_SET_DEBOUNCE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_SET_DEBOUNCE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_SET_DEBOUNCE response it is set to `0x0104`. The `DLN_MSG_ID_GPIO_SET_DEBOUNCE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` – debounce interval has been successfully set.
- `DLN_RES_COMMAND_NOT_SUPPORTED` – debounce filtering is not supported in the adapter.

duration
Contains minimum debounce interval approximated as the closest to user-defined value. The duration is specified in µs.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Debounce interval range	1µs to 4,294,967,295µs	1µs to 4,294,967,295µs	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.8. DLN_GPIO_GET_DEBOUNCE

DLN_GPIO_GET_DEBOUNCE Command

Go to response

The **DLN_GPIO_GET_DEBOUNCE** command is used to retrieve the current setting for the minimum duration of the pulse to be registered (the Debounce interval).

```
typedef struct
{
   DLN_MSG_HEADER header;
   } __PACKED_ATTR DLN_GPIO_GET_DEBOUNCE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_GET_DEBOUNCE_CMD structure.
- **msgId** - Defines the message. Must be set to `0x0105`. You can also use the `DLN_MSG_ID_GPIO_GET_DEBOUNCE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_GPIO_GET_DEBOUNCE Response

Go to command

The adapter sends the **DLN_GPIO_GET_DEBOUNCE** response after the command execution. The response will contain the curretly set debounce interval.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t duration;
} __PACKED_ATTR DLN_GPIO_GET_DEBOUNCE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_GET_DEBOUNCE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_GET_DEBOUNCE response it is set to `0x0105`. The `DLN_MSG_ID_GPIO_SET_DEBOUNCE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` – the current debounce interval has been successfully retrieved.
- `DLN_RES_COMMAND_NOT_SUPPORTED` – debounce filtering is not supported in the adapter.

duration
Contains minimum debounce interval approximated as the closest to user-defined value. The duration is specified in µs.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Debounce interval range	1µs to 4,294,967,295µs	1µs to 4,294,967,295µs	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.9. DLN_GPIO_PORT_SET_OUT_VAL

DLN_GPIO_PORT_SET_OUT_VAL Command

Go to response

The **DLN_GPIO_PORT_SET_OUT_VAL** command is used to set the output values for the specified GPIO port pins. If the pins are outputs, the values are applied immediately. If the pins are inputs, the values are stored in internal latches. These values will be applied when the pins become outputs.

```
typedef struct
{
   DLN_MSG_HEADER header;
```

```
    uint8_t port;
    uint8_t mask;
    uint8_t values;
} __PACKED_ATTR DLN_GPIO_PORT_SET_OUT_VAL_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PORT_SET_OUT_VAL_CMD structure.
- **msgId** - Defines the message. Must be set to `0x0107`. You can also use the `DLN_MSG_ID_GPIO_PORT_SET_OUT_VAL` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A port whose pins will be configured.

mask
A bit field that defines, which I/O lines are to be configured. Each of the eight bits, contained in the byte, corresponds to a pin of the port. The new configuration will be applied only to the pins with their mask bits set to 1. The configuration of pins with mask bits set to 0 will remain unchanged.

values
A bit field, which contains new port output values to be set. Note, that output values will be applied only for the pins with `mask` bits set to 1.

DLN_GPIO_PORT_SET_OUT_VAL Response

Go to command

The adapter sends the`DLN_GPIO_PORT_SET_OUT_VAL` response after the command execution. The `result` field informs a user if the settings were successfully configured.

```
typedef struct
{
    DLN_MSG_HEADER header;
    uint8_t result;
} __PACKED_ATTR DLN_GPIO_PORT_SET_OUT_VAL_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PORT_SET_OUT_VAL_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_SET_OUT_VAL response it is set to `0x0107`. The `DLN_MSG_ID_GPIO_PORT_SET_OUT_VAL` constant can be used.

- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` – output values were successfully set.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.10. DLN_GPIO_PORT_GET_VAL

DLN_GPIO_GET_PORT_VAL Command

Go to response

The **DLN_GPIO_PORT_GET_VAL** command is used to retrieve current values on the GPIO pins.

```
typedef struct
{
DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_GPIO_PORT_GET_VAL_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending a command, the user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PORT_GET_VAL_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_GET_VAL command it must be set to `0x0106`. You can also use the `DLN_MSG_ID_GPIO_PORT_GET_VAL` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A port to get the pin values from.

DLN_GPIO_PORT_GET_VAL Response

Go to command

The adapter sends the **DLN_GPIO_PORT_GET_VAL** response after the command execution. The response contains current values on the pins.

```
typedef struct
{
DLN_MSG_HEADER header;
uint8_t result;
uint8_t port;
   uint8_t values;
} __PACKED_ATTR DLN_GPIO_PORT_GET_VAL_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PORT_GET_VAL_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_GET_VAL response it is set to `0x0106`. The `DLN_MSG_ID_GPIO_PORT_GET_VAL` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` – values were successfully retrieved.

port
A port to get the pin values from.

values
A bit field, which contains the current pin values for the port.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.11. DLN_GPIO_PORT_GET_OUT_VAL

DLN_GPIO_PORT_GET_OUT_VAL Command

Go to response

The **DLN_GPIO_PORT_GET_OUT_VAL** command retrieves pin output values, set using the DLN_GPIO_PORT_SET_OUT_VAL command and stored in internal latches. If the pin is not an output, the value is taken from the latch.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
```

```
} __PACKED_ATTR DLN_GPIO_PORT_GET_OUT_VAL_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending a command, the user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PORT_GET_OUT_VAL_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_GET_OUT_VAL command it must be set to `0x0107`. You can use the `DLN_MSG_ID_GPIO_PORT_GET_OUT_VAL` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A port to get the pin output values from.

DLN_GPIO_PORT_GET_OUT_VAL Response

Go to command

The adapter sends the **DLN_GPIO_PORT_GET_OUT_VAL** response after the command execution. The response contains output values on the pins.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t port;
   uint8_t values;
} __PACKED_ATTR DLN_GPIO_PORT_GET_OUT_VAL_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PORT_GET_OUT_VAL_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PORT_GET_OUT_VAL response it is set to `0x0107`. The `DLN_MSG_ID_GPIO_PORT_GET_OUT_VAL` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` – pin output values were successfully retrieved.

port
A port to get the pin output values from.

values
A bit field, which contains the pin output values for the port.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.12. DLN_GPIO_PIN_SET_OUT_VAL

DLN_GPIO_PIN_SET_OUT_VAL Command

Go to response

The **DLN_GPIO_PIN_SET_OUT_VAL** command is used to set the output value for the specified GPIO pin. If the pin is an output, the value is applied immediately. If the pins is an input, the value is stored in the internal latch. This value will be applied when the pin becomes output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
   uint8_t value;
} __PACKED_ATTR DLN_GPIO_PIN_SET_OUT_VAL_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_SET_OUT_VAL_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_OUT_VAL command, it must be set to `0x010C`. You can use the `DLN_MSG_ID_GPIO_PIN_SET_OUT_VAL` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to be configured.

value
A pin output value to be set.

DLN_GPIO_PIN_SET_OUT_VAL Response

Go to command

The adapter sends the **DLN_GPIO_PIN_SET_OUT_VAL** response after the command execution. The result field informs a user if the setting was successfully configured.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_SET_OUT_VAL_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_SET_OUT_VAL_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_OUT_VAL response it is set to `0x010C`. The `DLN_MSG_ID_GPIO_PIN_SET_OUT_VAL` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the output value has been successfully set.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.13. DLN_GPIO_PIN_GET_VAL

DLN_GPIO_PIN_GET_VAL Command

Go to response

The **DLN_GPIO_PIN_GET_VAL** command is used to retrieve the current value on the specified GPIO pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
```

```
} __PACKED_ATTR DLN_GPIO_PIN_GET_VAL_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_GET_VAL_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_VAL command, it must be set to `0x010B`. You can use the `DLN_MSG_ID_GPIO_PIN_GET_VAL` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to get the value from.

DLN_GPIO_GET_PIN_VAL Response

Go to command

The adapter sends the **DLN_GPIO_PIN_GET_VAL** response after the command execution. The response contains the current value on the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t value;
} __PACKED_ATTR DLN_GPIO_PIN_GET_VAL_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_GET_VAL_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_VAL response it is set to `0x010B`. The `DLN_MSG_ID_GPIO_PIN_GET_VAL` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the value was successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_PIN_NOT_CONNECTED_TO_MODULE` - the pin is not assigned to the GPIO module of the adapter and its value cannot be retrieved.

pin
A pin to get the value from.

value
The current value on the pin.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.14. DLN_GPIO_PIN_GET_OUT_VAL

DLN_GPIO_PIN_GET_OUT_VAL Command

Go to response

The **DLN_GPIO_PIN_GET_OUT_VAL** command retrieves the pin output value, set using the DLN_GPIO_PIN_SET_OUT_VAL command and stored in the internal latch. If the pin is not an output, the value is taken from the latch.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_GET_OUT_VAL_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_GET_OUT_VAL_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_OUT_VAL command, it must be set to `0x010D`. You can use the `DLN_MSG_ID_GPIO_PIN_GET_OUT_VAL` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to get the output value from.

DLN_GPIO_PIN_GET_OUT_VAL Response

Go to command

The adapter sends the **DLN_GPIO_PIN_GET_OUT_VAL** response after the command execution. The response contains the output value on the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t value;
} __PACKED_ATTR DLN_GPIO_PIN_GET_OUT_VAL_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_GET_OUT_VAL_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_OUT_VAL response it is set to `0x010D`. The `DLN_MSG_ID_GPIO_PIN_GET_OUT_VAL` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the pin output value has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

pin
A pin to get the output value from.

value
The output value on the specified pin.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.15. DLN_GPIO_PIN_ENABLE

DLN_GPIO_PIN_ENABLE Command

Go to response

The **DLN_GPIO_PIN_ENABLE** command is used to configure a pin as general purpose input/output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_ENABLE_CMD command, it must be set to `0x0110`. You can use the `DLN_MSG_ID_GPIO_PIN_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_ENABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_ENABLE** response after the command execution. The `result` field informs a user if the pin has been successfully configured as general purpose input/output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_ENABLE_RSP response it is set to `0x0110`. The `DLN_MSG_ID_GPIO_PIN_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the pin has been successfully configured as general purpose input/output.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_PIN_IN_USE` - the pin is assigned to another module of the adapter and cannot be enabled as GPIO.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.16. DLN_GPIO_PIN_DISABLE

DLN_GPIO_PIN_DISABLE Command

Go to response

The **DLN_GPIO_PIN_DISABLE** command is used to disable a pin as general purpose input/output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DISABLE_CMD command, it must be set to `0x0111`. You can use the `DLN_MSG_ID_GPIO_PIN_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_DISABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_DISABLE** response after the command execution. The `result` field informs a user if the pin has been successfully disabled as general purpose input/output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DISABLE_RSP response it is set to `0x0111`. The `DLN_MSG_ID_GPIO_PIN_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the pin has been successfully disabled as general purpose input/output.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_PIN_IN_USE` - the pin is assigned to another module of the adapter and cannot be disabled as GPIO.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.17. DLN_GPIO_PIN_IS_ENABLED

DLN_GPIO_PIN_IS_ENABLED Command

Go to response

The **DLN_GPIO_PIN_IS_ENABLED** command is used to retrieve information on whether the specified pin is currently configured as general purpose input/output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_IS_ENABLED_CMD command, it must be set to `0x0112`. You can use the `DLN_MSG_ID_GPIO_PIN_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to retrieve the information from.

DLN_GPIO_PIN_IS_ENABLED Response

Go to command

The adapter sends the **DLN_GPIO_PIN_IS_ENABLED** response after the command execution. The response informs a user whether the specified pin is configured as general purpose input/output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t enabled;
} __PACKED_ATTR DLN_GPIO_PIN_IS_ENABLED_RSP;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_IS_ENABLED_RSP response it is set to `0x0112`. The `DLN_MSG_ID_GPIO_PIN_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the pin configuration has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

pin
Contains the number of the pin.

enabled
Contains the current configuration of the pin. The following values are available:

- `1` - the pin is configured as general purpose input/output;
- `0` - the pin is not configured as general purpose input/output.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.18. DLN_GPIO_PIN_SET_DIRECTION

DLN_GPIO_PIN_SET_DIRECTION Command

Go to response

The **DLN_GPIO_PIN_SET_DIRECTION** command is used to configure the pin as input or as output.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
   uint8_t direction;
} __PACKED_ATTR DLN_GPIO_PIN_SET_DIRECTION_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_SET_DIRECTION_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_DIRECTION_CMD command, it must be set to `0x0113`. You can use the `DLN_MSG_ID_GPIO_PIN_SET_DIRECTION` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to set direction for.

direction
Must contain the new direction of the pin. The following values are available:

- `1` - configure the pin as output;

- `0` - configure the pin as input.

DLN_GPIO_PIN_SET_DIRECTION Response

Go to command

The adapter sends the **DLN_GPIO_PIN_SET_DIRECTION** response after the command execution. The `result` field informs a user if the pin direction has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_SET_DIRECTION_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_SET_DIRECTION_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_DIRECTION_RSP response it is set to `0x0113`. The `DLN_MSG_ID_GPIO_PIN_SET_DIRECTION` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the pin direction has been successfully set.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.19. DLN_GPIO_PIN_GET_DIRECTION

DLN_GPIO_PIN_GET_DIRECTION Command

Go to response

The **DLN_GPIO_PIN_GET_DIRECTION** command is used to retrieve currently set direction of the pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
```

```
} __PACKED_ATTR DLN_GPIO_PIN_GET_DIRECTION_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_GET_DIRECTION_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_DIRECTION_CMD command, it must be set to `0x0114`. You can use the `DLN_MSG_ID_GPIO_PIN_GET_DIRECTION` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to retrieve the information from.

DLN_GPIO_PIN_GET_DIRECTION Response

Go to command.

The adapter sends the **DLN_GPIO_PIN_GET_DIRECITON** response after the command execution. The response contains currently selected direction of the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t direction;
} __PACKED_ATTR DLN_GPIO_PIN_GET_DIRECTION_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_GET_DIRECTION_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_DIRECTION_RSP response it is set to `0x0114`. The `DLN_MSG_ID_GPIO_PIN_GET_DIRECTION` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the current pin direction has been successfully retrieved.

- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

pin
Contains a number of the pin.

direction
Contains the currently set direction of the specified pin. The following values are available:

- `1` - the pin is configured as output;
- `0` - the pin is configured as input.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.20. DLN_GPIO_PIN_OPENDRAIN_ENABLE

DLN_GPIO_PIN_OPENDRAIN_ENABLE Command

Go to response

The **DLN_GPIO_PIN_OPENDRAIN_ENABLE** command is used to enable Open Drain mode for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_OPENDRAIN_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_OPENDRAIN_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_OPENDRAIN_ENABLE_CMD command, it must be set to `0x0115`. You can use the `DLN_MSG_ID_GPIO_PIN_OPENDRAIN_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_OPENDRAIN_ENABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_OPENDRAIN_ENABLE** response after the command execution. The `result` field informs a user if the Open Drain mode has been successfully enabled for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_OPENDRAIN_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_OPENDRAIN_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_OPENDRAIN_ENABLE_RSP response it is set to `0x0115`. The `DLN_MSG_ID_GPIO_PIN_OPENDRAIN_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the Open Drain mode has been successfully enabled for the specified pin.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Open drain capable pins	48	48	2 (Open drain only)
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.21. DLN_GPIO_PIN_OPENDRAIN_DISABLE

DLN_GPIO_PIN_OPENDRAIN_DISABLE Command

Go to response

The **DLN_GPIO_PIN_OPENDRAIN_DISABLE** command is used to disable the Open Drain mode for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
```

```
} __PACKED_ATTR DLN_GPIO_PIN_OPENDRAIN_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_OPENDRAIN_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_OPENDRAIN_DISABLE_CMD command, it must be set to `0x0116`. You can use the `DLN_MSG_ID_GPIO_PIN_OPENDRAIN_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_OPENDRAIN_DISABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_OPENDRAIN_DISABLE** response after the command execution. The `result` field informs a user if the Open Drain mode has been successfully disabled for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_OPENDRAIN_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_OPENDRAIN_DISABLE_RSP response it is set to `0x0116`. The `DLN_MSG_ID_GPIO_PIN_OPENDRAIN_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the Open Drain mode has been successfully disabled for the specified pin.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Open drain capable pins	48	48	2 (Open drain only)
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.22. DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED

DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED Command

Go to response

The **DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED** command is used to retrieve information on whether the Open Drain mode is currently enabled for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED_CMD command, it must be set to `0x0117`. You can use the `DLN_MSG_ID_GPIO_PIN_OPENDRAIN_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to retrieve the information from.

DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED Response

Go to command

The adapter sends the **DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED** response after the command execution. The response informs a user whether the Open Drain mode is currently enabled for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t enabled;
} __PACKED_ATTR DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED_RSP;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_OPENDRAIN_IS_ENABLED_RSP response it is set to `0x0117`. The `DLN_MSG_ID_GPIO_PIN_OPENDRAIN_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the current setting for the Open Drain mode for the specified pin has been successfully retrieved.
- `DLN_RES_INVALID_PIN` - invalid pin number was specified.

pin
Contains the number of the pin.

enabled
Contains the current configuration of the pin. The following values are available:

- `1` - the Open Drain mode is currently enabled for the specified pin;
- `0` - the Open Drain mode is currently disabled for the specified pin.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Open drain capable pins	48	48	2 (Open drain only)
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.23. DLN_GPIO_PIN_PULLUP_ENABLE

DLN_GPIO_PIN_PULLUP_ENABLE Command

Go to response

The **DLN_GPIO_PIN_PULLUP_ENABLE** command is used to enable embedded pull-up resistor for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_PULLUP_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_PULLUP_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_PULLUP_ENABLE_CMD command, it must be set to `0x0118`. You can use the `DLN_MSG_ID_GPIO_PIN_PULLUP_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_PULLUP_ENABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_PULLUP_ENABLE** response after the command execution. The `result` field informs a user if a pull-up resistor has been successfully activated for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_PULLUP_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_PULLUP_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_PULLUP_ENABLE_RSP response it is set to `0x0118`. The `DLN_MSG_ID_GPIO_PIN_PULLUP_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.

- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the embedded pull-up resistor has been successfully activated for the specified pin.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Pull-up resistors	48	48	30
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.24. DLN_GPIO_PIN_PULLUP_DISABLE

DLN_GPIO_PIN_PULLUP_DISABLE Command

Go to response

The **DLN_GPIO_PIN_PULLUP_DISABLE** command is used to disable the embedded pull-up resistor for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_PULLUP_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_PULLUP_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_PULLUP_DISABLE_CMD command, it must be set to `0x0119`. You can use the `DLN_MSG_ID_GPIO_PIN_PULLUP_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_PULLUP_DISABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_PULLUP_DISABLE** response after the command execution. The `result` field informs a user if the embedded pull-up resistor has been successfully disabled for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_PULLUP_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_PULLUP_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_PULLUP_DISABLE_RSP response it is set to `0x0119`. The `DLN_MSG_ID_GPIO_PIN_PULLUP_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the embedded pull-up resistor has been successfully deactivated for the specified pin.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Pull-up resistors	48	48	30
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.25. DLN_GPIO_PIN_PULLUP_IS_ENABLED

DLN_GPIO_PIN_PULLUP_IS_ENABLED Command

Go to response

The **DLN_GPIO_PIN_PULLUP_IS_ENABLED** command is used to retrieve information on whether the embedded pull-up resistor is currently activated for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
```

```
} __PACKED_ATTR DLN_GPIO_PIN_PULLUP_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_PULLUP_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_PULLUP_IS_ENABLED_CMD command, it must be set to `0x011A`. You can use the `DLN_MSG_ID_GPIO_PIN_PULLUP_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to retrieve the information from.

DLN_GPIO_PIN_PULLUP_IS_ENABLED Response

Go to command

The adapter sends the **DLN_GPIO_PIN_PULLUP_IS_ENABLED** response after the command execution. The response informs a user whether the embedded pull-up resistor is currently enabled for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t enabled;
} __PACKED_ATTR DLN_GPIO_PIN_PULLUP_IS_ENABLED_RSP;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_PULLUP_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_PULLUP_IS_ENABLED_RSP response it is set to `0x011A`. The `DLN_MSG_ID_GPIO_PIN_PULLUP_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the current setting for the embedded pull-up resistor for the specified pin has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

pin
Contains the number of the pin.

enabled
Contains the current configuration of the pin. The following values are available:

- `1` - the embedded pull-up resistor is currently enabled for the specified pin;
- `0` - the embedded pull-up resistor is currently disabled for the specified pin.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Pull-up resistors	48	48	30
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.26. DLN_GPIO_PIN_DEBOUNCE_ENABLE

DLN_GPIO_PIN_DEBOUNCE_ENABLE Command

Go to response

The **DLN_GPIO_PIN_PULLUP_ENABLE** command is used to enable Debounce Filtering for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_DEBOUNCE_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_DEBOUNCE_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DEBOUNCE_ENABLE_CMD command, it must be set to `0x011B`. You can use the `DLN_MSG_ID_GPIO_PIN_DEBOUNCE_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_DEBOUNCE_ENABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_DEBOUNCE_ENABLE** response after the command execution. The `result` field informs a user if Debounce Filtering has been successfully activated for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_DEBOUNCE_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_DEBOUNCE_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DEBOUNCE_ENABLE_RSP response it is set to `0x011B`. The `DLN_MSG_ID_GPIO_PIN_DEBOUNCE_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Debounce Filtering has been successfully activated for the specified pin.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_COMMAND_NOT_SUPPORTED` - debounce filtering is not supported by the adapter.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.27. DLN_GPIO_PIN_DEBOUNCE_DISABLE

DLN_GPIO_PIN_DEBOUNCE_DISABLE Command

Go to response

The **DLN_GPIO_PIN_DEBOUNCE_DISABLE** command is used to disable Debounce Filtering for the specified pin.

```
typedef struct
{
```

```
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_DEBOUNCE_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_DEBOUNCE_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DEBOUNCE_DISABLE_CMD command, it must be set to `0x011C`. You can use the `DLN_MSG_ID_GPIO_PIN_DEBOUNCE_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to be configured.

DLN_GPIO_PIN_DEBOUNCE_DISABLE Response

Go to command

The adapter sends the **DLN_GPIO_PIN_DEBOUNCE_DISABLE** response after the command execution. The `result` field informs a user if Debounce Filtering has been successfully deactivated for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_DEBOUNCE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_DEBOUNCE_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DEBOUNCE_DISABLE_RSP response it is set to `0x011C`. The `DLN_MSG_ID_GPIO_PIN_DEBOUNCE_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Debounce Filtering has been successfully deactivated for the specified pin.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_COMMAND_NOT_SUPPORTED` - debounce filtering is not supported by the adapter.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.28. DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED

DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED Command

Go to response

The **DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED** command is used to retrieve information on whether Debounce Filtering is currently activated for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED_CMD command, it must be set to `0x011D`. You can use the `DLN_MSG_ID_GPIO_PIN_DEBOUNCE_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A pin to retrieve the information from.

DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED Response

Go to command

The adapter sends the **DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED** response after the command execution. The response informs a user whether Debounce Filtering is currently enabled for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t enabled;
} __PACKED_ATTR DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED_RSP;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_DEBOUNCE_IS_ENABLED_RSP response it is set to `0x011D`. The `DLN_MSG_ID_GPIO_PIN_DEBOUNCE_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the current setting for Debounce Filtering for the specified pin has been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_COMMAND_NOT_SUPPORTED` - debounce filtering is not supported by the adapter.

pin
Contains the number of the pin.

enabled
Contains the current configuration of the pin. The following values are available:

- `1` - Debounce Filtering is currently activated for the specified pin;
- `0` - Debounce Filtering is currently deactivated for the specified pin.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✖
Required firmware	All versions	All versions	N/A
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.29. DLN_GPIO_PIN_SET_EVENT_CFG

DLN_GPIO_PIN_SET_EVENT_CFG Command

Go to response

The **DLN_GPIO_PIN_SET_EVENT_CFG** command is used to configure the event generation conditions for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t pin;
   uint8_t eventType;
   uint16_t eventPeriod;
} __PACKED_ATTR DLN_GPIO_PIN_SET_EVENT_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_SET_EVENT_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_EVENT_CFG_CMD command, it must be set to `0x011E`. You can use the `DLN_MSG_ID_GPIO_PIN_SET_EVENT_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to set direction for.

eventType
Must contain the condition for event generation for the pin. The following values are available:

- `0` or DLN_GPIO_EVENT_NONE - no events are generated for the current pin;
- `1` or DLN_GPIO_EVENT_CHANGE - events are generated when the level on the digital input line changes;
- `2` or DLN_GPIO_EVENT_LEVEL_HIGH - events are generated when high level (logical 1) is detected on the digital input line;
- `3` or DLN_GPIO_EVENT_LEVEL_LOW - events are generated when low level (logical 1) is detected on the digital input line;
- `4` or DLN_GPIO_EVENT_LEVEL_ALWAYS - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

For more detailed information see GPIO Events

eventPeriod
Must contain the repeat interval for DLN_GPIO_CONDITION_MET_EV event generation on the pin. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the device will send a single event when the level on the line changes to meet the specified conditions.

DLN_GPIO_PIN_SET_EVENT_CFG Response

Go to command

The adapter sends the **DLN_GPIO_PIN_SET_EVENT_CFG** response after the command execution. The `result` field informs a user if the event generation conditions for the pin have been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_GPIO_PIN_SET_EVENT_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_SET_EVENT_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_SET_EVENT_CFG_RSP response it is set to `0x011E`. The `DLN_MSG_ID_GPIO_PIN_SET_EVENT_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the event generation conditions for the specified pin have been successfully set.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.
- `DLN_RES_INVALID_EVENT_TYPE` - invalid event type was specified.
- `DLN_RES_INVALID_EVENT_PERIOD` - invalid event perios was specified.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.9.30. DLN_GPIO_PIN_GET_EVENT_CFG

DLN_GPIO_PIN_GET_EVENT_CFG Command

Go to response

The **DLN_GPIO_PIN_GET_EVENT_CFG** command is used to retrieve currently set event generation conditions for the specified pin.

```
typedef struct
{
```

```
   DLN_MSG_HEADER header;
   uint8_t pin;
} __PACKED_ATTR DLN_GPIO_PIN_GET_EVENT_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_PIN_GET_EVENT_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_EVENT_CFG_CMD command, it must be set to `0x011F`. You can use the `DLN_MSG_ID_GPIO_PIN_GET_EVENT_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

pin
A number of the pin to retrieve the information from.

DLN_GPIO_PIN_GET_EVENT_CFG Response

Go to command

The adapter sends the **DLN_GPIO_PIN_GET_EVENT_CFG** response after the command execution. The response contains currently set event generation conditions for the specified pin.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t pin;
   uint8_t eventType;
   uint16_t eventPeriod;
} __PACKED_ATTR DLN_GPIO_PIN_GET_EVENT_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_GPIO_PIN_GET_EVENT_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_PIN_GET_EVENT_CFG_RSP response it is set to `0x011F`. The `DLN_MSG_ID_GPIO_PIN_GET_EVENT_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

pin
Contains a number of the pin.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the currently set event generation conditions for the pin have been successfully retrieved.
- `DLN_RES_INVALID_PIN_NUMBER` - invalid pin number was specified.

eventType
Contains the event generation condition for the pin. The following values are available:

- `0` or DLN_GPIO_EVENT_NONE - no events are generated for the current pin;
- `1` or DLN_GPIO_EVENT_CHANGE - events are generated when the level on the digital input line changes;
- `2` or DLN_GPIO_EVENT_LEVEL_HIGH - events are generated when high level (logical 1) is detected on the digital input line;
- `3` or DLN_GPIO_EVENT_LEVEL_LOW - events are generated when low level (logical 1) is detected on the digital input line;
- `4` or DLN_GPIO_EVENT_LEVEL_ALWAYS - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

For more detailed information see GPIO Events

eventPeriod
Contains the repeat interval for DLN_GPIO_CONDIION_MET_EV event generation on the pin. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the device will send a single event when the level on the line changes to meet the specified conditions.

Summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✔	✔	✔
Required firmware	All versions	All versions	All versions
Defined in	dln_gpio.h [http://www.diolan.com/src/dln/common_c/a00003.html]		

2.10. Events

The adapter may send `DLN_GPIO_EVENT` event with current values on specified pins. The conditions of the event generation are configured with DLN_GPIO_PORT_SET_CFG command.

2.10.1. DLN_GPIO_CONDITION_MET

The structure of the `DLN_GPIO_CONDITION_MET` event is as follows:

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t eventCount;
   uint8_t eventType;
   uint8_t pin;
   uint8_t value;
```

```
} __PACKED_ATTR DLN_GPIO_CONDITION_MET_EV;
```

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages.

eventCount
Contains the number of generated events since the previous level change, except for the DLN_GPIO_EVENT_ALWAYS event. In this case, the parameter contains the number of events, sent after configuration setting.

eventType
Specifies the type of event to be generated.

pin
Contains the number of the pin which caused the event generation.

value
Contains current value of the pin, which caused the event generation. The following values are available:

- `1` – logical 1;
- `0` – logical 0.

Chapter 3. ADC Module

3.1. Overview

3.2. Structures

3.2.1. DLN_ADC_CONDITION_MET_EV

This structure is used to store descriptions of ADC events.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t eventCount;
   uint8_t port;
   uint8_t channel;
   uint16_t value;
   uint8_t eventType;
} __PACKED_ATTR DLN_ADC_CONDITION_MET_EV;
```

Members:

header
Defines the DLN message header DLN_MSG_HEADER. This structure's header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_CONDITION_MET_EV structure.
- **msgId** - Defines the message. For the DLN_ADC_CONDITION_MET_EV response it is set to `0x0610`. The `DLN_MSG_ID_ADC_CONDITION_MET_EV` constant can be used.
- **echoCounter** - .
- **handle** - A handle to the DLN-series adapter.

eventCount
Contains the number of generated events after configuration setting.

port
Contains the number of the ADC port.

channel
Contains the number of the ADC channel.

value
Contains the voltage level on the corresponding ADC channel.

eventType
Contains the condition that has triggered the event generation. The following values are available:

- `0` or `DLN_ADC_EVENT_NONE` - no events are generated for the current channel;
- `1` or `DLN_ADC_EVENT_BELOW` - events are generated when voltage level on the ADC channel crosses the lower threshhold;

- 2 or DLN_ADC_EVENT_LEVEL_ABOVE - events are generated when voltage level on the ADC channel crosses the higher threshold;
- 3 or DLN_ADC_EVENT_OUTSIDE - events are generated when voltage level on the ADC channel falls outside of the specified range between thresholds;
- 4 or DLN_ADC_EVENT_INSIDE - events are generated when voltage level on the ADC channel falls within the specified range between thresholds;
- 5 or DLN_ADC_EVENT_ALWAYS - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

3.3. Functions

3.3.1. DlnAdcGetPortCount()

```
DLN_RESULT DlnAdcGetPortCount(
HDLN handle,
uint8_t* count
);
```

The DlnAdcGetPortCount() function retrieves the number of ADC ports available in your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. The integer will be filled with the number of available ADC ports after the function execution.

3.3.2. DlnAdcGetChannelCount()

```
DLN_RESULT DlnAdcGetChannelCount(
HDLN handle,
uint8_t port,
uint8_t* count
);
```

The DlnAdcGetChannelCount() function retrieves the number of ADC channels, available in the specified ADC-port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
An ADC port to retrieve the number of channels from.

count
A pointer to an unsigned 8-bit integer. The integer will be filled with the available number of channels in the specified ADC port of the DLN-series adapter.

3.3.3. DlnAdcEnable()

```
DLN_RESULT DlnAdcEnable(
HDLN handle,
uint8_t port,
uint16_t* conflict
);
```

The `DlnAdcEnable()` function activates the corresponding ADC port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to be activated.

conflict
A pointer to an unsigned 16-bit integer. The integer will be filled with the number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used for the ADC module. To fix this a user has to disconnect a pin from a module that it has been assigned to and call the DlnAdcEnable() function once again. In case there still are conflicted pins, only the number of the next one will be returned.

3.3.4. DlnAdcDisable()

```
DLN_RESULT DlnAdcDisable(
HDLN handle,
uint8_t port
);
```

The `DlnAdcDisable()` function deactivates the specified ADC port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to be deactivated.

3.3.5. DlnAdcIsEnabled()

```
DLN_RESULT DlnAdcIsEnabled(
HDLN handle,
uint8_t port,
uint8_t* enabled
);
```

The `DlnAdcIsEnabled()` function retrieve information, whether the specified ADC port is activated.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with the information whether the specified ADC port is activated. There are two possible values:

- `0` or `DLN_ADC_DISABLED` - the ADC port is deactivated.
- `1` or `DLN_ADC_ENABLED` - the ADC port is activated.

3.3.6. DlnAdcChannelEnable()

```
DLN_RESULT DlnAdcChannelEnable(
HDLN handle,
uint8_t port,
uint8_t channel
);
```

The `DlnAdcChannelEnable()` function activates the specified channel from the corresponding ADC port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to be used.

channel
A number of the channel to be enabled.

3.3.7. DlnAdcChannelDisable()

```
DLN_RESULT DlnAdcChannelDisable(
HDLN handle,
uint8_t port,
uint8_t channel
);
```

The `DlnAdcChannelEnable()` function activates the specified channel from the corresponding ADC port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to be used.

channel
A number of the channel to be disabled.

3.3.8. DlnAdcChannelIsEnabled()

```
DLN_RESULT DlnAdcChannelIsEnabled(
   HDLN handle,
   uint8_t port,
   uint8_t channel,
   uint8_t* enabled
);
```

The `DlnAdcChannelIsEnabled()` retrieves information, whether the specified ADC channel is activated.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

channel
A number of the ADC channel to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with the information whether the specified ADC channel is activated. There are two possible values:

- `0` or `DLN_ADC_DISABLED` - the ADC channel is deactivated.
- `1` or `DLN_ADC_ENABLED` - the ADC channel is activated.

3.3.9. DlnAdcSetResolution()

```
DLN_RESULT DlnAdcSetResolution(
    HDLN handle,
    uint8_t port,
    uint8_t resolution
);
```

The `DlnAdcSetResolution()` function sets the ADC resolution value of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to be configured.

resolution
The new resolution in bits. For the list of compatible values see table below.

3.3.10. DlnAdcGetResolution()

```
DLN_RESULT DlnAdcGetResolution(
   HDLN handle,
   uint8_t port,
   uint8_t* resolution
);
```

The `DlnAdcGetResolution()` function retrieves the currently set ADC resolution of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

resolution
The current ADC resolution in bits.

3.3.11. DlnAdcGetValue()

```
DLN_RESULT DlnAdcGetValue(
                    HDLN handle,
                    uint8_t port,
                    uint8_t channel,
                    uint16_t* value
                        );
```

The `DlnAdcGetValue()` function retrieves current voltage on the specified ADC channel of your DLN-series adapter.

title

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

channel
A number of the ADC channel to retrieve the information from.

value
A pointer to an unsigned 8-bit integer. The integer will be filled with the voltage level on the specified ADC channel.

3.3.12. DlnAdcGetAllValues()

```
DLN_RESULT DlnAdcGetAllValues(
   HDLN handle,
   uint8_t port,
   uint16_t* channelMask,
   uint16_t* values
);
```

The `DlnAdcGetAllValues()` function retrieves current voltage values from all enabled ADC channels of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

channelMask
A pointer to an unsigned 16-bit integer. This integer will be filled bith a bit mask, each of the bits corresponding to an ADC channel of the port. This parameter contains currently enabled ADC channels of your DLN-series device.

3.3.13. DlnAdcChannelSetCfg()

```
DLN_RESULT DlnAdcChannelSetCfg(
   HDLN handle,
   uint8_t port,
   uint8_t channel,
   uint8_t eventType,
   uint16_t eventPeriod,
   uint16_t thresholdLow,
   uint16_t thresholdHigh
);
```

The `DlnAdcChannelSetCfg()` function changes the configuration of a single GPIO pin and set the conditions of `DLN_ADC_CONDITION_MET_EV` event generation.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to be configured.

channel
A number of the ADC channel to be configured.

eventType
Must contain the event generation condition for the ADC channel. The following values are available:

- `0` or `DLN_ADC_EVENT_NONE` - no events are generated for the current channel;
- `1` or `DLN_ADC_EVENT_BELOW` - events are generated when voltage level on the ADC channel crosses the lower threshhold;
- `2` or `DLN_ADC_EVENT_LEVEL_ABOVE` - events are generated when voltage level on the ADC channel crosses the higher threshold;
- `3` or `DLN_ADC_EVENT_OUTSIDE` - events are generated when voltage level on the ADC channel falls outside of the specified range between thresholds;
- `4` or `DLN_ADC_EVENT_INSIDE` - events are generated when voltage level on the ADC channel falls within the specified range between thresholds;
- `5` or `DLN_ADC_EVENT_ALWAYS` - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

eventPeriod
Must contain the repeat interval for `DLN_ADC_CONDITION_MET_EV` event generation on the pin. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the DLN-series adapter will send a single event when the level on the line changes to meet the specified conditions.

thresholdLow
The lower threshold value, specified in bits.

thresholdHigh
The higher threshold value specified in bits.

3.3.14. DlnAdcChannelGetCfg()

```
DLN_RESULT DlnAdcChannelGetCfg(
   HDLN handle,
   uint8_t port,
   uint8_t channel,
   uint8_t* eventType,
   uint16_t* eventPeriod,
   uint16_t* thresholdLow,
   uint16_t* thresholdHigh
);
```

The `DlnAdcChannelGetCfg()` function retrieves the current configuration settings of a single ADC channel.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

channel
A number of the ADC channel to retrieve the information from.

eventType
a pointer to an unsigned 8-bit integer. The integer will be filled the currently set event generation condition for the ADC channel after the function execution. The following values are available:

- `0` or `DLN_ADC_EVENT_NONE` - no events are generated for the current channel;
- `1` or `DLN_ADC_EVENT_BELOW` - events are generated when voltage level on the ADC channel crosses the lower threshhold;
- `2` or `DLN_ADC_EVENT_LEVEL_ABOVE` - events are generated when voltage level on the ADC channel crosses the higher threshold;
- `3` or `DLN_ADC_EVENT_OUTSIDE` - events are generated when voltage level on the ADC channel falls outside of the specified range between thresholds;
- `4` or `DLN_ADC_EVENT_INSIDE` - events are generated when voltage level on the ADC channel falls within the specified range between thresholds;
- `5` or `DLN_ADC_EVENT_ALWAYS` - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

eventPeriod
A pointer to an unsigned 16-bit integer. The integer will be filled with the repeat interval for `DLN_ADC_CONDITION_MET_EV` event generation on the pin after the function execution. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the DLN-series adapter will send a single event when the level on the line changes to meet the specified conditions.

thresholdLow
A pointer to an unsigned 16-bit integer. The integer will be filled with the lower voltage threshold value, specified in bits, after the function execution.

thresholdHigh
A pointer to an unsigned 16-bit integer. The integer will be filled with the higher voltage threshold value, specified in bits, after the function execution.

3.4. Commands and Responses

3.4.1. DLN_ADC_GET_PORT_COUNT

DLN_ADC_GET_PORT_COUNT Command

Go to response

The **DLN_ADC_GET_PORT_COUNT** command is used to retrieve the number of ADC ports available in your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_ADC_GET_PORT_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_GET_PORT_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_PORT_COUNT command it must be set to `0x0600`. You can use the `DLN_MSG_ID_ADC_GET_PORT_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_ADC_GET_PORT_COUNT Response

Go to Command

The adapter sends the **DLN_ADC_GET_PORT_COUNT** response after the command execution. The response contains the available number of ADC ports.

```
typedef struct
{
   DLN_MSG_HEADER header;
```

```
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_ADC_GET_PORT_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_GET_PORT_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_PORT_COUNT response it is set to `0x0600`. The `DLN_MSG_ID_ADC_GET_PORT_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of ADC ports has been successfully obtained.

count
Contains the available number of ADC ports.

3.4.2. DLN_ADC_GET_CHANNEL_COUNT

DLN_ADC_GET_CHANNEL_COUNT Command

Go to response

The **DLN_ADC_GET_CHANNEL_COUNT** command is used to retrieve the number of ADC channels, available in the specified ADC-port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_ADC_GET_CHANNEL_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_GET_CHANNEL_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_CHANNEL_COUNT command it must be set to `0x0601`. You can use the `DLN_MSG_ID_ADC_GET_CHANNEL_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the number of channels from.

DLN_ADC_GET_CHANNEL_COUNT Response

Go to Command

The adapter sends the **DLN_ADC_GET_CHANNEL_COUNT** response after the command execution. The response contains the available number of channels in the specified ADC port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_ADC_GET_CHANNEL_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_GET_CHANNEL_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_CHANNEL_COUNT response it is set to `0x0601`. The `DLN_MSG_ID_ADC_GET_CHANNEL_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of ADC channels has been successfully obtained.

count
Contains the available number of channels in the specified ADC port of the DLN-series adapter.

3.4.3. DLN_ADC_ENABLE

DLN_ADC_ENABLE Command

Go to response

The **DLN_ADC_ENABLE** command is used to activate the corresponding ADC port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_ADC_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_ENABLE command it must be set to `0x0602`. You can use the `DLN_MSG_ID_ADC_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to be activated.

DLN_ADC_ENABLE Response

Go to command

The adapter sends the **DLN_ADC_ENABLE** response after the command execution. The `result` field informs a user if the specified ADC port has been successfully activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t conflict;
}__PACKED_ATTR DLN_ADC_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_ENABLE response it is set to `0x0602`. The `DLN_MSG_ID_ADC_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified ADC port has been successfully activated.

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used for the ADC module. To fix this a user has to disconnect a pin from a module that it has been

assigned to and send the DLN_ADC_ENABLE command once again. In case there still are conflicted pins, only the number of the next one will be returned.

3.4.4. DLN_ADC_DISABLE

DLN_ADC_DISABLE Command

Go to response

The **DLN_ADC_DISABLE** command is used to deactivate the specified ADC port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_ADC_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_DISABLE command it must be set to `0x0603`. You can use the `DLN_MSG_ID_ADC_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to be deactivated.

DLN_ADC_DISABLE Response

Go to command

The adapter sends the **DLN_ADC_DISABLE** response after the function execution. The `result` field informs a user if the specified ADC port has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_ADC_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_DISABLE response it is set to `0x0603`. The `DLN_MSG_ID_ADC_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified ADC port has been successfully deactivated.

3.4.5. DLN_ADC_IS_ENABLED

DLN_ADC_IS_ENABLED Command

Go to response

The **DLN_ADC_IS_ENABLED** command is used to retrieve information, whether the specified ADC port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_ADC_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_IS_ENABLED command it must be set to `0x0604`. You can use the `DLN_MSG_ID_ADC_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

DLN_ADC_IS_ENABLED Response

Go to command

The adapter sends the **DLN_ADC_IS_ENABLED** response after the command execution. The response informs a user if the specified ADC port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_ADC_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_IS_ENABLED response it is set to `0x0604`. The `DLN_MSG_ID_ADC_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the ADC port has been successfully retrieved.

enabled
Informs whether the specified ADC port is activated. There are two possible values:

- `0` or `DLN_ADC_DISABLED` - the ADC port is deactivated.
- `1` or `DLN_ADC_ENABLED` - the ADC port is activated.

3.4.6. DLN_ADC_CHANNEL_ENABLE

DLN_ADC_CHANNEL_ENABLE Command

Go to response

The **DLN_ADC_CHANNEL_ENABLE** command is used to activate the specified channel from the corresponding ADC port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
}__PACKED_ATTR DLN_ADC_CHANNEL_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_CHANNEL_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_ENABLE command it must be set to `0x0605`. You can use the `DLN_MSG_ID_ADC_CHANNEL_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to be used.

channel
A number of the channel to be enabled.

DLN_ADC_CHANNEL_ENABLE Response

Go to command

The adapter sends the **DLN_ADC_CHANNEL_ENABLE** response after the command execution. The `result` field informs a user if the specified channel has been successfully activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t channel;
}__PACKED_ATTR DLN_ADC_CHANNEL_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_CHANNEL_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_ENABLE response it is set to `0x0605`. The `DLN_MSG_ID_ADC_CHANNEL_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified ADC channel has been successfully activated.

3.4.7. DLN_ADC_CHANNEL_DISABLE

DLN_ADC_CHANNEL_DISABLE Command

Go to response

The **DLN_ADC_CHANNEL_DISABLE** command is used to deactivate the specified ADC channel of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
}__PACKED_ATTR DLN_ADC_CHANNEL_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_CHANNEL_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_DISABLE command it must be set to `0x0606`. You can use the `DLN_MSG_ID_ADC_CHANNEL_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to be used.

channel
A number of the ADC channel to be deactivated.

DLN_ADC_CHANNEL_DISABLE Response

Go to command

The adapter sends the **DLN_ADC_CHANNEL_DISABLE** response after the function execution. The `result` field informs a user if the specified ADC channel has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_ADC_CHANNEL_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_CHANNEL_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_DISABLE response it is set to `0x0606`. The `DLN_MSG_ID_ADC_CHANNEL_DISABLE` constant can be used.

- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified ADC channel has been successfully deactivated.

3.4.8. DLN_ADC_CHANNEL_IS_ENABLED

DLN_ADC_CHANNEL_IS_ENABLED Command

Go to response

The **DLN_ADC_CHANNEL_IS_ENABLED** command is used to retrieve information, whether the specified ADC channel is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
}__PACKED_ATTR DLN_ADC_CHANNEL_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_CHANNEL_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_IS_ENABLED command it must be set to `0x0607`. You can use the `DLN_MSG_ID_ADC_CHANNEL_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

channel
A number of the ADC channel to retrieve the information from.

DLN_ADC_CHANNEL_IS_ENABLED Response

Go to command

The adapter sends the **DLN_ADC_CHANNEL_IS_ENABLED** response after the command execution. The response informs a user if the specified ADC channel is activated.

```
typedef struct
```

```
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_ADC_CHANNEL_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_CHANNEL_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_IS_ENABLED response it is set to `0x0607`. The `DLN_MSG_ID_ADC_CHANNEL_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the ADC channel has been successfully retrieved.

enabled
Informs whether the specified ADC channel is activated. There are two possible values:

- `0` or `DLN_ADC_CHANNEL_DISABLED` - the specified ADC channel is deactivated.
- `1` or `DLN_ADC_CHANNEL_ENABLED` - the specified ADC channel is activated.

3.4.9. DLN_ADC_SET_RESOLUTION

DLN_ADC_SET_RESOLUTION Command

Go to response

The **DLN_ADC_SET_RESOLUTION** command is used to set the ADC resolution value of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t resolution;
} __PACKED_ATTR DLN_ADC_SET_RESOLUTION_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_SET_RESOLUTION_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_SET_RESOLUTION_ENABLED command it must be set to `0x0608`. You can use the `DLN_MSG_ID_ADC_SET_RESOLUTION` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to be configured.

resolution
The new resolution in bits. For the list of compatible values see table below.

DLN_ADC_SET_RESOLUTION Response

Go to command

The adapter sends the **DLN_ADC_SET_RESOLUTION** response after the command execution. The `result` field informs a user if analog-to-digital conversion resolution has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_ADC_SET_RESOLUTION_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_SET_RESOLUTION_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_SET_RESOLUTION response it is set to `0x0608`. The `DLN_MSG_ID_ADC_SET_RESOLUTION` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The ADC resolution has been successfully set.

3.4.10. DLN_ADC_GET_RESOLUTION

DLN_ADC_GET_RESOLUTION_CMD

Go to response

The **DLN_ADC_GET_RESOLUTION** command is used to retrieve the currently set ADC resolution of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_ADC_GET_RESOLUTION_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_GET_RESOLUTION_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_RESOLUTION_ENABLED command it must be set to `0x0609`. You can use the `DLN_MSG_ID_ADC_GET_RESOLUTION` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to retrieve the information from.

DLN_ADC_GET_RESOLUTION Response

Go to command

The adapter sends the **DLN_ADC_GET_RESOLUTION** response after the command execution. The response contains the currently set ADC resolution of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t resolution;
} __PACKED_ATTR DLN_ADC_GET_RESOLUTION_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_GET_RESOLUTION_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_RESOLUTION response it is set to `0x0609`. The `DLN_MSG_ID_ADC_GET_RESOLUTION` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The ADC resolution has been successfully retrieved.

resolution
The current ADC resolution in bits.

3.4.11. DLN_ADC_GET_VALUE

DLN_ADC_GET_VALUE Command

Go to response

The **DLN_ADC_GET_VALUE** command is used to retrieve current voltage on the specified ADC channel of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
} __PACKED_ATTR DLN_ADC_GET_VALUE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_GET_VALUE_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_VALUE command it must be set to `0x060A`. You can use the `DLN_MSG_ID_ADC_GET_VALUE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
An ADC port to retrieve the information from.

channel
An ADC channel to retrieve the information from.

DLN_ADC_GET_VALUE Response

Go to command

The adapter sends the **DLN_ADC_GET_VALUE** response after the command execution. The response contains current voltage on the specified ADC channel of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
```

```
   DLN_RESULT result;
   uint16_t value;
} __PACKED_ATTR DLN_ADC_GET_VALUE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_GET_VALUE_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_VALUE response it is set to `0x060A`. The `DLN_MSG_ID_ADC_GET_VALUE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The voltage on the specified ADC channel has been successfully retrieved.

value
The voltage on the specified ADC channel.

3.4.12. DLN_ADC_GET_ALL_VALUES

DLN_ADC_GET_ALL_VALUES Command

Go to response

The **DLN_ADC_GET_ALL_VALUES** command retrieves current voltage values from all enabled ADC channels of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_ADC_GET_ALL_VALUES_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_GET_ALL_VALUES_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_ALL_VALUES command it must be set to `0x060B`. You can use the `DLN_MSG_ID_ADC_GET_ALL_VALUES` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
An ADC port to retrieve the information from.

DLN_ADC_GET_ALL_VALUES Response

Go to command

The adapter sends the **DLN_ADC_GET_ALL_VALUES** response after the function execution. The response contains current voltage values from all enabled ADC channels of your DLN-series adapter

```
typedef struct
{
DLN_MSG_HEADER header;
DLN_RESULT result;
uint16_t channelMask;
uint16_t values[DLN_ADC_CHANNEL_COUNT_MAX];
} __PACKED_ATTR DLN_ADC_GET_ALL_VALUES_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_GET_ALL_VALUES_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_GET_ALL_VALUES response it is set to `0x060B`. The `DLN_MSG_ID_ADC_GET_ALL_VALUES` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Current voltage levels on all enabled ADC channels of your DLN-series device have been successfully retrieved.

channelMask
A 16-bit mask. Each of the bits corresponds to an ADC channel of the port. This parameter contains currently enabled ADC channels of your DLN-series device.

values
An array. Each of its elements contains current voltage value on one of the enabled ADC channels of your DLN-series adapter.

3.4.13. DLN_ADC_CHANNEL_SET_CFG

DLN_ADC_CHANNEL_SET_CFG Command

Go to response

The **DLN_ADC_CHANNEL_SET_CFG** command is used to change the configuration of a single GPIO pin and set the conditions of `DLN_ADC_CONDITION_MET_EV` event generation.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
   uint8_t eventType;
   uint16_t eventPeriod;
   uint16_t thresholdLow;
   uint16_t thresholdHigh;
} __PACKED_ATTR DLN_ADC_CHANNEL_SET_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_CHANNEL_SET_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_SET_CFG command it must be set to `0x060C`. You can use the `DLN_MSG_ID_ADC_CHANNEL_SET_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the ADC port to be configured.

channel
A number of the ADC channel to be configured.

eventType
Must contain the event generation condition for the ADC channel. The following values are available:

- `0` or `DLN_ADC_EVENT_NONE` - no events are generated for the current channel;
- `1` or `DLN_ADC_EVENT_BELOW` - events are generated when voltage level on the ADC channel crosses the lower threshhold;
- `2` or `DLN_ADC_EVENT_LEVEL_ABOVE` - events are generated when voltage level on the ADC channel crosses the higher threshold;
- `3` or `DLN_ADC_EVENT_OUTSIDE` - events are generated when voltage level on the ADC channel falls outside of the specified range between thresholds;
- `4` or `DLN_ADC_EVENT_INSIDE` - events are generated when voltage level on the ADC channel falls within the specified range between thresholds;
- `5` or `DLN_ADC_EVENT_ALWAYS` - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

eventPeriod
Must contain the repeat interval for `DLN_ADC_CONDITION_MET_EV` event generation on the pin. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the DLN-series adapter will send a single event when the level on the line changes to meet the specified conditions.

thresholdLow
The lower threshold value, specified in bits.

thresholdHigh
The higher threshold value specified in bits.

DLN_ADC_CHANNEL_SET_CFG Response

Go to command

The adapter sends the **DLN_ADC_CHANNEL_SET_CFG** response after the command execution. The `result` field informs a user is the ADC channel configuration has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_ADC_CHANNEL_SET_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_CHANNEL_SET_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_SET_CFG response it is set to `0x060C`. The `DLN_MSG_ID_ADC_CHANNEL_SET_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The new configuration of the specified ADC channel has been successfully set.

3.4.14. DLN_ADC_CHANNEL_GET_CFG

DLN_ADC_CHANNEL_GET_CFG Command

Go to response

The **DLN_ADC_CHANNEL_GET_CFG** command is used to retrieve the current configuration settings of a single ADC channel.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
} __PACKED_ATTR DLN_ADC_CHANNEL_GET_CFG_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_ADC_CHANNEL_GET_CFG_CMD structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_GET_CFG command it must be set to `0x060D`. You can use the `DLN_MSG_ID_ADC_CHANNEL_GET_CFG` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
An ADC port to retrieve the information from.

channel
An ADC channel to retrieve the information from.

DLN_ADC_CHANNEL_GET_CFG Response

Go to command

The adapter sends the **DLN_ADC_CHANNEL_GET_CFG** response after the command execution. The response contains current configuration of the specified ADC channel.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t eventType;
   uint16_t eventPeriod;
   uint16_t thresholdLow;
   uint16_t thresholdHigh;
} __PACKED_ATTR DLN_ADC_CHANNEL_GET_CFG_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_ADC_CHANNEL_GET_CFG_RSP structure.
- **msgId** - Defines the message. For the DLN_ADC_CHANNEL_GET_CFG response it is set to `0x060D`. The `DLN_MSG_ID_ADC_CHANNEL_GET_CFG` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- DLN_RES_SUCCESS - Current configuration of the specified ADC channel has been successfully retrieved.

eventType

Contains the currently set event generation condition for the ADC channel. The following values are available:

- 0 or DLN_ADC_EVENT_NONE - no events are generated for the current channel;
- 1 or DLN_ADC_EVENT_BELOW - events are generated when voltage level on the ADC channel crosses the lower threshhold;
- 2 or DLN_ADC_EVENT_LEVEL_ABOVE - events are generated when voltage level on the ADC channel crosses the higher threshold;
- 3 or DLN_ADC_EVENT_OUTSIDE - events are generated when voltage level on the ADC channel falls outside of the specified range between thresholds;
- 4 or DLN_ADC_EVENT_INSIDE - events are generated when voltage level on the ADC channel falls within the specified range between thresholds;
- 5 or DLN_ADC_EVENT_ALWAYS - events are sent periodically with predefined repeat interval. The non-zero interval must be specified for this event type.

eventPeriod

Contains the repeat interval for DLN_ADC_CONDITION_MET_EV event generation on the pin. The repeat interval is set in ms (1 to 65,535ms). If the repeat interval is set to 0, the DLN-series adapter will send a single event when the level on the line changes to meet the specified conditions.

thresholdLow

The lower voltage threshold value, specified in bits.

thresholdHigh

The higher voltage threshold value specified in bits.

Chapter 4. PWM Module

4.1. PWM Module

4.2. Functions

4.2.1. DlnPwmGetPortCount()

```
DLN_RESULT DlnPwmGetPortCount(
HDLN handle,
uint8_t* count
);
```

The `DlnPwmGetPortCount()` function retrieves the number of PWM ports available in your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. The integer will be filled with the number of available PWM ports after the function execution.

4.2.2. DlnPwmGetChannelCount()

```
DLN_RESULT DlnPwmGetChannelCount(
HDLN handle,
uint8_t port,
uint8_t* count
);
```

The `DlnPwmGetChannelCount()` function retrieves the number of PWM channels, available in the specified PWM-port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A PWM port to retrieve the number of channels from.

count
A pointer to an unsigned 8-bit integer. The integer will be filled with the available number of channels in the specified PWM port of the DLN-series adapter.

4.2.3. DlnPwmEnable()

```
DLN_RESULT DlnPwmEnable(
```

```
                    HDLN handle,
                    uint8_t port,
                    uint16_t* conflict
                    );
```

The `DlnPwmEnable()` function activates the corresponding PWM port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to be activated.

conflict
A pointer to an unsigned 16-bit integer. The integer will be filled with the number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used for the PWM module. To fix this a user has to disconnect a pin from a module that it has been assigned to and call the DlnPwmEnable() function once again. In case there still are conflicted pins, only the number of the next one will be returned.

4.2.4. DlnPwmDisable()

```
                    DLN_RESULT DlnPwmDisable(
                    HDLN handle,
                    uint8_t port
                    );
```

The `DlnPwmDisable()` function deactivates the specified PWM port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to be deactivated.

4.2.5. DlnPwmIsEnabled()

```
                    DLN_RESULT DlnPwmIsEnabled(
                    HDLN handle,
                    uint8_t port,
                    uint8_t* enabled
                    );
```

The `DlnPwmIsEnabled()` function retrieves information, whether the specified PWM port is activated.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with the information whether the specified PWM port is activated. There are two possible values:

- `0` or `DLN_PWM_DISABLED` - the PWM port is deactivated.
- `1` or `DLN_PWM_ENABLED` - the PWM port is activated.

4.2.6. DlnPwmChannelEnable()

```
DLN_RESULT DlnPwmChannelEnable(
HDLN handle,
uint8_t port,
uint8_t channel
);
```

The `DlnPwmChannelEnable()` function activates the specified channel from the corresponding PWM port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to be used.

channel
A number of the channel to be enabled.

4.2.7. DlnPwmChannelDisable()

```
DLN_RESULT DlnPwmChannelDisable(
HDLN handle,
uint8_t port,
uint8_t channel
);
```

The `DlnPwmChannelEnable()` function activates the specified channel from the corresponding PWM port of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to be used.

channel
A number of the channel to be disabled.

4.2.8. DlnPwmChannelIsEnabled()

```
DLN_RESULT DlnPwmChannelIsEnabled(
HDLN handle,
uint8_t port,
uint8_t channel,
uint8_t* enabled
);
```

The `DlnPwmChannelIsEnabled()` retrieves information, whether the specified PWM channel is activated.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve the information from.

channel
A number of the PWM channel to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with the information whether the specified PWM channel is activated. There are two possible values:

- `0` or `DLN_PWM_DISABLED` - the PWM channel is deactivated.
- `1` or `DLN_PWM_ENABLED` - the PWM channel is activated.

4.2.9. DlnPwmSetFrequency()

```
DLN_RESULT DlnPwmSetFrequency(
   HDLN handle,
   uint8_t port,
   uint8_t channel,
   uint32_t frequency,
   uint32_t* actualFrequency
);
```

The `DlnPwmSetFrequency()` function sets the PWM frequency.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to be configured.

channel
A number of the PWM channel to be configured.

frequency
The frequency value, specified in Hz. A user may specify any value within the range, supported by the DLN-series adapter. This range can be retrieved using the respective function. In case a user enters

an incompatible value, it will be approximated as the closest lower frequency value, supported by the adapter.

actualFrequency
A pointer to an unsigned 32-bit integer. This integer will be filled with the actually set frequency approximated as the closest to user-defined lower value.

4.2.10. DlnPwmGetFrequency()

```
DLN_RESULT DlnPwmGetFrequency(
HDLN handle,
uint8_t port,
uint8_t channel,
uint32_t* frequency,
);
```

The `DlnPwmSetFrequency()` function retrieves current setting for PWM frequency.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve current frequency setting from.

channel
A number of the PWM channel to retrieve current frequency setting from.

frequency
A pointer to an unsigned 32-bit integer. This integer will be filled with the current frequency setting for the DLN-series adapter.

4.2.11. DlnPwmSetDutyCycle()

```
DLN_RESULT DlnPwmSetDutyCycle(
   HDLN handle,
   uint8_t port,
   uint8_t channel,
   double dutyCycle,
   double* actualDutyCycle
);
```

The `DlnPwmSetDutyCycle()` function sets a PWM duty cycle, which is the ratio of the high time to the PWM period.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to set the duty cycle for.

channel
A number of the PWM channel to set the duty cycle for.

dutyCycle
A double precision floating point number. Must contain a duty cycle to be set.

actualDutyCycle
A pointer to a double precision floating point number. This number will be filled with actually set duty cycle after the function execution.

4.2.12. DlnPwmGetDutyCycle()

```
DLN_RESULT DlnPwmGetDutyCycle(
   HDLN handle,
   uint8_t port,
   uint8_t channel,
   double* dutyCycle
);
```

The `DlnPwmGetDutyCycle()` function retrieves the currently set PWM duty cycle from your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve the current duty cycle from.

channel
A number of the PWM channel to retrieve the current duty cycle from.

dutyCycle
A pointer to the double precsion floating point number. This number will be filled with current duty cycle after the function execution.

4.3. Commands and Responses

4.3.1. DLN_PWM_GET_PORT_COUNT

DLN_PWM_GET_PORT_COUNT Command

Go to response

The **DLN_PWM_GET_PORT_COUNT** command is used to retrieve the number of PWM ports available in your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_PWM_GET_PORT_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_GET_PORT_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_PORT_COUNT command it must be set to `0x0700`. You can use the `DLN_MSG_ID_PWM_GET_PORT_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_PWM_GET_PORT_COUNT Response

Go to Command

The adapter sends the **DLN_PWM_GET_PORT_COUNT** response after the command execution. The response contains the available number of PWM ports.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_PWM_GET_PORT_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_GET_PORT_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_PORT_COUNT response it is set to `0x0700`. The `DLN_MSG_ID_PWM_GET_PORT_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of PWM ports has been successfully obtained.

count
Contains the available number of PWM ports.

4.3.2. DLN_PWM_GET_CHANNEL_COUNT

DLN_PWM_GET_CHANNEL_COUNT Command

Go to response

The **DLN_PWM_GET_CHANNEL_COUNT** command is used to retrieve the number of PWM channels, available in the specified PWM-port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_PWM_GET_CHANNEL_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_GET_CHANNEL_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_CHANNEL_COUNT command it must be set to `0x0701`. You can use the `DLN_MSG_ID_PWM_GET_CHANNEL_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve the number of channels from.

DLN_PWM_GET_CHANNEL_COUNT Response

Go to Command

The adapter sends the **DLN_PWM_GET_CHANNEL_COUNT** response after the command execution. The response contains the available number of channels in the specified PWM port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_PWM_GET_CHANNEL_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_GET_CHANNEL_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_CHANNEL_COUNT response it is set to `0x0701`. The `DLN_MSG_ID_PWM_GET_CHANNEL_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.

- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of PWM channels has been successfully obtained.

count
Contains the available number of channels in the specified PWM port of the DLN-series adapter.

4.3.3. DLN_PWM_ENABLE

DLN_PWM_ENABLE Command

Go to response

The **DLN_PWM_ENABLE** command is used to activate the corresponding PWM port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_PWM_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_ENABLE command it must be set to `0x0702`. You can use the `DLN_MSG_ID_PWM_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to be activated.

DLN_PWM_ENABLE Response

Go to command

The adapter sends the **DLN_PWM_ENABLE** response after the command execution. The `result` field informs a user if the specified PWM port has been successfully activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
```

```
   uint16_t conflict;
}__PACKED_ATTR DLN_PWM_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_ENABLE response it is set to `0x0702`. The `DLN_MSG_ID_PWM_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified PWM port has been successfully activated.

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used for the PWM module. To fix this a user has to disconnect a pin from a module that it has been assigned to and send the DLN_PWM_ENABLE command once again. In case there still are conflicted pins, only the number of the next one will be returned.

4.3.4. DLN_PWM_DISABLE

DLN_ADC_DISABLE Command

Go to response

The **DLN_PWM_DISABLE** command is used to deactivate the specified PWM port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_PWM_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_DISABLE_CMD structure.

- **msgId** - Defines the message. For the DLN_ADC_DISABLE command it must be set to `0x0703`. You can use the `DLN_MSG_ID_PWM_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to be deactivated.

DLN_PWM_DISABLE Response

Go to command

The adapter sends the **DLN_PWM_DISABLE** response after the function execution. The `result` field informs a user if the specified PWM port has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_PWM_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_DISABLE response it is set to `0x0703`. The `DLN_MSG_ID_PWM_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified PWM port has been successfully deactivated.

4.3.5. DLN_PWM_IS_ENABLED

DLN_PWM_IS_ENABLED Command

Go to response

The **DLN_PWM_IS_ENABLED** command is used to retrieve information, whether the specified PWM port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
```

```
   uint8_t port;
}__PACKED_ATTR DLN_PWM_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_IS_ENABLED command it must be set to `0x0704`. You can use the `DLN_MSG_ID_PWM_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve the information from.

DLN_PWM_IS_ENABLED Response

Go to command

The adapter sends the **DLN_PWM_IS_ENABLED** response after the command execution. The response informs a user if the specified PWM port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_PWM_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_IS_ENABLED response it is set to `0x0704`. The `DLN_MSG_ID_PWM_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the PWM port has been successfully retrieved.

enabled
Informs whether the specified PWM port is activated. There are two possible values:

- `0` or `DLN_PWM_DISABLED` - the PWM port is deactivated.
- `1` or `DLN_PWM_ENABLED` - the PWM port is activated.

4.3.6. DLN_PWM_CHANNEL_ENABLE

DLN_PWM_CHANNEL_ENABLE Command

Go to response

The **DLN_PWM_CHANNEL_ENABLE** command is used to activate the specified channel from the corresponding PWM port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
}__PACKED_ATTR DLN_PWM_CHANNEL_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_CHANNEL_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_CHANNEL_ENABLE command it must be set to `0x0705`. You can use the `DLN_MSG_ID_PWM_CHANNEL_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to be used.

channel
A number of the channel to be enabled.

DLN_PWM_CHANNEL_ENABLE Response

Go to command

The adapter sends the **DLN_PWM_CHANNEL_ENABLE** response after the command execution. The `result` field informs a user if the specified channel has been successfully activated.

```
typedef struct
```

```
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t channel;
}__PACKED_ATTR DLN_PWM_CHANNEL_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_CHANNEL_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_CHANNEL_ENABLE response it is set to `0x0705`. The `DLN_MSG_ID_PWM_CHANNEL_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified PWM channel has been successfully activated.

4.3.7. DLN_PWM_CHANNEL_DISABLE

DLN_PWM_CHANNEL_DISABLE Command

Go to response

The **DLN_PWM_CHANNEL_DISABLE** command is used to deactivate the specified PWM channel of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
}__PACKED_ATTR DLN_PWM_CHANNEL_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_CHANNEL_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_CHANNEL_DISABLE command it must be set to `0x0706`. You can use the `DLN_MSG_ID_PWM_CHANNEL_DISABLE` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to be used.

channel
A number of the PWM channel to be deactivated.

DLN_PWM_CHANNEL_DISABLE Response

Go to command

The adapter sends the **DLN_PWM_CHANNEL_DISABLE** response after the function execution. The `result` field informs a user if the specified PWM channel has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_PWM_CHANNEL_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_CHANNEL_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_CHANNEL_DISABLE response it is set to `0x0706`. The `DLN_MSG_ID_PWM_CHANNEL_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified PWM channel has been successfully deactivated.

4.3.8. DLN_PWM_CHANNEL_IS_ENABLED

DLN_PWM_CHANNEL_IS_ENABLED Command

Go to response

The **DLN_PWM_CHANNEL_IS_ENABLED** command is used to retrieve information, whether the specified PWM channel is activated.

```
typedef struct
{
```

```
    DLN_MSG_HEADER header;
    uint8_t port;
    uint8_t channel;
}__PACKED_ATTR DLN_PWM_CHANNEL_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_CHANNEL_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_CHANNEL_IS_ENABLED command it must be set to `0x0707`. You can use the `DLN_MSG_ID_PWM_CHANNEL_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve the information from.

channel
A number of the PWM channel to retrieve the information from.

DLN_PWM_CHANNEL_IS_ENABLED Response

Go to command

The adapter sends the **DLN_PWM_CHANNEL_IS_ENABLED** response after the command execution. The response informs a user if the specified PWM channel is activated.

```
typedef struct
{
    DLN_MSG_HEADER header;
    DLN_RESULT result;
    uint8_t enabled;
}__PACKED_ATTR DLN_PWM_CHANNEL_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_CHANNEL_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_CHANNEL_IS_ENABLED response it is set to `0x0707`. The `DLN_MSG_ID_PWM_CHANNEL_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.

- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the PWM channel has been successfully retrieved.

enabled
Informs whether the specified PWM channel is activated. There are two possible values:

- `0` or `DLN_PWM_CHANNEL_DISABLED` - the specified PWM channel is deactivated.
- `1` or `DLN_PWM_CHANNEL_ENABLED` - the specified PWM channel is activated.

4.3.9. DLN_PWM_SET_FREQUENCY

DLN_PWM_SET_FREQUENCY Command

Go to response

The **DLN_PWM_SET_FREQUENCY** command is used to set the PWM frequency.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
   uint32_t frequency;
} __PACKED_ATTR DLN_PWM_SET_FREQUENCY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_SET_FREQUENCY_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_SET_FREQUENCY command it must be set to `0x0708`. You can use the `DLN_MSG_ID_PWM_SET_FREQUENCY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to be configured.

channel
A number of the PWM channel to be configured.

frequency
The frequency value, specified in Hz. A user may specify any value within the range, supported by the DLN-series adapter. This range can be retrieved using the respective function. In case a user enters an incompatible value, it will be approximated as the closest lower frequency value, supported by the adapter.

DLN_PWM_SET_FREQUENCY Response

Go to command

The adapter sends the **DLN_PWM_SET FREQUENCY** response after the command execution. The response informs a user if the PWM clock frequency has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t frequency;
} __PACKED_ATTR DLN_I2C_MASTER_SET_FREQUENCY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_SET_FREQUENCY_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_SET_FREQUENCY response it is set to `0x0708`. The `DLN_MSG_ID_PWM_SET_FREQUENCY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The PWM clock frequency has been successfully set.

frequency
Contains the actual frequency applied by this command. If the frequency specified in `frequency` parameter is supported, the actual frequency will equal to it. Otherwise the closest lower value will be applied. If `NULL` is specified in this parameter, the actual frequency value will not be returned. You can still use the DLN_PWM_GET_FREQUENCY command to check the actual frequency.

4.3.10. DLN_PWM_GET_FREQUENCY

DLN_PWM_GET_FREQUENCY Command

Go to response

The **DLN_PWM_GET_FREQUENCY** command is used to retrieve current PWM clock frequency.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
} __PACKED_ATTR DLN_PWM_GET_FREQUENCY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_GET_FREQUENCY_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_FREQUENCY command it must be set to `0x0709`. You can use the `DLN_MSG_ID_PWM_GET_FREQUENCY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve current frequency from.

port
A number of the PWM channel to retrieve current frequency from.

DLN_PWM_GET_FREQUENCY Response

Go to command

The adapter sends the **DLN_PWM_GET_FREQUENCY** response after the command execution. The response contains current PWM clock frequency setting.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t frequency;
} __PACKED_ATTR DLN_PWM_GET_FREQUENCY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_GET_FREQUENCY_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_FREQUENCY response it is set to `0x0709`. The `DLN_MSG_ID_PWM_GET_FREQUENCY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Current PWM clock frequency has been successfully retrieved.

frequency
Contains current PWM clock frequency in Hz.

4.3.11. DLN_PWM_SET_DUTY_CYCLE

DLN_PWM_SET_DUTY_CYCLE Command

Go to response

The **DLN_PWM_SET_DUTY_CYCLE** command is used to set a PWM duty cycle, which is the ratio of the high time to the PWM period.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
   uint16_t bsDutyCycle;
} __PACKED_ATTR DLN_PWM_SET_DUTY_CYCLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_SET_DUTY_CYCLE_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_SET_DUTY_CYCLE command it must be set to `0x070A`. You can use the `DLN_MSG_ID_PWM_SET_DUTY_CYCLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to set the duty cycle for.

channel
A number of the PWM channel to set the duty cycle for.

bsDutyCycle
The duty cycle to be set. This is a basis point parameter.

DLN_PWM_SET_DUTY_CYCLE Response

Go to command

The adapter sends the **DLN_PWM_SET_DUTY_CYCLE** response after the command execution. The `result` field informs a user if the new duty cycle has been successfully set. Also, the response contains actual duty cycle value, in case the one specified by a user is not supported by the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
```

```
   DLN_RESULT result;
   uint16_t bsDutyCycle;
} __PACKED_ATTR DLN_PWM_SET_DUTY_CYCLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_SET_DUTY_CYCLE_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_SET_DUTY_CYCLE response it is set to `0x070A`. The `DLN_MSG_ID_PWM_SET_DUTY_CYCLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the new PWM duty cycle has been successfully set.

bsDutyCycle
Actually set duty cycle. This is a basis point parameter.

4.3.12. DLN_PWM_GET_DUTY_CYCLE

DLN_PWM_GET_DUTY_CYCLE Command

Go to response

The **DLN_PWM_GET_DUTY_CYCLE** command is used to retrieve the currently set PWM duty cycle from the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t channel;
} __PACKED_ATTR DLN_PWM_SET_DUTY_CYCLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_PWM_GET_DUTY_CYCLE_CMD structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_DUTY_CYCLE command it must be set to `0x070B`. You can use the `DLN_MSG_ID_PWM_GET_DUTY_CYCLE` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the PWM port to retrieve the current duty cycle from.

channel
A number of the PWM channel to retrieve the current duty cycle from.

DLN_PWM_GET_DUTY_CYCLE Response

Go to command

The adapter sends the **DLN_PWM_GET_DUTY_CYCLE** response after the command execution. The response contains the currently set PWM duty cycle of the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t bsDutyCycle;
} __PACKED_ATTR DLN_PWM_GET_DUTY_CYCLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_PWM_GET_DUTY_CYCLE_RSP structure.
- **msgId** - Defines the message. For the DLN_PWM_GET_DUTY_CYCLE response it is set to `0x070B`. The `DLN_MSG_ID_PWM_GET_DUTY_CYCLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the PWM duty cycle has been successfully retrieved.

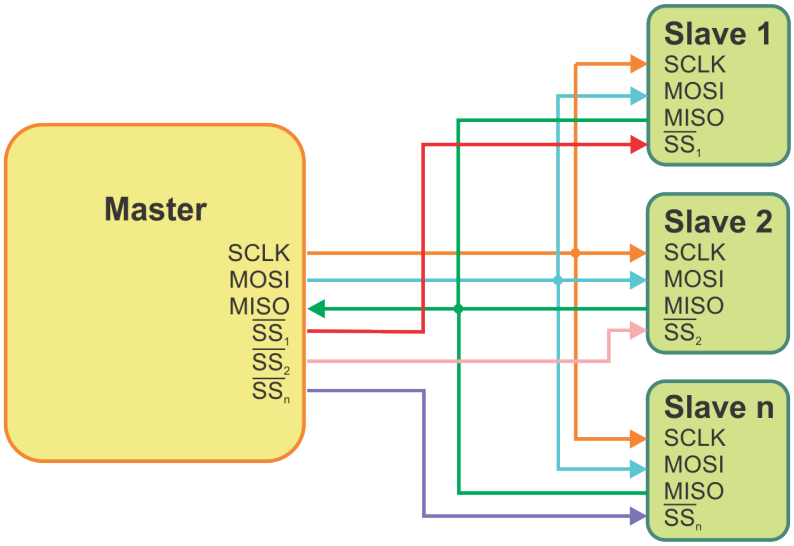
bsDutyCycle
A PWM duty cycle currently set for the DLN-series adapter.

Chapter 5. SPI Interface

5.1. Overview

The **Serial Peripheral Interface** (**SPI**) is a synchronous serial data link, developed by Motorola. SPI operates either in **full-duplex** or in **half-duplex** mode and can transfer data over short distances at high speeds.

During data transfer devices can work either in master or in slave mode. The source of synchronization is the system clock, which is generated by the master. The SPI interface allows to connect one or more slave devices to a single master device via a single bus.



The SPI interface has four signal lines: MOSI, MISO, SCLK, and SS.

- **MOSI** (Master Output, Slave Input) –the line is used to transfer data from master to slave.
- **MISO** (Master Input, Slave Output) – the line is used to transfer data from slave to master.
- **SS** (Slave Select) – the master uses this line to select the slave. SS is an **active low** line.
- **SCLK** (Serial Clock) – clock pulse output from the master and clock pulse input for the slave. The SCLK line is used to synchronize data transfer between master and slave devices via MOSI and MISO lines.

The data is transferred bit by bit, starting with the high bit. The slave must be in high-impedance state if the slave is not selected through the SS line.

SPI Key Features

- Read, write and full-duplex (simultaneous read/write) data transactions;
- SPI master and SPI slave configuration;
- Configurable SPI bus clock polarity, phase and frequency;
- Configurable number of bits to transfer;
- Communication with serial external devices (ADC and DAC, RTC, temperature and pressure sensors, LCD controller, etc.).

SPI Benefits

- SPI is considered the fastest synchronous serial data transfer interface;
- SPI is a very simple communication protocol;
- Supports full-duplex communication;

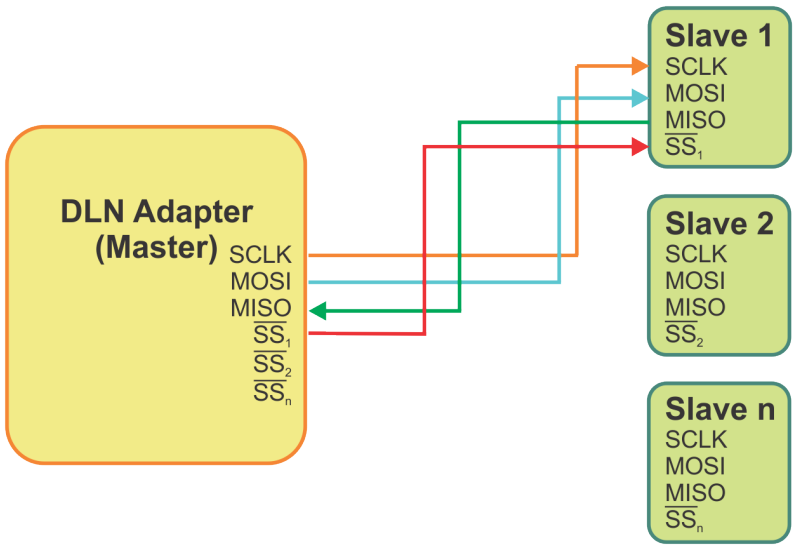
SPI Drawbacks

- Requires more traces on the board (X + 3, where X is the number of slave devices);
- No hardware flow control;
- No slave acknowledgment;
- May prone to noise spikes causing faulty communication.

5.2. Modes of Operation

Full Duplex Mode

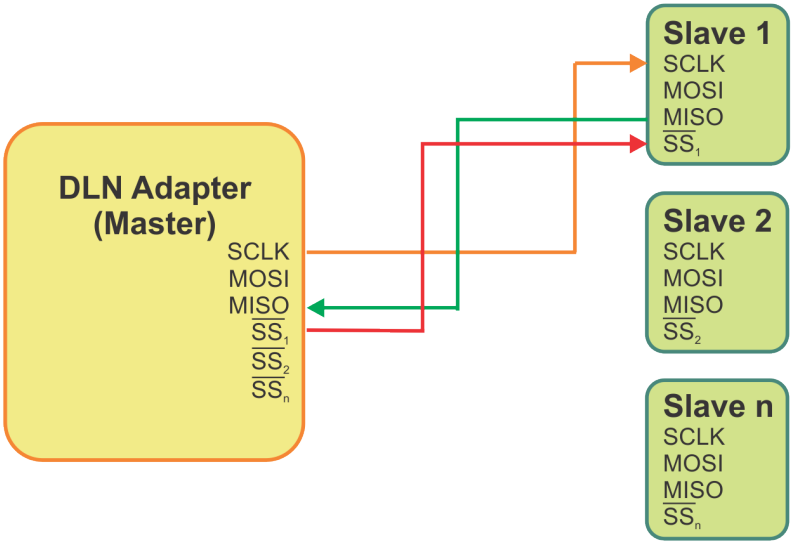
In **full duplex** mode the master simultaneously transmits data to a slave and receives data from a slave. This way only a single slave device can be engaged at one time. An SS line is used to select a slave device.



SPI in full duplex mode

Half Duplex Mode (Single Read)

In this mode a master only receives data from a slave. This way only a single slave device can be engaged at one time. An SS line is used to select the slave device.



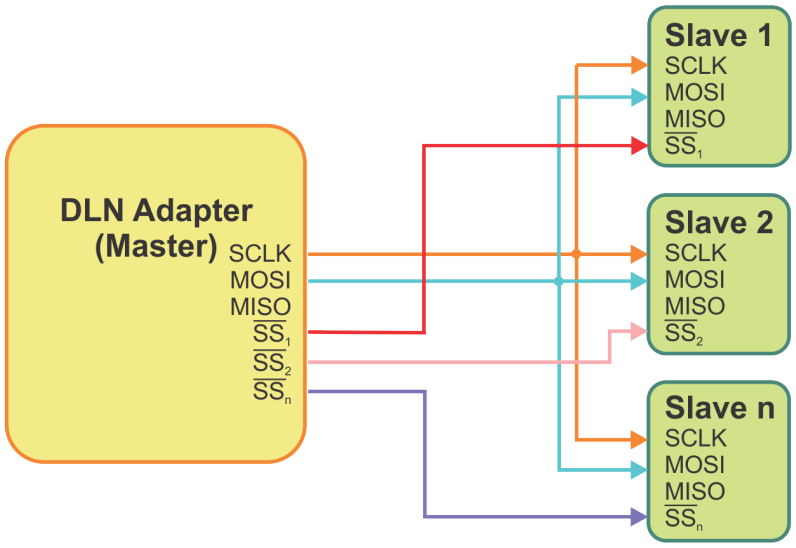
SPI in half duplex single read mode

Half Duplex Mode (Single Write)

In this mode a master device only transmits data to a slave, and doesn't receive any data from the slave. This way simultaneous operation with several slave devices is possible.

Warning

When operating in Half Duplex (Single Write) mode users are strongly encouraged NOT to interconnect the MISO lines and not to connect them to the master. This precaution will prevent possible damage to equipment if several slaves accidentally start outputting data.



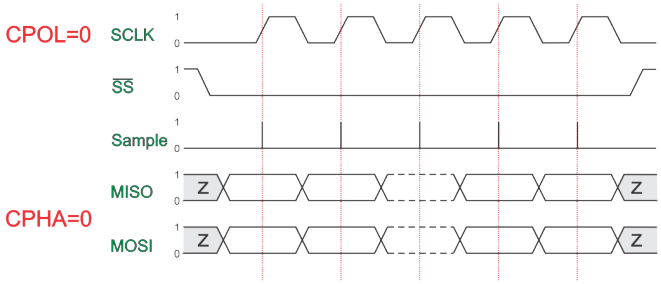
SPI in half duplex single write mode

5.3. Transfer Modes

SPI interface allows to transmit and receive data simultaneously on two lines (MOSI and MISO). Clock polarity (CPOL) and clock phase (CPHA) are the main parameters that define a clock format to be used

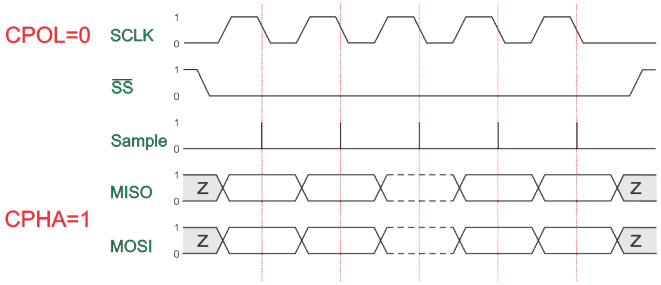
by the SPI system. Depending on CPOL parameter, SPI clock may be inverted or non-inverted. CPHA parameter is used to shift the sampling phase. If CPHA=0 the data are sampled on the leading (first) clock edge. If CPHA=1 the data are sampled on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling.

CPOL=0, CPHA=0



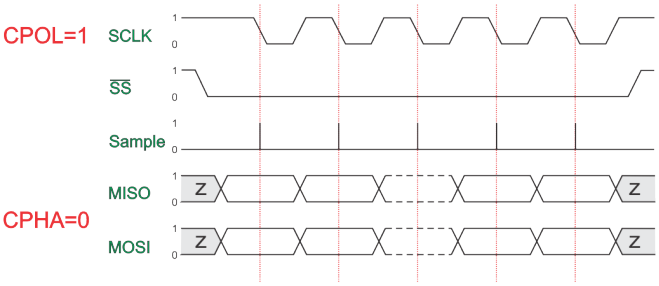
The data must be available before the first clock signal rising. The clock idle state is zero. The data on MISO and MOSI lines must be stable while the clock is high and can be changed when the clock is low. The data is captured on the clock's low-to-high transition and propagated on high-to-low clock transition.

CPOL=0, CPHA=1



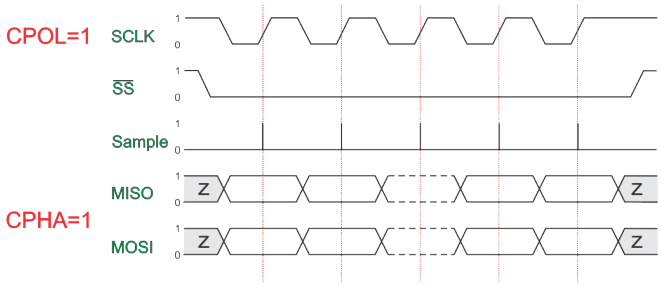
The first clock signal rising can be used to prepare the data. The clock idle state is zero. The data on MISO and MOSI lines must be stable while the clock is low and can be changed when the clock is high. The data is captured on the clock's high-to-low transition and propagated on low-to-high clock transition.

CPOL=1, CPHA=0



The data must be available before the first clock signal falling. The clock idle state is one. The data on MISO and MOSI lines must be stable while the clock is low and can be changed when the clock is high. The data is captured on the clock's high-to-low transition and propagated on low-to-high clock transition.

CPOL=1, CPHA=1

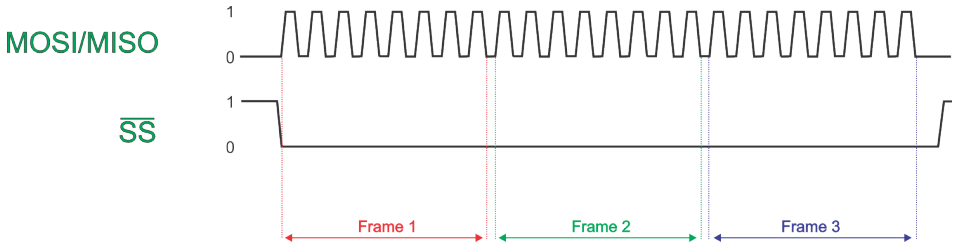


The first clock signal falling can be used to prepare the data. The clock idle state is one. The data on MISO and MOSI lines must be stable while the clock is high and can be changed when the clock is low. The data is captured on the clock's low-to-high transition and propagated on high-to-low clock transition.

5.4. SPI Master Module

5.4.1. Frames

A frame is a single portion of data sent through SPI bus. Some of the DLN adapters support adjustable frame size. The frame data are transferred starting from the most significant bit and up to the least significant bit.



In case the frame size is set to 8 bits or less, only one byte is needed to store the frame data. If the frame size is lesser than 8 bits, the least significant bits of the byte will be transferred, while extra (most significant) bits will be discarded, regardless of their content. The DlnSpiMasterReadWrite() function can be called to transmit 8-bit and smaller frames.

In case a frame size is 9-16 bits, two bytes are needed to store the frame data. You can allocate an array of up to 128 16-bit integers. Such integers must be stored inside the array in Little Endian format. This shouldn't cause a problem, since Little Endian format is used for data storage by most of modern computers. If the frame size is less than 16 bits, 2-byte integers will be used. The least significant bits of the integer will be transferred, while extra (most significant) bits will be discarded, regardless of their content.

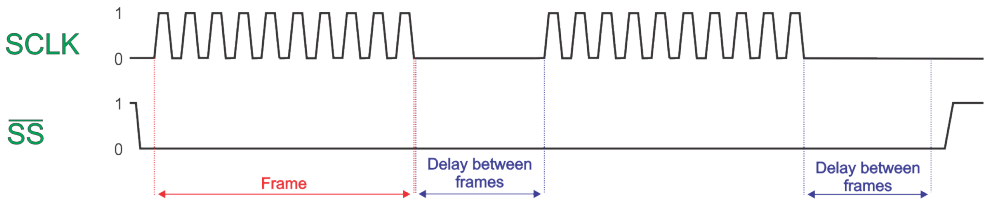
The size of the frame does not limit the size of the buffer to be transferred. DlnSpiMasterReadWrite() (DlnSpiMasterReadWrite16()) function accepts buffer sizes of up to 256 bytes. You can send call the function several times to send a larger buffer.

Splitting the data transfer into frames allows to set delays and configure SS line release between such frames. By default no delays are set and SS line remains asserted all the time.

5.4.2. Delays

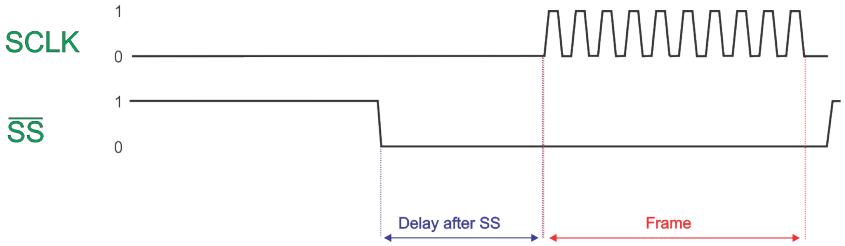
Sometimes slave devices need additional time to process data. In order to provide this time DLN-series adapters can insert delays on different data transmission stages. Those are Delay between frames, Delay after slave selection and Delay between slave selection. All of the delays are set in nanoseconds and have to be adjusted only once, after which they are applied to all the SS lines.

Delay Between Frames



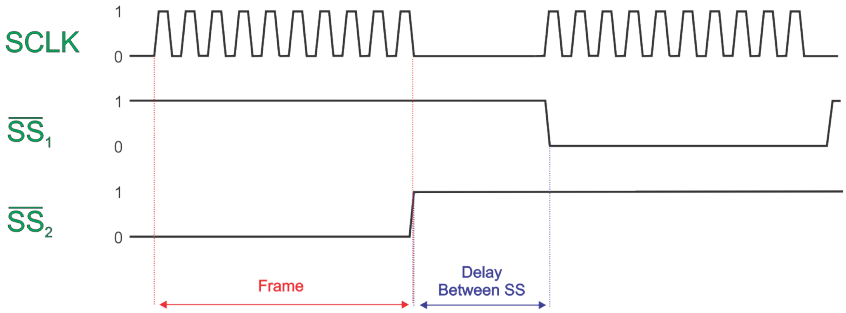
In case a slave device is not fast enough to process continuously incoming data, a user can configure the DLN adapter to insert delays between each two consecutive frames. This gives the slave device additional time to process data received in the previous frame. Once enabled, the delay is inserted after each frame. The delay value is adjusted using the DlnSpiMasterSetDelayBetweenFrames() function. The current Delay Between Frames value can be retrieved by calling the DlnSpiGetDelayBetweenFrames() function.

Delay After Slave Selection



In case a slave device needs additional time for initialization, a Delay After Slave Selection (SS) can be used. When enabled, it is placed after an SS line assertion and before the first frame transmission. The delay value is adjusted using the DlnSpiMasterSetDelayAfterSS() function. A user can retreieve the current delay value by calling the DlnSpiMasterGetDelayAfterSS() function.

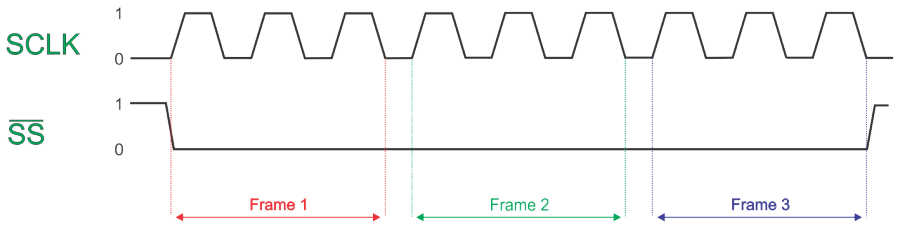
Delay Between Slave Selection



Delay Between Slave Selection (SS) is inserted between one SS line release and assertion of another SS line. The delay value is adjusted using the DlnSpiMasterSetDelayBetweenSS() function, and can be retrieved using the DlnSpiMasterGetDelayBetweenSS() function.

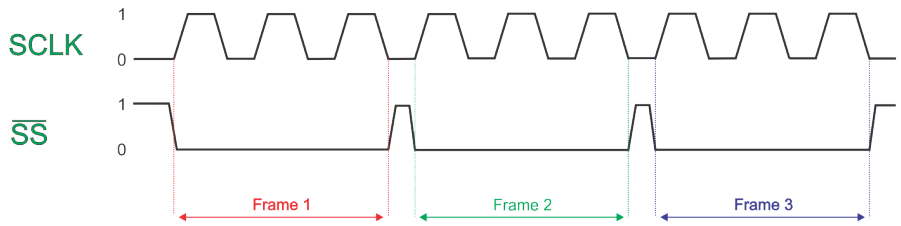
5.4.3. SS Line Release Between Frames

By default, SPI data transmission between a master and a slave looks like this.



SS release between frames is disabled.

However, some slave devices require SS line to be released and deasserted between data frames.



SS release between frames is enabled.

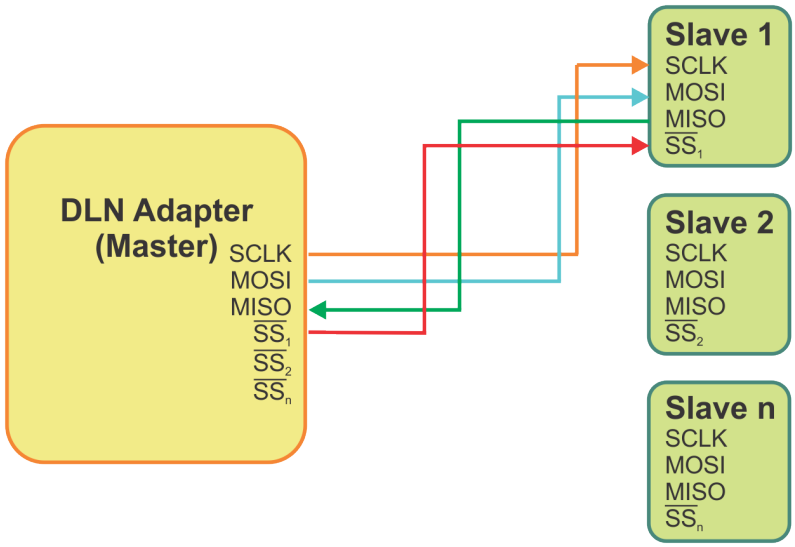
In this case an SS line is released after each frame and asserted before the next frame. If Delay Between Frames is adjusted, the SS line will be released for the whole delay duration. Yet, the line will still be released and asserted between data frames even if this delay is zero.

5.4.4. SPI Slave Selection

Each DLN-series adapter is fitted with four SS lines. An SS line is used to select a slave device. Therefore up to four slave devices can be adressed via SPI bus. Slave device is selected by calling the DlnSpiMasterSetSS() function.

Normal Mode

If you use the DLN-series adapter in Full Duplex or Half Duplex (Single Read) mode, you can only select one slave device at a time.

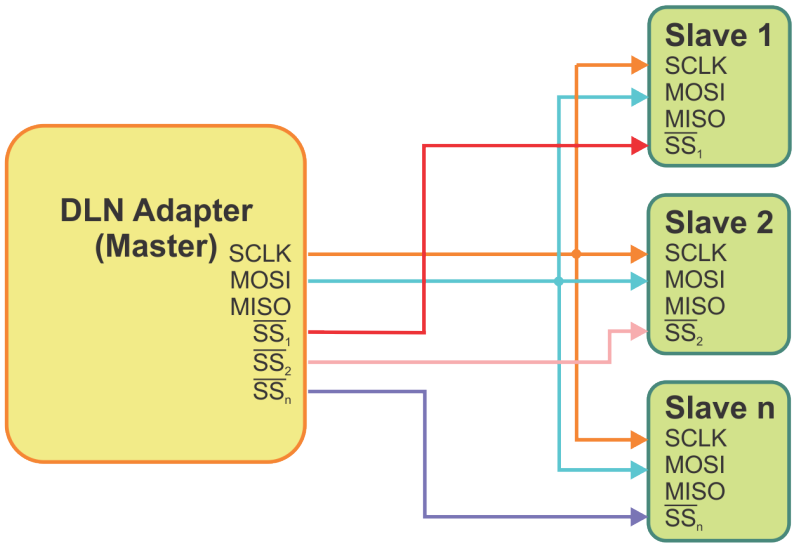


In order to deactivate all devices send a `0xFF` (1111 1111) value. A slave device can be selected by changing the respective bit value to 0.

Example: Slave device #01 can be selected by calling the DlnSpiMasterSetSS() function and passing `0xFD` (1111 1101) value as the `ss` parameter.

Selecting Several Slave Devices At Once

If you use the DLN-series adapter in Half Duplex (Single Write) mode, you can activate several slave devices at once.

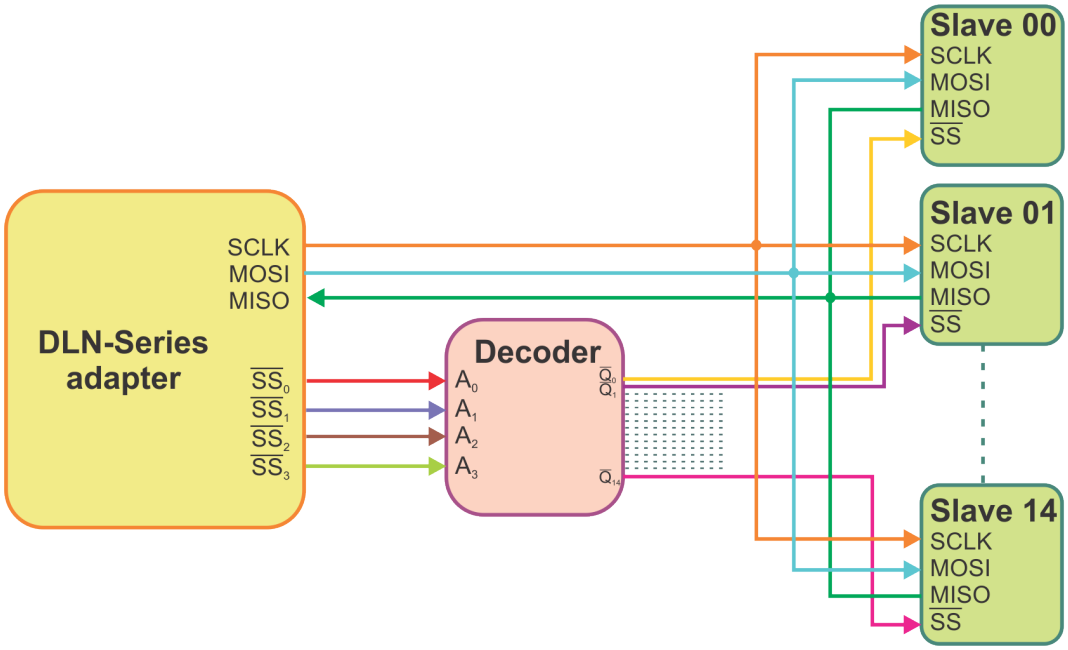


A user can address the required slave devices by changing the respective bit values to 0.

Example: Slave device #00 and #02 can be activated by calling the DlnSpiMasterSetSS() function and passing the `0xFA` (1111 1010) value as the `ss` parameter.

Connecting up to 15 devices

If you connect the ss lines to an external decoder/demultimplexer, you will be able to connect more slave devices to the DLN-series adapter adapter.



In order to use this feature you need an external decoder/demultiplexer (e.g. 74HC4515). With a decoder/demultiplexer connected, the SS lines are used to send a 4-bit value. A user can address the necessary slave by sending its number (values from 0 to 15). The numbers 0 to 14 represent the number of the slave device, while 15 means that no device is selected. This means you can connect up to 15 slave devices to the DLN-series adapter.

Example: Slave device #11, can be activated by calling the DlnSpiMasterSetSS() function and passing the `0xFB` (1111 1011) value as the `ss` parameter.

5.4.5. Functions

The following functions are used to control and monitor the SPI master module of a DLN-series adapter.

- DlnSpiMasterGetPortCount() - retrieves the total number of SPI master ports available in your DLN-series adapter.
- DlnSpiMasterEnable() - activates corresponding SPI master port on your DLN-series adapter.
- DlnSpiMasterDisable() - deactivates corresponding SPI master port on your DLN-Series adapter.
- DlnSpiMasterIsEnabled() - retrieves information whether the specified SPI master port is activated.
- DlnSpiMasterSetMode() - sets SPI transmission parameters (CPOL and CPHA).
- DlnSpiMasterGetMode() - retrieves current configuration of the specified SPI master port.
- DlnSpiMasterSetFrameSize() - sets the size of a single SPI data frame.
- DlnSpiMasterGetFrameSize() - retrieves current size setting for SPI data frames.
- DlnSpiMasterSetFrequency() - sets the SPI clock frequency.
- DlnSpiMasterGetFrequency() - retrieves current setting for SPI clock frequency.
- DlnSpiMasterReadWrite() - sends and receives data via SPI.
- DlnSpiMasterReadWrite16() - sends and receives 2-byte frames.
- DlnSpiMasterSetDelayBetweenSS() - sets a minimum delay between release of an SS line and assertion of another SS line.
- DlnSpiMasterGetDelayBetweenSS() - retrieves current setting for minimum delay between release of an SS line and assertion of another SS line.
- DlnSpiMasterSetDelayAfterSS() - sets a delay duration between assertion of an SS line and first data frame.
- DlnSpiMasterGetDelayAfterSS() - retrieves current setting for minimum delay between assertion of an SS line and first data frame.
- DlnSpiMasterSetDelayBetweenFrames() - sets a delay between data frames exchanged with a single slave device.
- DlnSpiMasterGetDelayBetweenFrames() - retrieves current setting for delay between data frames exchanged with a single slave device.
- DlnSpiMasterSetSS() - selects a Slave Select (SS) line.
- DlnSpiMasterGetSS() - retrieves current Slave Select (SS)line.
- DlnSpiMasterSSBetweenFramesEnable() - enables release of an SS line between data frames exchanged with a single slave device.
- DlnSpiMasterSSBetweenFramesDisable() - disables release of an SS line between data frames exchanged with a single slave device.
- DlnSpiMasterSSBetweenFramesIsEnabled() - disables release of an SS line between data frames exchanged with a single slave device.

  Actual control of the device is performed by use of commands and responses. Each function encapsulates respective commands and responses. You can send such commands directly if necessary.

5.4.5.1. DlnSpiMasterGetPortCount()

```
DLN_RESULT DlnSpiMasterGetPortCount(
   HDLN handle,
   uint8_t* count
);
```

The `DlnSpiMasterGetPortCount()` function retrieves the total number of SPI master ports available in your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. This integer will be filled with the number of available SPI master ports after the function execution.

This function is defined in the `dln_spi_master.h` file.

5.4.5.2. DlnSpiMasterEnable()

```
DLN_RESULT DlnSpiMasterEnable(
   HDLN handle,
   uint8_t port,
   uint16_t* conflict
);
```

The `DlnSpiMasterEnable()` function activates corresponding SPI master port on your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be enabled .

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used by the SPI module. To fix this a user has to disconnect a pin from a module that it has been assigned to and call the DlnSpiMasterEnable() function once again. If there are any more conflicting pins, the next conflicted pin number will be returned.

This function is defined in the `dln_spi_master.h` file.

5.4.5.3. DlnSpiMasterDisable()

```
DLN_RESULT DlnSpiMasterDisable(
   HDLN handle,
   uint8_t port,
```

```
    uint8_t waitForTransferCompletion
);
```

The `DlnSpiMasterDisable()` function deactivates corresponding SPI master port on your DLN-Series adapter.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be disabled.

waitForTransferCompletion
Used to choose whether the device should wait for current data transfers to complete before disabling as SPI master. The following values are available:

- `1` or `DLN_SPI_MASTER_WAIT_FOR_TRANSFERS` - wait until transfers are completed;
- `0` or `DLN_SPI_MASTER_CANCEL_TRANSFERS` - cancel all pending data transfers and deactivate the module.

This function is defined in the `dln_spi_master.h` file.

5.4.5.4. DlnSpiMasterIsEnabled()

```
DLN_RESULT DlnSpiMasterIsEnabled(
    HDLN handle,
    uint8_t port,
    uint8_t* enabled
);
```

The `DlnSpiMasterIsEnabled()` function retrieves information whether the specified SPI master port is activated.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with information whether the specified SPI master port is activated after the function execution. There are two possible values:

- `0` or `DLN_SPI_MASTER_DISABLED` - the port is not configured as SPI master.
- `1` or `DLN_SPI_MASTER_ENABLED` - the port is configured as SPI master.

This function is defined in the `dln_spi_master.h` file.

5.4.5.5. DlnSpiMasterSetMode()

```
DLN_RESULT DlnSpiMasterSetMode(
    HDLN handle,
    uint8_t port,
    uint8_t mode
```

```
);
```

The `DlnSpiMasterSetMode()` function sets SPI transmission parameters (CPOL and CPHA).

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to apply configuration to.

mode
A bit field consisting of 8 bits. The bits `0` and `1` correspond to CPOL and CPHA parameters respectively and define the SPI mode. The rest of the bits is not used. You can also use special constants, defined in the `dln_spi_master.h` file for each of the bits. See Transfer Modes for additional info.

Bit	Value	Description	Constant
0	0	CPOL=0	DLN_SPI_MASTER_CPOL_0
0	1	CPOL=1	DLN_SPI_MASTER_CPOL_1
1	0	CPHA=0	DLN_SPI_MASTER_CPHA_0
1	1	CPHA=1	DLN_SPI_MASTER_CPHA_1

This function is defined in the `dln_spi_master.h` file.

5.4.5.6. DlnSpiMasterGetMode()

```
DLN_RESULT DlnSpiMasterGetMode(
   HDLN handle,
   uint8_t port,
   uint8_t* mode
);
```

The `DlnSpiMasterGetMode()` function retrieves current configuration of the specified SPI master port.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

mode
A pointer to an unsigned 8 bit integer. This integer will be filled with the SPI mode description after the function execution. See DlnSpiMasterSetMode() for details.

This function is defined in the `dln_spi_master.h` file.

5.4.5.7. DlnSpiMasterSetFrameSize()

```
DLN_RESULT DlnSpiMasterSetFrameSize(
   HDLN handle,
   uint8_t port,
   uint8_t frameSize
);
```

The `DlnSpiMasterSetFrameSize()` function sets the size of a single SPI data frame.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

frameSize
A number of bits to be transferred. The DLN-series adapter supports 8 to 16 bytes per frame.

This function is defined in the `dln_spi_master.h` file.

5.4.5.8. DlnSpiMasterGetFrameSize()

```
DLN_RESULT DlnSpiMasterGetFrameSize(
   HDLN handle,
   uint8_t port,
   uint8_t* frameSize
);
```

The `DlnSpiMasterGetFrameSize()` function retrieves current size setting for SPI data frames.

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

frameSize
A number of bits to be transferred in a single frame. The DLN-series adapter supports 8 to 16 bits per transfer.

This function is defined in the `dln_spi_master.h` file.

5.4.5.9. DlnSpiMasterSetFrequency()

```
DLN_RESULT DlnSpiMasterSetFrequency(
   HDLN handle,
   uint8_t port,
   uint32_t frequency,
   uint32_t* actualFrequency
);
```

The `DlnSpiMasterSetFrequency()` function sets the clock frequency on the **SCLK** line, used to synchronize data transfer between master and slave devices.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

frequency
SCLK line frequency value, specified in Hz. A user may specify any value within the range, supported by the DLN-series adapter. This range can be retrieved using the respective function. In case a user enters an incompatible value, it will be approximated as the closest lower frequency value, supported by the adapter.

actualFrequency
A pointer to an unsigned 32-bit integer. This integer will be filled with the frequency approximated as the closest to user-defined lower value. The value is specified in Hz and can be null.

This function is defined in the `dln_spi_master.h` file.

5.4.5.10. DlnSpiMasterGetFrequency()

```
DLN_RESULT DlnSpiMasterGetFrequency(
   HDLN handle,
   uint8_t port,
   uint32_t* frequency
);
```

The `DlnSpiMasterGetFrequency()` function retrieves current setting for SPI clock frequency.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

frequency
A pointer to an unsigned 32-bit integer. This integer will be filled with current SPI clock frequency value in Hz after the function execution.

This function is defined in the `dln_spi_master.h` file.

5.4.5.11. DlnSpiMasterReadWrite()

```
DLN_RESULT DlnSpiMasterReadWrite(
   HDLN handle,
   uint8_t port,
   uint16_t size,
   uint8_t* writeBuffer,
   uint8_t* readBuffer
);
```

The `DlnSpiMasterReadWrite()` function sends and receives data via SPI bus. The data are received as an array of 1-byte elements. This function is suited to transmit data frames of 8 bits or less. In case you set a frame size of more than 8 bits, it is advised to use the DlnSpiMasterReadWrite16() function.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port.

size
The size of the message buffer. This parameter is specified in bytes. The maximum value is 256 bytes.

writeBuffer
A pointer to an array of unsigned 8-bit integers. This array must be filled with data to be sent to a slave during the function execution.

readBuffer
A pointer to an array of unsigned 8-bit integers. This array will be filled with data recieved from slave during the function execution.

This function is defined in the `dln_spi_master.h` file.

5.4.5.12. DlnSpiMasterReadWrite16()

```
DLN_RESULT DlnSpiMasterReadWrite16(
   HDLN handle,
   uint8_t port,
   uint16_t count,
   uint16_t* writeBuffer,
   uint16_t* readBuffer
);
```

The `DlnSpiMasterReadWrite16()` function sends and receives 2-byte frames via SPI.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port.

count
The number of 2-byte array elements.

writeBuffer
A pointer to an array of unsigned 16-bit integer. This array must be filled with data to be sent to slave during the function execution.

readBuffer
A pointer to an array unsigned 16-bit integer. This array will be filled with data received from slave during the function execution.

This function is defined in the `dln_spi_master.h` file.

5.4.5.13. DlnSpiMasterSetDelayBetweenSS()

```
DLN_RESULT DlnSpiMasterSetDelayBetweenSS(
   HDLN handle,
   uint8_t port,
   uint32_t delayBetweenSS,
   uint32_t* actualDelayBetweenSS
);
```

The `DlnSpiMasterSetDelayBetweenSS()` function sets a minimum delay between release of an SS line and assertion of another SS line.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

delayBetweenSS
The delay value in nanoseconds. In case a user specifies an unsupported value, it will be approximated as the closest higher supported value.

actualDelayBetweenSS
Actual set delay value in nanoseconds.

This function is defined in the `dln_spi_master.h` file.

5.4.5.14. DlnSpiMasterGetDelayBetweenSS()

```
DLN_RESULT DlnSpiMasterGetDelayBetweenSS(
   HDLN handle,
   uint8_t port,
   uint32_t* delayBetweenSS
);
```

The `DlnSpiMasterGetDelayBetweenSS()` function retrieves current setting for minimum delay between release of an SS line and assertion of another SS line.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

delayBetweenSS
A pointer to an unsigned 32-bit integer. The integer will be filled with current delay value in nanoseconds.

This function is defined in the `dln_spi_master.h` file.

5.4.5.15. DlnSpiMasterSetDelayAfterSS()

```
DLN_RESULT DlnSpiMasterSetDelayAfterSS(
   HDLN handle,
   uint8_t port,
   uint32_t delayAfterSS,
   uint32_t* actualDelayAfterSS
);
```

The `DlnSpiMasterSetDelayAfterSS()` function sets a delay duration between assertion of an SS line and first data frame.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

delayAfterSS
The delay value in nanoseconds. In case a user specifies an unsupported value, it will be approximated as the closest higher supported value.

> **Warning**
> In case a user sets a 0ns delay time, the actual delay will be equal to 1/2 of the SPI clock frequency, specified using the DlnSpiMasterSetFrequency() function.

actualDelayAfterSS
Actual set delay value in nanoseconds.

This function is defined in the `dln_spi_master.h` file.

5.4.5.16. DlnSpiMasterGetDelayAfterSS()

```
DLN_RESULT DlnSpiMasterGetDelayAfterSS(
   HDLN handle,
   uint8_t port,
   uint32_t* delayAfterSS
);
```

The `DlnSpiMasterGetDelayAfterSS()` function retrieves current setting for minimum delay between assertion of an SS line and first data frame.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

delayAfterSS
A pointer to an unsigned 32-bit integer. The integer will be filled with current delay value in nanoseconds.

This function is defined in the `dln_spi_master.h` file.

5.4.5.17. DlnSpiMasterSetDelayBetweenFrames()

```
DLN_RESULT DlnSpiMasterSetDelayBetweenFrames(
   HDLN handle,
   uint8_t port,
   uint32_t delayBetweenFrames,
   uint32_t* actualDelayBetweenFrames
);
```

The `DlnSpiMasterSetDelayBetweenFrames()` function sets a delay between data frames exchanged with a single slave device.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

delayBetweenFrames
The delay value in nanoseconds. In case a user specifies an unsupported value, it will be approximated as the closest higher supported value.

actualDelayBetweenFrames
Actual set delay value in nanoseconds.

This function is defined in the `dln_spi_master.h` file.

5.4.5.18. DlnSpiMasterGetDelayBetweenFrames()

```
DLN_RESULT DlnSpiMasterGetDelayBetweenFrames(
   HDLN handle,
   uint8_t port,
   uint32_t* delayBetweenFrames
);
```

The `DlnSpiMasterGetDelayBetweenFrames()` function retrieves current setting for delay between data frames exchanged with a single slave device.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

delayBetweenFrames
A pointer to an unsigned 32-bit integer. The integer will be filled with current delay value in nanoseconds.

This function is defined in the `dln_spi_master.h` file.

5.4.5.19. DlnSpiMasterSetSS()

```
DLN_RESULT DlnSpiMasterSetSS(
   HDLN handle,
   uint8_t port,
   uint8_t ss
);
```

The `DlnSpiMasterSetSS()` function selects a Slave Select (SS) line.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

ss
The value on the SS lines. The bits 4-7 are reserved and must be set to 1. See SS Line Decoding for details.

> **Warning**
>
> If you expect slaves to output data, you must ensure that only one slave is activated. If several slaves start outputting data simultaneously, the equipment can be damaged.

This function is defined in the `dln_spi_master.h` file.

5.4.5.20. DlnSpiMasterGetSS()

```
DLN_RESULT DlnSpiMasterGetSS(
   HDLN handle,
   uint8_t port,
   uint8_t* ss
);
```

The `DlnSpiMasterGetSS()` function retrieves current Slave Select (SS)line.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

ss
A pointer to an unsigned 8-bit integer. This integer will be contain the value on the SS lines.

This function is defined in the `dln_spi_master.h` file.

5.4.5.21. DlnSpiMasterSSBetweenFramesEnable()

```
DLN_RESULT DlnSpiMasterSSBetweenFramesEnable(
   HDLN handle,
   uint8_t port
);
```

The `DlnSpiMasterSSBetweenFramesEnable()` function enables release of an SS line between data frames exchanged with a single slave device.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

This function is defined in the `dln_spi_master.h` file.

5.4.5.22. DlnSpiMasterSSBetweenFramesDisable()

```
DLN_RESULT DlnSpiMasterSSBetweenFramesDisable(
   HDLN handle,
   uint8_t port
```

```
);
```

The `DlnSpiMasterSSBetweenFramesDisable()` function disables release of an SS line between data frames exchanged with a single slave device.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

This function is defined in the `dln_spi_master.h` file.

5.4.5.23. DlnSpiMasterSSBetweenFramesIsEnabled()

```
DLN_RESULT DlnSpiMasterSSBetweenFramesIsEnabled(
   HDLN handle,
   uint8_t port,
   uint8_t* enabled
);
```

The `DlnSpiMasterSSBetweenFramesIsEnabled()` function retrieves information whether release of an SS line between data frames exchanged with a slave device is enabled.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with information whether release of an SS line between data frames exchanged with a slave device is enabled after the function execution. The following values are available:

- `1` or `DLN_SPI_MASTER_SS_BETWEEN_TRANSFERS_ENABLED` - SS line release is enabled;
- `0` or `DLN_SPI_MASTER_SS_BETWEEN_TRANSFERS_DISABLED` - SS line release is disabled.

This function is defined in the `dln_spi_master.h` file.

5.4.6. Commands and Responses

5.4.6.1. DLN_SPI_MASTER_GET_PORT_COUNT

DLN_SPI_MASTER_GET_PORT_COUNT Command

Go to response

The **DLN_SPI_MASTER_GET_PORT_COUNT** command is used to retrieve the number of SPI master ports available in your DLN-series adapter.

```
typedef struct
{
```

```
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_SPI_MASTER_GET_PORT_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_PORT_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_PORT_COUNT command it must be set to `0x0200`. You can use the `DLN_MSG_ID_SPI_MASTER_GET_PORT_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_SPI_MASTER_GET_PORT_COUNT Response

Go to Command

The adapter sends the **DLN_SPI_MASTER_GET_PORT_COUNT** response after the command execution. The response contains the available number of SPI master ports.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_SPI_MASTER_GET_PORT_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_GET_PORT_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_PORT_COUNT response it is set to `0x0200`. The `DLN_MSG_ID_SPI_MASTER_GET_PORT_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of the SPI master ports has been successfully obtained.

count
Contains the available number of SPI master ports.

5.4.6.2. DLN_SPI_MASTER_ENABLE

DLN_SPI_MASTER_ENABLE Command

Go to response

The **DLN_SPI_MASTER_ENABLE** command is used to activate the corresponding SPI master port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_SPI_MASTER_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_ENABLE command it must be set to `0x0211`. You can use the `DLN_MSG_ID_SPI_MASTER_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be activated.

DLN_SPI_MASTER_ENABLE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_ENABLE** response after the command execution. The `result` field informs a user if the specified SPI master port has been successfully activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t conflict;
}__PACKED_ATTR DLN_SPI_MASTER_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_ENABLE_RSP structure.

- **msgId** - Defines the message. For the DLN_SPI_MASTER_ENABLE response it is set to `0x0211`. The `DLN_MSG_ID_SPI_MASTER_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified SPI master port has been successfully activated.

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used for the SPI master module. To fix this a user has to disconnect a pin from a module that it has been assigned to and send the DLN_SPI_MASTER_ENABLE command once again. In case there still are conflicted pins, only the number of the next one will be returned.

5.4.6.3. DLN_SPI_MASTER_DISABLE

DLN_SPI_MASTER_DISABLE Command

Go to response

The **DLN_SPI_MASTER_DISABLE** command is used to deactivate the specified SPI master port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t waitForTransferCompletion;
}__PACKED_ATTR DLN_SPI_MASTER_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_DISABLE command it must be set to `0x0212`. You can use the `DLN_MSG_ID_SPI_MASTER_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be deactivated.

waitForTransferCompletion
Used to choose whether the DLN-series adapter should wait for current data transfers to complete before deactivating the SPI master port. The following values are available:

- `1` or `DLN_SPI_MASTER_WAIT_FOR_TRANSFERS` - wait until transfers are completed;
- `0` or `DLN_SPI_MASTER_CANCEL_TRANSFERS` - cancel all pending data transfers and deactivate the module.

DLN_SPI_MASTER_DISABLE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_DISABLE** response after the function execution. The `result` field informs a user if the specified SPI master port has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_SPI_MASTER_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_DISABLE response it is set to `0x0212`. The `DLN_MSG_ID_SPI_MASTER_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified SPI master port has been successfully deactivated.

5.4.6.4. DLN_SPI_MASTER_IS_ENABLED

DLN_SPI_MASTER_IS_ENABLED Command

Go to response

The **DLN_SPI_MASTER_IS_ENABLED** command is used to retrieve information, whether the specified SPI master port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_SPI_MASTER_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_IS_ENABLED command it must be set to `0x0213`. You can use the `DLN_MSG_ID_SPI_MASTER_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

DLN_SPI_MASTER_IS_ENABLED Response

Go to command

The adapter sends the **DLN_SPI_MASTER_IS_ENABLED** response after the command execution. The response informs a user if the specified SPI master port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_SPI_MASTER_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_IS_ENABLED response it is set to `0x0213`. The `DLN_MSG_ID_SPI_MASTER_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the SPI master port has been successfully retrieved.

enabled
Informs whether the specified SPI master port is activated. There are two possible values:

- `0` or `DLN_SPI_MASTER_DISABLED` - the SPI master port is activated.
- `1` or `DLN_SPI_MASTER_ENABLED` - the SPI master port is deactivated.

5.4.6.5. DLN_SPI_MASTER_SET_MODE

DLN_SPI_MASTER_SET_MODE Command

Go to response

The **DLN_SPI_MASTER_SET_MODE** command is used to set SPI transmission parameters (CPOL and CPHA).

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t mode;
}__PACKED_ATTR DLN_SPI_MASTER_SET_MODE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SET_MODE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_MODE command it must be set to `0x0214`. You can use the `DLN_MSG_ID_SPI_MASTER_SET_MODE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to apply configuration to.

mode
A bit field consisting of 8 bits. The bits `0` and `1` correspond to CPOL and CPHA parameters respectively and define the SPI mode. The rest of the bits is not used. You can also use special constants, defined in the `dln_spi_master.h` file for each of the bits. See Transfer Modes for additional info.

Bit	Value	Description	Constant
0	0	CPOL=0	DLN_SPI_MASTER_CPOL_0
0	1	CPOL=1	DLN_SPI_MASTER_CPOL_1
1	0	CPHA=0	DLN_SPI_MASTER_CPHA_0
1	1	CPHA=1	DLN_SPI_MASTER_CPHA_1

DLN_SPI_MASTER_SET_MODE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SET_MODE** response after the command execution. The `result` field informs a user whether the configuration has been successfully changed.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_SPI_MASTER_SET_MODE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_MODE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_MODE response it is set to `0x0214`. The `DLN_MSG_ID_SPI_MASTER_SET_MODE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The configuration has been successfully set.

5.4.6.6. DLN_SPI_MASTER_GET_MODE

DLN_SPI_MASTER_GET_MODE Command

Go to response

The **DLN_SPI_MASTER_GET_MODE** command is used to retrieve current configuration of an SPI master port of the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_SPI_MASTER_GET_MODE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_MODE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_MODE command it must be set to `0x0215`. You can use the `DLN_MSG_ID_SPI_MASTER_GET_MODE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

DLN_SPI_MASTER_GET_MODE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_GET_MODE** response after the command execution. The response contains current mode of the specified SPI master port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t mode;
}__PACKED_ATTR DLN_SPI_MASTER_GET_MODE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_GET_MODE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_MODE response it is set to `0x0215`. The `DLN_MSG_ID_SPI_MASTER_GET_MODE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The configuration has been successfully retrieved.

mode
A bit field, consisting of 8 bits and used to describe SPI transmission mode. See DLN_SPI_MASTER_SET_MODE Command for details.

5.4.6.7. DLN_SPI_MASTER_SET_FRAME_SIZE

DLN_SPI_MASTER_SET_FRAME_SIZE Command

Go to response

The **DLN_SPI_MASTER_SET_FRAME_SIZE** command is used to set the size of a single portion of data transferred via SPI.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t frameSize;
```

```
} __PACKED_ATTR DLN_SPI_MASTER_SET_FRAME_SIZE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SET_FRAME_SIZE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_TRANSFER_SIZE command it must be set to `0x0216`. You can use the `DLN_MSG_ID_SPI_MASTER_SET_FRAME_SIZE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

frameSize
A number of bytes to be transferred, specified in bits. The DLN-series adapter supports 8 to 16 bits per frame.

DLN_SPI_MASTER_SET_FRAME_SIZE Response

Go to command

The device sends the **DLN_SPI_MASTER_SET_FRAME_SIZE** response after the command execution. The response informs a user if the frame size has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SPI_MASTER_SET_FRAME_SIZE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_FRAME_SIZE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_FRAME_SIZE response it is set to `0x0216`. The `DLN_MSG_ID_SPI_MASTER_SET_FRAME_SIZE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The number of bytes per frame has been successfully set.

5.4.6.8. DLN_SPI_MASTER_GET_FRAME_SIZE

DLN_SPI_GET_FRAME_SIZE Command

Go to response

The **DLN_SPI_MASTER_GET_FRAME_SIZE** command is used to retrieve current size setting for a single portion of data transferred via SPI.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_MASTER_GET_FRAME_SIZE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_FRAME_SIZE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_FRAME_SIZE command it must be set to `0x0217`. You can use the `DLN_MSG_ID_SPI_MASTER_GET_FRAME_SIZE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to get the information from.

DLN_SPI_MASTER_GET_FRAME_SIZE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_GET_FRAME_SIZE** response after the command execution. The response contains current setting for the number of bits per transfer.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t frameSize;
} __PACKED_ATTR DLN_SPI_MASTER_GET_FRAME_SIZE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_FRAME_SIZE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_FRAME_SIZE response it is set to `0x0217`. The `DLN_MSG_ID_SPI_MASTER_GET_FRAME_SIZE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The number of bytes per frame has been successfully retrieved.

frameSize
A number of bits to be transferred. The DLN-series adapter supports 8 to 16 bytes per frame.

5.4.6.9. DLN_SPI_MASTER_SET_FREQUENCY

DLN_SPI_MASTER_SET_FREQUENCY Command

Go to response

The **DLN_SPI_MASTER_SET_FREQUENCY** command is used to set the clock frequency on the **SCLK** line, used to synchronize data transfer between master and slave devices.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint32_t frequency;
} __PACKED_ATTR DLN_SPI_MASTER_SET_FREQUENCY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SET_FREQUENCY_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_FREQUENCY command it must be set to `0x0218`. You can use the `DLN_MSG_ID_SPI_MASTER_SET_FREQUENCY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

frequency
SCLK line frequency value, specified in Hz. A user may specify any value within the range, supported by the DLN-series adapter. This range can be retrieved using the respective function. In case a user

enters an incompatible value, it will be approximated as the closest lower frequency value, supported by the adapter.

DLN_SPI_MASTER_SET_FREQUENCY Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SET FREQUENCY** response after the command execution. The response informs a user if the SPI clock frequency has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t frequency;
} __PACKED_ATTR DLN_SPI_MASTER_SET_FREQUENCY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_FREQUENCY_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_FREQUENCY response it is set to `0x0218`. The `DLN_MSG_ID_SPI_MASTER_SET_FREQUENCY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The clock frequency has been successfully set.
- `DLN_RES_SPI_MASTER_BUSY_TRANSFERRING` - The SPI module is busy transferring data and cannot currently be configured.

frequency
Contains the frequency approximated as the closest to user-defined lower value. The value is specified in Hz.

5.4.6.10. DLN_SPI_MASTER_GET_FREQUENCY

DLN_SPI_MASTER_GET_FREQUENCY Command

Go to response

The **DLN_SPI_GET_FREQUENCY** command is used to retrieve current setting for SPI clock frequency.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
```

```
} __PACKED_ATTR DLN_SPI_MASTER_GET_FREQUENCY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_FREQUENCY_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_FREQUENCY command it must be set to `0x0219`. You can use the `DLN_MSG_ID_SPI_MASTER_GET_FREQUENCY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve information from.

DLN_SPI_MASTER_GET_FREQUENCY Response

Go to command

The adapter sends the **DLN_SPI_MASTER_GET_FREQUENCY** response after the command execution. The response contains current SPI clock frequency setting.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t frequency;
} __PACKED_ATTR DLN_SPI_MASTER_GET_FREQUENCY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_GET_FREQUENCY_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_FREQUENCY response it is set to `0x0219`. The `DLN_MSG_ID_SPI_MASTER_GET_FREQUENCY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Current SPI clock frequency has been successfully retrieved.

- `DLN_RES_SPI_MASTER_BUSY_TRANSFERRING` - The SPI module is busy transferring data and cannot currently be configured.

frequency
Contains current SPI clock frequency value in Hz.

5.4.6.11. DLN_SPI_MASTER_READ_WRITE

DLN_SPI_MASTER_READ_WRITE Command

Go to response

The **DLN_SPI_MASTER_READ_WRITE** command is used to send and receive data via SPI-bus.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint16_t size;
   uint8_t attribute;
   uint8_t buffer[256];
} __PACKED_ATTR DLN_SPI_MASTER_READ_WRITE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Can be less than the size of the DLN_SPI_MASTER_READ_WRITE_CMD structure. The value can be calculated using the following formula:

  ```
  header.size = sizeof(DLN_SPI_READ_WRITE_CMD) - 256 + size
  ```

  Setting the size of the message equal to the size of the DLN_SPI_MASTER_READ_WRITE_CMD structure will not cause an error. However, more data will be transferred via USB. The number of bytes, included in the `size` field will be transferred through SPI.

- **msgId** - Defines the message. For the DLN_SPI_MASTER_READ_WRITE command it must be set to `0x021A`. You can also use the `DLN_MSG_ID_SPI_MASTER_READ_WRITE` constant.

- **echoCounter** - Can be used to link a command to a response. The response to the command will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be used.

size
A size of the message buffer. This parameter is specified in bytes. The maximum value is 256 bytes.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer must contain the information to be sent to a slave.

DLN_SPI_READ_WRITE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_READ_WRITE** response after the command execution. The response contains confirmation of data transfer as well as data received from a slave.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t size;
   uint8_t buffer[256];
} __PACKED_ATTR DLN_SPI_MASTER_READ_WRITE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_READ_WRITE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_READ_WRITE response it is set to `0x021A`. The `DLN_MSG_ID_SPI_MASTER_READ_WRITE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The data have been successfully transmitted.
- `DLN_RES_INVALID_BUFFER_SIZE` - The buffer size is beyond 1 to 256 limits.

size
The size of the message buffer specified in bytes.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer contains the information received from a slave.

5.4.6.12. DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS

DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS Command

Go to response

The **DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS** command is used to set a minimum delay between release of an SS line and assertion of another SS line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
```

```
   uint32_t delayBetweenSS;
} __PACKED_ATTR DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS command it must be set to `0x0220`. You can use the `DLN_MSG_ID_SPI_MASTER_SET_DELAY_BETWEEN_SS` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

delayBetweenSS
The delay value in nanoseconds. In case a user specifies an unsupported value, it will be approximated as the closest higher supported value. The DLN-series adapter will return the actual delay time in the response to this command.

DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS** response after the command execution. The `result` field informs a user if the minimum delay between release of one SS line and assertion of another SS line has been set successfully. The response also contains actual delay setting, in case a user has specified an incompatible value.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t delayBetweenSS;
} __PACKED_ATTR DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS response it is set to `0x0220`. The `DLN_MSG_ID_SPI_MASTER_SET_DELAY_BETWEEN_SS` constant can be used.

- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The delay setting has been successfully configured.

delayBetweenSS
Actual set delay value in nanoseconds.

5.4.6.13. DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS

DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS Command

Go to response

The **DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS** command is used to retrieve current setting for minimum delay between release of one SS line and assertion of another SS line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   } __PACKED_ATTR DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS command it must be set to `0x0221`. You can use the `DLN_MSG_ID_SPI_MASTER_GET_DELAY_BETWEEN_SS` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the delay setting from.

DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS Response

Go to command

The adapter sends the **DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS** response after the command execution. The response contains current setting for minimum delay between release of one SS line and assertion of another SS line.

```
typedef struct
```

```
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t delayBetweenSS;
} __PACKED_ATTR DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_DELAY_BETWEEN_SS response it is set to `0x0221`. The `DLN_MSG_ID_SPI_MASTER_GET_DELAY_BETWEEN_SS` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The current delay setting has been successfully retrieved.

delayBetweenSS
Current delay value in nanoseconds.

5.4.6.14. DLN_SPI_MASTER_SET_DELAY_AFTER_SS

DLN_SPI_MASTER_SET_DELAY_AFTER_SS Command

Go to response

The **DLN_SPI_MASTER_SET_DELAY_AFTER_SS** command is used to set a delay duration between assertion of an SS line and first data frame.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint32_t delayAfterSS;
} __PACKED_ATTR DLN_SPI_MASTER_SET_DELAY_AFTER_SS_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SET_DELAY_AFTER_SS_CMD structure.

- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_DELAY_AFTER_SS command it must be set to `0x0222`. You can use the `DLN_MSG_ID_SPI_MASTER_SET_DELAY_AFTER_SS` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

delayAfterSS
The delay value in nanoseconds. In case a user specifies an unsupported value, it will be approximated as the closest lesser supported value. The DLN-series adapter will return the actual delay time in the response to this command.

Warning

In case a user sets a 0ns delay time, the actual delay will be equal to 1/2 of the clock frequency, specified using the DLN_SPI_MASTER_SET_FREQUENCY Command.

DLN_SPI_MASTER_SET_DELAY_BETWEEN_SS Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SET_DELAY_AFTER_SS** response after the command execution. The `result` field informs a user if the delay between assertion of an SS line and first data transfer has been set successfully. The response also contains actual delay setting, in case a user has specified an incompatible value.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t delayAfterSS;
} __PACKED_ATTR DLN_SPI_MASTER_SET_DELAY_AFTER_SS_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_DELAY_AFTER_SS_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_DELAY_AFTER_SS response it is set to `0x0222`. The `DLN_MSG_ID_SPI_MASTER_SET_DELAY_AFTER_SS` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- DLN_RES_SUCCESS - The delay setting has been successfully configured.

delayAfterSS
Actual set delay value in nanoseconds.

5.4.6.15. DLN_SPI_MASTER_GET_DELAY_AFTER_SS

DLN_SPI_MASTER_GET_DELAY_AFTER_SS Command

Go to response

The **DLN_SPI_MASTER_GET_DELAY_AFTER_SS** command is used to retrieve current setting for delay between assertion of an SS line and first data frameslave.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_MASTER_GET_DELAY_AFTER_SS_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_DELAY_AFTER_SS_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_DELAY_AFTER_SS command it must be set to 0x0223. You can use the DLN_MSG_ID_SPI_MASTER_GET_DELAY_AFTER_SS constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same echoCounter value. This field can be any numerical value from 0 to 0xFFFF (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the delay setting from.

DLN_SPI_MASTER_GET_DELAY_AFTER_SS Response

Go to command

The adapter sends the **DLN_SPI_MASTER_GET_DELAY_AFTER_SS** response after the command execution. The response contains current setting for delay between assertion of an SS line and first data frame.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t delayAfterSS;
} __PACKED_ATTR DLN_SPI_MASTER_GET_DELAY_AFTER_SS_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_GET_DELAY_AFTER_SS_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_DELAY_AFTER_SS response it is set to `0x0223`. The `DLN_MSG_ID_SPI_MASTER_GET_DELAY_AFTER_SS` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Current delay setting has been successfully retrieved.

delayAfterSS
Current delay value in nanoseconds.

5.4.6.16. DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES

DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES Command

Go to response

The **DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES** command is used to set a delay between data frames on a single SS line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint32_t delayBetweenFrames;
} __PACKED_ATTR DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES command it must be set to `0x0224`. You can use the `DLN_MSG_ID_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured

delayBetweenFrames
The delay value in nanoseconds. In case a user specifies an unsupported value, it will be approximated as the closest lesser supported value. The DLN-series adapter will return the actual delay time in the response to this command.

DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES** response after the command execution. The `result` field informs a user if the minimum delay between data frames on a single SS line has been set successfully. The response also contains actual delay setting, in case a user has specified an incompatible value.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t delayBetweenFrames;
   } __PACKED_ATTR DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES response it is set to `0x0224`. The `DLN_MSG_ID_SPI_MASTER_SET_DELAY_BETWEEN_FRAMES` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The delay setting has been successfully configured.

delayBetweenFrames
Actual set delay value in nanoseconds.

5.4.6.17. DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES

DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES Command

Go to response

The **DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES** command is used to retrieve current setting for delay between data frames on a single SS line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES command it must be set to `0x0225`. You can use the `DLN_MSG_ID_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the delay setting from.

DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES Response

Go to command

The adapter sends the **DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES** response after the command execution. The response contains current setting for delay between data frames on a single SS line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t delayBetweenFrames;
} __PACKED_ATTR DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES response it is set to `0x0225`. The `DLN_MSG_ID_SPI_MASTER_GET_DELAY_BETWEEN_FRAMES` constant can be used.

- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Current delay setting has been successfully retrieved.

delayBetweenFrames
Current delay value in nanoseconds.

5.4.6.18. DLN_SPI_MASTER_SET_SS

DLN_SPI_MASTER_SET_SS Command

Go to response

The **DLN_SPI_MASTER_SET_SS** command is used to select a Slave Select (SS) line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t ss;
} __PACKED_ATTR DLN_SPI_MASTER_SET_SS_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SET_SS_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_SS command it must be set to `0x0226`. You can use the `DLN_MSG_ID_SPI_MASTER_SET_SS` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

ss
An 8-bit mask, containing the value on the SS lines. The bits 4-7 are reserved and must be set to 1. For Details see Slave Selection.

Warning

If you expect slaves to output data, you must ensure that only one slave is activated. If several slaves start outputting data simultaneously, the equipment can be damaged.

If you use Slave Selection, with an external decoder/multiplexer connected, the SS lines are used to encode the number of slave device. The numbers 0 to 14 represent the number of the slave device, while 15 means that no device is selected.

DLN_SPI_MASTER_SET_SS Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SET_SS** response after the command execution. The `result` field informs a user if a Slave Select (SS) line has been successfully selected.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SPI_MASTER_SET_SS_RSP;
```

Parameters:

header

Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SET_SS_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_SS response it is set to `0x0226`. The `DLN_MSG_ID_SPI_MASTER_SET_SS` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result

Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The SS line has been successfully selected.

5.4.6.19. DLN_SPI_MASTER_GET_SS

DLN_SPI_MASTER_GET_SS Command

Go to response

The **DLN_SPI_MASTER** command is used to retrieve current Slave Select (SS)line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_MASTER_GET_SS_CMD;
```

Parameters:

header

Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_GET_SS_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_SS command it must be set to `0x0227`. You can use the `DLN_MSG_ID_SPI_MASTER_GET_SS` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to retrieve the information from.

DLN_SPI_MASTER_GET_SS Response

Go to command

The adapter sends the **DLN_SPI_MASTER_GET_SS** response after the command execution. The response contains currently selected SS line.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t ss;
} __PACKED_ATTR DLN_SPI_MASTER_GET_SS_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_GET_SS_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_GET_SS response it is set to `0x0227`. The `DLN_MSG_ID_SPI_MASTER_GET_SS` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Current SS line has been successfully retrieved.

ss
Contains the value on the SS lines.

5.4.6.20. DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE

DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE Command

Go to response

The **DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE** command is used to enable release of an SS line between data frames exchanged with a single slave device.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE command it must be set to `0x0231`. You can use the `DLN_MSG_ID_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE** response after the command execution. The `result` field informs a user if release of an SS line between data frames, exchanged with a single slave device, has been successfully enabled.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE_RSP response it is set to `0x0231`. The `DLN_MSG_ID_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLE` constant can be used.

- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Release of an SS line between data frames exchanged with a single slave device has been successfully enabled.

5.4.6.21. DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE

DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE Command

Go to response

The **DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE** command is used to disable release of an SS line between data frames exchanged with a single slave device.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE command it must be set to `0x0232`. You can use the `DLN_MSG_ID_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to be configured.

DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE** response after the command execution. The `result` field informs a user if release of an SS line between data frames exchanged with a single slave device has been successfully disabled.

```
typedef struct
{
```

```
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE_RSP response it is set to `0x0232`. The `DLN_MSG_ID_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Release of an SS line between data frames exchanged with a single slave device has been successfully disabled.

5.4.6.22. DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED

DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED Command

Go to response

The **DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED** command is used to retrieve information whether release of an SS line between data frames exchanged with a single slave device is enabled.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED_CMD;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED command it must be set to `0x0233`. You can use the `DLN_MSG_ID_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI master port to get the information from.

DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED Response

Go to command

The adapter sends the **DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED** response after the command execution. The response informs a user if release of an SS line between data frames exchanged with a single slave device is enabled.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
} __PACKED_ATTR DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED_RSP response it is set to `0x0233`. The `DLN_MSG_ID_SPI_MASTER_SS_BETWEEN_FRAMES_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - state of an SS line release between data frames exchanged with a single slave device has been successfully retrieved.

enabled
Indicates whether release of an SS line between data frames exchanged with a single slave device is enabled. The following values are available:

- `1` or `DLN_SPI_MASTER_SS_BETWEEN_FRAMES_ENABLED` - release of an SS line between data frames exchanged with a single slave device is enabled;
- `0` or `DLN_SPI_MASTER_SS_BETWEEN_FRAMES_DISABLED` - release of an SS line between data frames exchanged with a single slave device is disabled.

5.5. SPI Slave Module

5.5.1. SPI Slave Events

A DLN-series adapter can be configured to send DLN_SPI_SLAVE_TRANSFER_EV events. They are generated when certain predefined conditions are met. The SPI slave events are configured using the DlnSpiSlaveSetEvent() function. The `eventType` parameter defines conditions for event generation. For some of the event types the `bufferSize` parameter can be defined. In this case an event will be sent once buffer size reaches the `bufferSize` value. You can retrieve current event generation conditions by calling the DlnSpiSlaveGetEvent(). There are three types of events:

- DLN_SPI_SLAVE_EVENT_NONE - no events are generated.
- DLN_SPI_SLAVE_EVENT_SS_RISE - events are generated when an SS line is released.
- DLN_SPI_SLAVE_EVENT_SS_BUFFER_FULL - events are generated when the allocated buffer becomes full.

5.5.1.1. DLN_SPI_SLAVE_EVENT_NONE

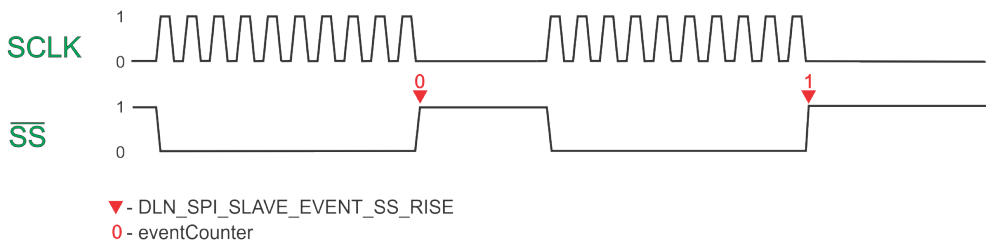
No events are generated.

In order to choose this event type, call the DlnSpiSlaveSetEvent() and set `0` or `SPI_SLAVE_EVENT_NONE` as the `eventType` parameter.

In this case the `bufferSize` parameter is ignored and can be any value.

5.5.1.2. DLN_SPI_SLAVE_EVENT_SS_RISE

Events are generated when an SS line is released. This event type ignores the data transmitted and only monitors the state of the SS line.

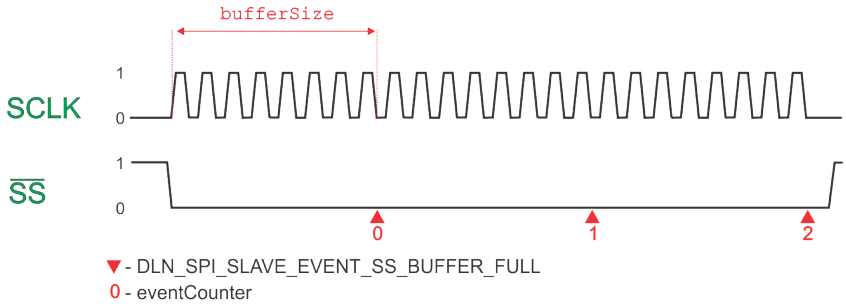


In order to choose this event type, call the DlnSpiSlaveSetEvent() and set `1` or `SPI_SLAVE_EVENT_SS_RISE` as the `eventType` parameter.

In this case the `bufferSize` parameter is ignored and can be any value.

5.5.1.3. DLN_SPI_SLAVE_SS_BUFFER_FULL

Events are generated when the allocated buffer becomes full.



In order to choose this event type, call the DlnSpiSlaveSetEvent() and set `2` or `SPI_SLAVE_EVENT_SS_BUFFER_FULL` as the `eventType` parameter.

In this case the `bufferSize` parameter defines the size of the buffer that stores data received from a master. This parameter is set in bytes. The maximum value is 256 bytes.

5.5.2. Structure

5.5.2.1. DLN_SPI_SLAVE_TRANSFER_EV

This structure is used to store the description of SPI slave events.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t eventCount;
   uint8_t eventType;
   uint8_t port;
   uint16_t size;
   uint8_t buffer[DLN_SPI_SLAVE_BUFFER_SIZE];
} __PACKED_ATTR DLN_SPI_SLAVE_TRANSFER_EV;
```

Members:

header
Defines the DLN message header DLN_MSG_HEADER. This structure's header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_TRANSFER_EV structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_TRANFSER_EV response it is set to `0x0B10`. The `DLN_MSG_ID_SPI_SLAVE_TRANSFER_EV` constant can be used.
- **echoCounter** - .
- **handle** - A handle to the DLN-series adapter.

eventCount
Contains the number of generated events after configuration setting.

eventType
Contains the condition for event generation on the SPI-slave port. The following values are available:

- `0` or `SPI_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `SPI_SLAVE_EVENT_SS_RISE` - events are generated when an SS line is released;
- `2` or `SPI_SLAVE_EVENT_SS_BUFFER_FULL` - an event is generated when the allocated buffer becomes full.

port
A number of the SPI slave port.

size
A size of the buffer that stores the event data. This parameter is specified in bytes. The maximum value us 256 bytes.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer contains the event data.

5.5.3. Functions

5.5.3.1. DlnSpiSlaveGetPortCount()

```
DLN_RESULT DlnSpiSlaveGetPortCount(
   HDLN handle,
   uint8_t* count
);
```

The `DlnSpiSlaveGetPortCount()` function retrieves the total number of SPI slave ports available in your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. This integer will be filled with the number of available SPI slave ports after the function execution.

This function is defined in the `dln_spi_slave.h` file.

5.5.3.2. DlnSpiSlaveEnable()

```
DLN_RESULT DlnSpiSlaveEnable(
   HDLN handle,
   uint8_t port,
   uint16_t* conflict
);
```

The `DlnSpiSlaveEnable()` function activates corresponding SPI slave port on your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to be enabled .

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used by the SPI slave module. To fix this a user has to disconnect a pin from a module that it has been assigned to and call the DlnSpiSlaveEnable() function once again. If there are any more conflicting pins, the next conflicted pin number will be returned.

This function is defined in the `dln_spi_slave.h` file.

5.5.3.3. DlnSpiSlaveDisable()

```
DLN_RESULT DlnSpiSlaveDisable(
   HDLN handle,
   uint8_t port,
   uint8_t waitForTransferCompletion
);
```

The `DlnSpiSlaveDisable()` function deactivates corresponding SPI slave port on your DLN-Series adapter.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to be disabled.

waitForTransferCompletion
Used to choose whether the device should wait for current data transfers to complete before disabling as SPI slave. The following values are available:

- `1` or `DLN_SPI_SLAVE_WAIT_FOR_TRANSFERS` - wait until transfers are completed;
- `0` or `DLN_SPI_SLAVE_CANCEL_TRANSFERS` - cancel all pending data transfers and deactivate the module.

This function is defined in the `dln_spi_slave.h` file.

5.5.3.4. DlnSpiSlaveIsEnabled()

```
DLN_RESULT DlnSpiSlaveIsEnabled(
   HDLN handle,
   uint8_t port,
   uint8_t* enabled
);
```

The `DlnSpiSlaveIsEnabled()` function retrieves information whether the specified SPI master port is activated.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with information whether the specified SPI slave port is activated after the function execution. There are two possible values:

- `0` or `DLN_SPI_SLAVE_DISABLED` - the port is not configured as SPI slave.
- `1` or `DLN_SPI_SLAVE_ENABLED` - the port is configured as SPI slave.

This function is defined in the `dln_spi_slave.h` file.

5.5.3.5. DlnSpiSlaveSetMode()

```
DLN_RESULT DlnSpiSlaveSetMode(
```

```
    HDLN handle,
    uint8_t port,
    uint8_t mode
);
```

The `DlnSpiSlaveSetMode()` function sets SPI transmission parameters (CPOL and CPHA).

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to apply configuration to.

mode
A bit field consisting of 8 bits. The bits `0` and `1` correspond to CPOL and CPHA parameters respectively and define the SPI mode. The rest of the bits is not used. You can also use special constants, defined in the `dln_spi_slave.h` file for each of the bits. See Transfer Modes for additional info.

Bit	Value	Description	Constant
0	0	CPOL=0	DLN_SPI_SLAVE_CPOL_0
0	1	CPOL=1	DLN_SPI_SLAVE_CPOL_1
1	0	CPHA=0	DLN_SPI_SLAVE_CPHA_0
1	1	CPHA=1	DLN_SPI_SLAVE_CPHA_1

This function is defined in the `dln_spi_slave.h` file.

5.5.3.6. DlnSpiSlaveGetMode()

```
DLN_RESULT DlnSpiSlaveGetMode(
    HDLN handle,
    uint8_t port,
    uint8_t* mode
);
```

The `DlnSpiSlaveGetMode()` function retrieves current configuration of the specified SPI slave port.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to retrieve the information from.

mode
A pointer to an unsigned 8 bit integer. This integer will be filled with the SPI mode description after the function execution. See DlnSpiSlaveSetMode() for details.

This function is defined in the `dln_spi_slave.h` file.

5.5.3.7. DlnSpiSlaveSetFrameSize()

```
DLN_RESULT DlnSpiSlaveSetFrameSize(
    HDLN handle,
```

```
    uint8_t port,
    uint8_t frameSize
);
```

The `DlnSpiSlaveSetFrameSize()` function sets the size of a single SPI data frame.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to be configured.

frameSize
A number of bits to be transferred. The DLN-series adapter supports 8 to 16 bytes per frame.

This function is defined in the `dln_spi_slave.h` file.

5.5.3.8. DlnSpiSlaveGetFrameSize()

```
DLN_RESULT DlnSpiSlaveGetFrameSize(
    HDLN handle,
    uint8_t port,
    uint8_t* frameSize
);
```

The `DlnSpiSlaveGetFrameSize()` function retrieves current size setting for SPI data frames.

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to retrieve the information from.

frameSize
A number of bits to be transferred in a single frame. The DLN-series adapter supports 8 to 16 bits per transfer.

This function is defined in the `dln_spi_slave.h` file.

5.5.3.9. DlnSpiSlaveLoadReply()

The `DlnSpiSlaveLoadReply()` function loads information to be transferred to an SPI-master device.

```
DLN_RESULT DlnSpiSlaveLoadReply(
    HDLN handle,
    uint8_t port,
    uint16_t size,
uint8_t* buffer
);
```

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI-slave port to be used.

size
A size of the message buffer. This parameter is specified in bytes. The maximum value is 256 bytes.

buffer
A pointer to an array of unsigned 8-bit integers.256-element array. Each of the elements is an 8-bit value. The buffer must contain the information to be sent to a master.

5.5.3.10. DlnSpiSlaveSetEvent()

The `DlnSpiSlaveSetEvent()` function configures the event generation conditions for the SPI slave port of the DLN-series adapter.

```
DLN_RESULT DlnSpiSlaveSetEvent(
   HDLN handle,
   uint8_t port,
   uint8_t eventType,
   uint16_t bufferSize
);
```

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI-slave port to be configured.

eventType
Must contain the condition for event generation on the SPI-slave port. The following values are available:

- `0` or `SPI_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `SPI_SLAVE_EVENT_SS_RISE` - events are generated when an SS line is released;
- `2` or `SPI_SLAVE_EVENT_SS_BUFFER_FULL` - an event is generated when the allocated buffer becomes full.

bufferSize
The size of the buffer that stores data received from a master. This parameter is set in bytes. The maximum value is 256 bytes.

5.5.3.11. DlnSpiSlaveGetEvent()

The `DlnSpiSlaveGetEvent()` function retrieves currently set event generation conditions for the specified SPI-slave port.

```
DLN_RESULT DlnSpiSlaveGetEvent(
   HDLN handle,
   uint8_t port,
   uint8_t* eventType,
   uint16_t* bufferSize
);
```

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the SPI slave port to be used.

eventType
A pointer to an unsigned 8-bit integer. The integer will be filled with current event generation conditions for the specified SPI slave port.

bufferSize
A pointer to an unsigned 16-bit integer. The integer will be filled with currently buffer size.

5.5.4. Commands and Responses

5.5.4.1. DLN_SPI_SLAVE_GET_PORT_COUNT

DLN_SPI_SLAVE_GET_PORT_COUNT Command

Go to response

The **DLN_SPI_SLAVE_GET_PORT_COUNT** command is used to retrieve the number of SPI slave ports available in your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_SPI_SLAVE_GET_PORT_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_GET_PORT_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_PORT_COUNT command it must be set to `0x0B00`. You can use the `DLN_MSG_ID_SPI_SLAVE_GET_PORT_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_SPI_SLAVE_GET_PORT_COUNT Response

Go to Command

The adapter sends the **DLN_SPI_SLAVE_GET_PORT_COUNT** response after the command execution. The response contains the available number of SPI slave ports.

```
typedef struct
{
   DLN_MSG_HEADER header;
```

```
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_SPI_SLAVE_GET_PORT_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_GET_PORT_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_PORT_COUNT response it is set to `0x0B00`. The `DLN_MSG_ID_SPI_SLAVE_GET_PORT_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of the SPI slave ports has been successfully obtained.

count
Contains the available number of SPI slave ports.

5.5.4.2. DLN_SPI_SLAVE_ENABLE

DLN_SPI_SLAVE_ENABLE Command

Go to response

The **DLN_SPI_SLAVE_ENABLE** command is used to activate the corresponding SPI slave port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_SPI_SLAVE_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_ENABLE command it must be set to `0x0B01`. You can use the `DLN_MSG_ID_SPI_SLAVE_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to be activated.

DLN_SPI_SLAVE_ENABLE Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_ENABLE** response after the command execution. The `result` field informs a user if the specified SPI slave port has been successfully activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t conflict;
}__PACKED_ATTR DLN_SPI_SLAVE_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_ENABLE response it is set to `0x0B01`. The `DLN_MSG_ID_SPI_SLAVE_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified SPI master port has been successfully activated.

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used for the SPI slave module. To fix this a user has to disconnect a pin from a module that it has been assigned to and send the DLN_SPI_SLAVE_ENABLE command once again. In case there still are conflicted pins, only the number of the next one will be returned.

5.5.4.3. DLN_SPI_SLAVE_DISABLE

DLN_SPI_SLAVE_DISABLE Command

Go to response

The **DLN_SPI_SLAVE_DISABLE** command is used to deactivate the specified SPI slave port of your DLN-series adapter.

```
typedef struct
```

```
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t waitForTransferCompletion;
}__PACKED_ATTR DLN_SPI_SLAVE_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_DISABLE command it must be set to `0x0B02`. You can use the `DLN_MSG_ID_SPI_SLAVE_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to be deactivated.

waitForTransferCompletion
Used to choose whether the DLN-series adapter should wait for current data transfers to complete before deactivating the SPI slave port. The following values are available:

- `1` or `DLN_SPI_SLAVE_WAIT_FOR_TRANSFERS` - wait until transfers are completed;
- `0` or `DLN_SPI_SLAVE_CANCEL_TRANSFERS` - cancel data transfers.

DLN_SPI_SLAVE_DISABLE Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_DISABLE** response after the function execution. The `result` field informs a user if the specified SPI slave port has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_SPI_SLAVE_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_DISABLE response it is set to `0x0B02`. The `DLN_MSG_ID_SPI_SLAVE_DISABLE` constant can be used.

- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified SPI slave port has been successfully deactivated.

5.5.4.4. DLN_SPI_SLAVE_IS_ENABLED

DLN_SPI_SLAVE_IS_ENABLED Command

Go to response

The **DLN_SPI_SLAVE_IS_ENABLED** command is used to retrieve information, whether the specified SPI slave port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_SPI_SLAVE_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_IS_ENABLED command it must be set to `0x0B03`. You can use the `DLN_MSG_ID_SPI_SLAVE_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to retrieve the information from.

DLN_SPI_SLAVE_IS_ENABLED Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_IS_ENABLED** response after the command execution. The response informs a user if the specified SPI slave port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_SPI_SLAVE_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_MASTER_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_IS_ENABLED response it is set to `0x0B03`. The `DLN_MSG_ID_SPI_SLAVE_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the SPI slave port has been successfully retrieved.

enabled
Informs whether the specified SPI slave port is activated. There are two possible values:

- `0` or `DLN_SPI_SLAVE_DISABLED` - the SPI slave port is activated.
- `1` or `DLN_SPI_SLAVE_ENABLED` - the SPI slave port is deactivated.

5.5.4.5. DLN_SPI_SLAVE_SET_MODE

DLN_SPI_SLAVE_SET_MODE Command

Go to response

The **DLN_SPI_SLAVE_SET_MODE** command is used to set SPI transmission parameters (CPOL and CPHA).

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t mode;
}__PACKED_ATTR DLN_SPI_SLAVE_SET_MODE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_SET_MODE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_SET_MODE command it must be set to `0x0B04`. You can use the `DLN_MSG_ID_SPI_SLAVE_SET_MODE` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to apply configuration to.

mode
A bit field consisting of 8 bits. The bits `0` and `1` correspond to CPOL and CPHA parameters respectively and define the SPI slave mode. The rest of the bits is not used. You can also use special constants, defined in the `dln_spi_slave.h` file for each of the bits. See Transfer Modes for additional info.

Bit	Value	Description	Constant
0	0	CPOL=0	DLN_SPI_SLAVE_CPOL_0
0	1	CPOL=1	DLN_SPI_SLAVE_CPOL_1
1	0	CPHA=0	DLN_SPI_SLAVE_CPHA_0
1	1	CPHA=1	DLN_SPI_SLAVE_CPHA_1

DLN_SPI_SLAVE_SET_MODE Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_SET_MODE** response after the command execution. The `result` field informs a user whether the configuration has been successfully changed.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
}__PACKED_ATTR DLN_SPI_SLAVE_SET_MODE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_SET_MODE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_SET_MODE response it is set to `0x0B04`. The `DLN_MSG_ID_SPI_SLAVE_SET_MODE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The configuration has been successfully set.

5.5.4.6. DLN_SPI_SLAVE_GET_MODE

DLN_SPI_SLAVE_GET_MODE Command

Go to response

The **DLN_SPI_SLAVE_GET_MODE** command is used to retrieve current configuration of an SPI slave port of the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_SPI_SLAVE_GET_MODE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_GET_MODE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_MODE command it must be set to `0x0B05`. You can use the `DLN_MSG_ID_SPI_SLAVE_GET_MODE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to retrieve the information from.

DLN_SPI_MASTER_GET_MODE Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_GET_MODE** response after the command execution. The response contains current mode of the specified SPI slave port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t mode;
}__PACKED_ATTR DLN_SPI_SLAVE_GET_MODE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_GET_MODE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_MODE response it is set to `0x0B05`. The `DLN_MSG_ID_SPI_SLAVE_GET_MODE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.

- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The configuration has been successfully retrieved.

mode
A bit field, consisting of 8 bits and used to describe SPI slave transmission mode. See DLN_SPI_SLAVE_SET_MODE Command for details.

5.5.4.7. DLN_SPI_SLAVE_SET_FRAME_SIZE

DLN_SPI_SLAVE_SET_FRAME_SIZE Command

Go to response

The **DLN_SPI_SLAVE_SET_FRAME_SIZE** command is used to set the size of a single portion of data transferred via SPI.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t frameSize;
} __PACKED_ATTR DLN_SPI_SLAVE_SET_FRAME_SIZE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_SET_FRAME_SIZE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_SET_TRANSFER_SIZE command it must be set to `0x0B06`. You can use the `DLN_MSG_ID_SPI_SLAVE_SET_FRAME_SIZE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to be configured.

frameSize
A number of bytes to be transferred, specified in bits. The DLN-series adapter supports 8 to 16 bits per frame.

DLN_SPI_SLAVE_SET_FRAME_SIZE Response

Go to command

The device sends the **DLN_SPI_SLAVE_SET_FRAME_SIZE** response after the command execution. The response informs a user if the frame size has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SPI_MASTER_SET_FRAME_SIZE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_SET_FRAME_SIZE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_MASTER_SET_FRAME_SIZE response it is set to `0x0B06`. The `DLN_MSG_ID_SPI_SLAVE_SET_FRAME_SIZE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The number of bytes per frame has been successfully set.

5.5.4.8. DLN_SPI_SLAVE_GET_FRAME_SIZE

DLN_SPI_GET_FRAME_SIZE Command

Go to response

The **DLN_SPI_SLAVE_GET_FRAME_SIZE** command is used to retrieve current size setting for a single portion of data transferred via SPI.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_SLAVE_GET_FRAME_SIZE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_GET_FRAME_SIZE_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_FRAME_SIZE command it must be set to `0x0B07`. You can use the `DLN_MSG_ID_SPI_SLAVE_GET_FRAME_SIZE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to get the information from.

DLN_SPI_SLAVE_GET_FRAME_SIZE Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_GET_FRAME_SIZE** response after the command execution. The response contains current setting for the number of bits per transfer.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t frameSize;
} __PACKED_ATTR DLN_SPI_SLAVE_GET_FRAME_SIZE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_SET_FRAME_SIZE_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_FRAME_SIZE response it is set to `0x0B07`. The `DLN_MSG_ID_SPI_SLAVE_GET_FRAME_SIZE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The number of bytes per frame has been successfully retrieved.

frameSize
A number of bits to be transferred. The DLN-series adapter supports 8 to 16 bytes per frame.

5.5.4.9. DLN_SPI_SLAVE_LOAD_REPLY

DLN_SPI_SLAVE_LOAD_REPLY Command

Go to response

The **DLN_SPI_SLAVE_LOAD_REPLY** Command is used to load information to be transferred to an SPI-master device.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint16_t size;
```

```
    uint8_t buffer[DLN_SPI_SLAVE_BUFFER_SIZE];
} __PACKED_ATTR DLN_SPI_SLAVE_LOAD_REPLY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_LOAD_REPLY_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_LOAD_REPLY command it must be set to `0x0B08`. You can use the `DLN_MSG_ID_SPI_SLAVE_LOAD_REPLY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI slave port to be used.

size
A size of the message buffer. This parameter is specified in bytes. The maximum value is 256 bytes.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer must contain the information to be sent to a master.

DLN_SPI_SLAVE_LOAD_REPLY Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_LOAD_REPLY** response after the command execution. The response informs a user if the data to be sent to an SPI-master were successfully loaded.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SPI_SLAVE_LOAD_REPLY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_LOAD_REPLY_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_LOAD_REPLY response it is set to `0x0B08`. The `DLN_MSG_ID_SPI_SLAVE_LOAD_REPLY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.

- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The information has been successfully loaded.

5.5.4.10. DLN_SPI_SLAVE_SET_EVENT

DLN_SPI_SLAVE_SET_EVENT Command

Go to response

The **DLN_SPI_SLAVE_SET** command is used to configure the event generation conditions for the SPI-slave module of the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t eventType;
   uint16_t bufferSize;
} __PACKED_ATTR DLN_SPI_SLAVE_SET_EVENT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_SET_EVENT_CMD structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_SET_EVENT command it must be set to `0x0B09`. You can use the `DLN_MSG_ID_SPI_SLAVE_SET_EVENT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI-slave port to be configured.

eventType
Must contain the condition for event generation on the SPI-slave port. The following values are available:

- `0` or `SPI_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `SPI_SLAVE_EVENT_SS_RISE` - events are generated when an SS line is released;
- `2` or `SPI_SLAVE_EVENT_SS_BUFFER_FULL` - an event is generated when the allocated buffer becomes full.

bufferSize
The size of the buffer that stores data received from a master. This parameter is set in bytes. The maximum value is 256 bytes.

DLN_SPI_SLAVE_SET_EVENT Response

The adapter sends the **DLN_SPI_SLAVE_SET_EVENT** Response after the command execution. The `result` field informs a user if the SPI-slave event generation conditions have been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_SPI_SLAVE_SET_EVENT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_SET_EVENT_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_SET_EVENT response it is set to `0x0B09`. The `DLN_MSG_ID_SPI_SLAVE_SET_EVENT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The event generation conditions have been successfully set.

5.5.4.11. DLN_SPI_SLAVE_GET_EVENT

DLN_SPI_SLAVE_GET_EVENT Command

Go to response

The **DLN_SPI_SLAVE_GET_EVENT** command is used to retrieve currently set event generation conditions for the specified SPI-slave port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_SPI_SLAVE_GET_EVENT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_SPI_SLAVE_GET_EVENT_CMD structure.

- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_EVENT command it must be set to `0x0B0A`. You can use the `DLN_MSG_ID_SPI_SLAVE_GET_EVENT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the SPI-slave port to retrieve the information from.

DLN_SPI_SLAVE_GET_EVENT Response

Go to command

The adapter sends the **DLN_SPI_SLAVE_GET_EVENT** response after the command execution. The response contains current event generation conditions for the specified SPI-slave port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t eventType;
   uint16_t bufferSize;
} __PACKED_ATTR DLN_SPI_SLAVE_GET_EVENT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_SPI_SLAVE_GET_EVENT_RSP structure.
- **msgId** - Defines the message. For the DLN_SPI_SLAVE_GET_EVENT response it is set to `0x0B0A`. The `DLN_MSG_ID_SPI_SLAVE_GET_EVENT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The event generation conditions have been successfully retrieved.

eventType
Contains the condition for event generation on the SPI-slave port. The following values are available:

- `0` or `SPI_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `SPI_SLAVE_EVENT_SS_RISE` - events are generated when an SS line is released;
- `2` or `SPI_SLAVE_EVENT_SS_BUFFER_FULL` - an event is generated when the allocated buffer becomes full.

bufferSize
The size of the buffer that stores data received from a master. This parameter is set in bytes. The maximum value is 256 bytes.

Chapter 6. I²C Interface

6.1. Overview

6.2. I²C Master Module

6.2.1. Functions

6.2.1.1. DlnI2cMasterGetPortCount()

```
DLN_RESULT DlnI2cMasterGetPortCount(
   HDLN handle,
   uint8_t* count
);
```

The `DlnI2cMasterGetPortCount()` function retrieves the total number of I²C master ports available in your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. This integer will be filled with the number of available I²C master ports after the function execution.

This function is defined in the `dln_i2c_master.h` file.

6.2.1.2. DlnI2cMasterEnable()

```
DLN_RESULT DlnI2cMasterEnable(
   HDLN handle,
   uint8_t port,
   uint16_t* conflict
);
```

The `DlnI2cMasterEnable()` function activates corresponding I²C master port on your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port to be enabled .

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used by the I²C master module. To fix this a user has to disconnect a pin from a module that it

has been assigned to and call the DlnI2cMasterEnable() function once again. If there are any more conflicting pins, the next conflicted pin number will be returned.

This function is defined in the `dln_i2c_master.h` file.

6.2.1.3. DlnI2cMasterDisable()

```
DLN_RESULT DlnI2cMasterDisable(
   HDLN handle,
   uint8_t port,
   uint8_t waitForTransferCompletion
);
```

The `DlnI2cMasterDisable()` function deactivates corresponding I²C master port on your DLN-Series adapter.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port to be disabled.

waitForTransferCompletion
Used to choose whether the device should wait for current data transfers to complete before disabling an I²C master. The following values are available:

- `1` or `DLN_I2C_MASTER_WAIT_FOR_TRANSFERS` - wait until transfers are completed;
- `0` or `DLN_I2C_MASTER_CANCEL_TRANSFERS` - cancel all pending data transfers and deactivate the module.

This function is defined in the `dln_i2c_master.h` file.

6.2.1.4. DlnI2cMasterIsEnabled()

```
DLN_RESULT DlnI2cMasterIsEnabled(
   HDLN handle,
   uint8_t port,
   uint8_t* enabled
);
```

The `DlnI2cMasterIsEnabled()` function retrieves information whether the specified I²C master port is activated.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with information whether the specified I²C master port is activated after the function execution. There are two possible values:

- `0` or `DLN_I2C_MASTER_DISABLED` - the port is not configured as I²C master.
- `1` or `DLN_I2C_MASTER_ENABLED` - the port is configured as I²C master.

This function is defined in the `dln_i2c_master.h` file.

6.2.1.5. DlnI2cMasterSetFrequency()

```
DLN_RESULT DlnI2cMasterSetFrequency(
   HDLN handle,
   uint8_t port,
   uint32_t frequency,
   uint32_t* actualFrequency
);
```

The `DlnI2cMasterSetFrequency()` function sets the clock frequency for the I²C bus.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port to be configured.

frequency
I²C frequency value, specified in Hz. A user may specify any value within the range, supported by the DLN-series adapter. This range can be retrieved using the respective function. In case a user enters an incompatible value, it will be approximated as the closest lower frequency value, supported by the adapter.

actualFrequency
A pointer to an unsigned 32-bit integer. This integer will be filled with the actual frequency applied by this function. The frequency is specified in Hz. If the frequency specified in `frequency` parameter is supported, the actual frequency will equal to it. Otherwise the closest lower value will be applied. If `NULL` is specified in this parameter, the actual frequency value will not be returned. You can still use the DlnI2cMasterGetFrequency() function to check the actual frequency.

This function is defined in the `dln_i2c_master.h` file.

6.2.1.6. DlnI2cMasterGetFrequency()

```
DLN_RESULT DlnI2cMasterGetFrequency(
   HDLN handle,
   uint8_t port,
   uint32_t* frequency
);
```

The `DlnI2cMasterGetFrequency()` function retrieves current I²C bus clock frequency.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port to retrieve the information from.

frequency
A pointer to an unsigned 32-bit integer. This integer will be filled with current I²C bus clock frequency in Hz after the function execution.

This function is defined in the `dln_i2c_master.h` file.

6.2.1.7. DlnI2cMasterWrite()

```
DLN_RESULT DlnI2cMasterWrite(
   HDLN handle,
   uint8_t port,
   uint8_t slaveDeviceAddress,
   uint8_t memoryAddressLength,
   uint32_t memoryAddress,
   uint16_t bufferLength,
   uint8_t* buffer
);
```

The `DlnI2cMasterWrite()` function sends data via I²C interface.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port to be used.

slaveDeviceAddress
A 7-bit number, uniquely assigned to each I²C slave device.

memoryAddressLength
Must contain an internal address length. If set to zero, no internal address is sent.

memoryAddress
An internal I²C slave device address.

bufferLength
The size of the message buffer in bytes. This value must fall within a range from 1 to 256.

buffer
A pointer to an array of unsigned 8-bit integers. The buffer must contain the information to be sent to a slave during the function execution. The buffer size must not exceed 256 bytes.

6.2.1.8. DlnI2cMasterRead()

```
DLN_RESULT DlnI2cMasterRead(
   HDLN handle,
   uint8_t port,
   uint8_t slaveDeviceAddress,
   uint8_t memoryAddressLength,
   uint32_t memoryAddress,
   uint16_t bufferLength,
   uint8_t* buffer
);
```

The `DlnI2cMasterRead()` function reads data from I²C slave device.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port to be used.

slaveDeviceAddress
A 7-bit number, assigned to each I²C slave device.

memoryAddressLength
An internal address length. If set to zero, no internal address is sent.

memoryAddress
An internal I²C slave device address.

bufferLength
The size of the message buffer (in the range from 1 to 256 bytes).

buffer
A pointer to an array of unsigned 8-bit integers. The buffer will be filled with the data received from the I²C slave device during the function execution. The array must contain at least `bufferLength` elements.

6.2.1.9. DlnI2cMasterScanDevices()

```
DLN_RESULT DlnI2cMasterScanDevices(
   HDLN handle,
   uint8_t port,
   uint8_t* addressCount,
   uint8_t* addressList
);
```

The `DlnI2cMasterScanDevices()` function scans all the 127 slave addresses searching for connected I²C slave devices.

handle
A handle to the DLN-series adapter.

port
A number of the I²C master port scan for I2C slave devices.

addressCount
A pointer to an unsigned 8-bit integer. The integer will be filled with the number of found I²C slave devices after the function execution.

addressList
A pointer to an array of unsigned 8-bit integers. The array will contain the addresses of found I²C slave devices after the function execution.

6.2.2. Commands and Responses

6.2.2.1. DLN_I2C_MASTER_GET_PORT_COUNT

DLN_I2C_MASTER_GET_PORT_COUNT Command

Go to response

The **DLN_I2C_MASTER_GET_PORT_COUNT** command is used to retrieve the number of I2C master ports available in your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_I2C_MASTER_GET_PORT_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_GET_PORT_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_GET_PORT_COUNT command it must be set to `0x0300`. You can use the `DLN_MSG_ID_I2C_MASTER_GET_PORT_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_I2C_MASTER_GET_PORT_COUNT Response

Go to command

The adapter sends the **DLN_I2C_MASTER_GET_PORT_COUNT** response after the command execution. The response contains the available number of I2C master ports.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_I2C_MASTER_GET_PORT_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_GET_PORT_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_GET_PORT_COUNT response it is set to `0x0300`. The `DLN_MSG_ID_I2C_MASTER_GET_PORT_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of I²C master ports has been successfully obtained.

count
Contains the available number of I²C master ports.

6.2.2.2. DLN_I2C_MASTER_ENABLE

DLN_I2C_MASTER_ENABLE Command

Go to response

The **DLN_I2C_MASTER_ENABLE** command is used to activate the corresponding I²C master port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_MASTER_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_ENABLE command it must be set to `0x0301`. You can use the `DLN_MSG_ID_I2C_MASTER_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C master port to retrieve the information from.

DLN_I2C_MASTER_ENABLE Response

Go to command

The adapter sends the **DLN_I2C_MASTER_ENABLE** response after the command execution. The `result` field informs a user if the specified I²C master port has been successfully activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t conflict;
} __PACKED_ATTR DLN_I2C_MASTER_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_ENABLE response it is set to `0x0301`. The `DLN_MSG_ID_I2C_MASTER_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified I2C master port has been successfully activated.

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used by the I2C module. To fix this a user has to disconnect a pin from a module that it has been assigned to and send the DLN_I2C_MASTER_ENABLE command once again. In case there still are conflicted pins, only the number of the next one will be returned.

6.2.2.3. DLN_I2C_MASTER_DISABLE

DLN_I2C_MASTER_DISABLE Command

Go to response

The **DLN_I2C_MASTER_DISABLE** command is used to deactivate the specified I2C master port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_MASTER_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_DISABLE command it must be set to `0x0302`. You can use the `DLN_MSG_ID_I2C_MASTER_DISABLE` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C master port to be deactivated.

DLN_I2C_MASTER_DISABLE Response

Go to command

The adapter sends the **DLN_I2C_MASTER_DISABLE** response after the function execution. The result field informs a user if the specified I²C master port has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_MASTER_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_DISABLE response it is set to `0x0302`. The `DLN_MSG_ID_I2C_MASTER_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified I²C master port has been successfully deactivated.

6.2.2.4. DLN_I2C_MASTER_IS_ENABLED

DLN_I2C_MASTER_IS_ENABLED Command

Go to response

The **DLN_I2C_MASTER_IS_ENABLED** command is used to retrieve information, whether the specified I²C master port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_I2C_MASTER_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_IS_ENABLED command it must be set to `0x0303`. You can use the `DLN_MSG_ID_I2C_MASTER_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C master port to retrieve the information from.

DLN_I2C_MASTER_IS_ENABLED Response

Go to command

The adapter sends the **DLN_I2C_MASTER_IS_ENABLED** response after the command execution. The response informs a user if the specified I²C master port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_I2C_MASTER_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_IS_ENABLED response it is set to `0x0303`. The `DLN_MSG_ID_I2C_MASTER_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the I²C master port has been successfully retrieved.

enabled
Informs whether the specified I²C master port is activated. There are two possible values:

- `0` or `DLN_I2C_MASTER_DISABLED` - the I2C master port is activated.
- `1` or `DLN_I2C_MASTER_ENABLED` - the I2C master port is deactivated.

6.2.2.5. DLN_I2C_MASTER_SET_FREQUENCY

DLN_I2C_MASTER_SET_FREQUENCY Command

Go to response

The **DLN_I2C_MASTER_SET_FREQUENCY** command is used to set the clock frequency to synchronize data transfer between master and slave devices.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint32_t frequency;
} __PACKED_ATTR DLN_I2C_MASTER_SET_FREQUENCY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_SET_FREQUENCY_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_SET_FREQUENCY command it must be set to `0x0304`. You can use the `DLN_MSG_ID_I2C_MASTER_SET_FREQUENCY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C master port to be configured.

frequency
The frequency value, specified in Hz. A user may specify any value within the range, supported by the DLN-series adapter. This range can be retrieved using the respective function. In case a user enters an incompatible value, it will be approximated as the closest lower frequency value, supported by the adapter.

DLN_I2C_MASTER_SET_FREQUENCY Response

Go to command

The adapter sends the **DLN_I2C_MASTER_SET FREQUENCY** response after the command execution. The response informs a user if the I2C clock frequency has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t frequency;
```

```
} __PACKED_ATTR DLN_I2C_MASTER_SET_FREQUENCY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_SET_FREQUENCY_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_SET_FREQUENCY response it is set to `0x0304`. The `DLN_MSG_ID_I2C_MASTER_SET_FREQUENCY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The clock frequency has been successfully set.

frequency
Contains the actual frequency applied by this command. If the frequency specified in `frequency` parameter is supported, the actual frequency will equal to it. Otherwise the closest lower value will be applied. If `NULL` is specified in this parameter, the actual frequency value will not be returned. You can still use the DLN_I2C_MASTER_GET_FREQUENCY command to check the actual frequency.

6.2.2.6. DLN_I2C_MASTER_GET_FREQUENCY

DLN_I2C_MASTER_GET_FREQUENCY Command

Go to response

The **DLN_I2C_GET_FREQUENCY** command is used to retrieve current I2C bus clock frequency.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_MASTER_GET_FREQUENCY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_GET_FREQUENCY_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_GET_FREQUENCY command it must be set to `0x0305`. You can use the `DLN_MSG_ID_I2C_MASTER_GET_FREQUENCY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).

- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C master port to retrieve information from.

DLN_I2C_MASTER_GET_FREQUENCY Response

Go to command

The adapter sends the **DLN_I2C_MASTER_GET_FREQUENCY** response after the command execution. The response contains current I2C clock frequency setting.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint32_t frequency;
} __PACKED_ATTR DLN_I2C_MASTER_GET_FREQUENCY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_GET_FREQUENCY_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_GET_FREQUENCY response it is set to `0x0305`. The `DLN_MSG_ID_I2C_MASTER_GET_FREQUENCY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - Current I2C clock frequency has been successfully retrieved.

frequency
Contains current I2C bus clock frequency in Hz.

6.2.2.7. DLN_I2C_MASTER_WRITE

DLN_I2C_MASTER_WRITE Command

Go to response

The **DLN_I2C_MASTER_WRITE** command is used to send the data via I2C interface.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t slaveDeviceAddress;
   uint8_t memoryAddressLength;
   uint32_t memoryAddress;
```

```
   uint16_t bufferLength;
   uint8_t buffer[256];
} __PACKED_ATTR DLN_I2C_MASTER_WRITE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_WRITE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_WRITE command it must be set to `0x0306`. You can use the `DLN_MSG_ID_I2C_MASTER_WRITE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C master port to be used.

slaveDeviceAddress
A 7-bit number, uniquely assigned to each I2C slave device.

memoryAddressLength
An internal address length. If set to zero, no internal address is sent.

memoryAddress
An internal I2C slave device address.

bufferLength
The size of the message buffer in bytes. This value must fall witin the range from 1 to 256.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer contains the information to be sent to a slave. The buffer size must not exceed 256 bytes.

DLN_I2C_MASTER_WRITE Response

Go to command

The adapter sends the **DLN_I2C_MASTER_WRITE** response after the command execution. The `result` fields informs a user if the data were successfully written.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_MASTER_WRITE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_WRITE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_WRITE response it is set to `0x0306`. The `DLN_MSG_ID_I2C_MASTER_WRITE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - An I²C data writing completed successfully.

6.2.2.8. DLN_I2C_MASTER_READ

DLN_I2C_MASTER_READ Command

Go to response

The **DLN_I2C_MASTER_READ** command is used to read data from I²C slave device.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t slaveDeviceAddress;
   uint8_t memoryAddressLength;
   uint32_t memoryAddress;
   uint16_t bufferLength;
} __PACKED_ATTR DLN_I2C_MASTER_READ_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_READ_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_READ command it must be set to `0x0307`. You can use the `DLN_MSG_ID_I2C_MASTER_READ` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C master port to be used.

slaveDeviceAddress
A 7-bit number, uniquely assigned to each I²C slave device.

memoryAddressLength
An the internal address length. If set to zero, no internal address is sent.

memoryAddress
An the internal I2C slave device address.

bufferLength
The size of the message buffer (in the range from 1 to 256 bytes).

DLN_I2C_MASTER_READ Response

Go to command

The adapter sends the **DLN_I2C_MASTER_READ** response after the command execution. The response contains confirmation of data transfer as well as data received from a slave.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t bufferLength;
   uint8_t buffer[256];
} __PACKED_ATTR DLN_I2C_MASTER_READ_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_READ_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_READ response it is set to `0x0307`. The `DLN_MSG_ID_I2C_MASTER_READ` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - An I2C data reading completed successfully.

bufferLength
The size of the message buffer.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer will be filled with the data received from the I2C slave device. The array must contain at least `bufferLength` elements.

6.2.2.9. DLN_I2C_MASTER_SCAN_DEVICES

DLN_I2C_MASTER_SCAN_DEVICES Command

Go to response

The program is used to scan all the 127 slave addresses searching for connected I2C slave devices.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_MASTER_SCAN_DEVICES_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_SCAN_DEVICES_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_SCAN_DEVICES command it must be set to `0x0308`. You can use the `DLN_MSG_ID_I2C_MASTER_SCAN_DEVICES` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C master port to scan for I²C slave devices.

DLN_I2C_MASTER_SCAN_DEVICES Response

Go to command

The adapter sends the **DLN_I2C_MASTER_SCAN_DEVICES** response after the command execution. The response contains the number of found I²C slave devices and lists their slave addresses.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t addressCount;
   uint8_t addressList[128];
} __PACKED_ATTR DLN_I2C_MASTER_SCAN_DEVICES_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_SCAN_DEVICES_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_SCAN_DEVICES response it is set to `0x0308`. The `DLN_MSG_ID_I2C_MASTER_SCAN_DEVICES` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - I2C slave device search completed successfully.

addressCount
A number of found slave devices.

addressList
A 128-element array. Each of the elements is an 8-bit value. It will contain the addresses of found I2C slave devices.

6.2.2.10. DLN_I2C_MASTER_PULLUP_ENABLE

DLN_I2C_MASTER_PULLUP_ENABLE Command

Go to response

The **DLN_I2C_MASTER_PULLUP_ENABLE** command is used to enable embedded pull-up resistor for the specified I2C master port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_MASTER_PULLUP_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_GPIO_I2C_MASTER_PULLUP_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_PULLUP_ENABLE_CMD command, it must be set to `0x0309`. You can use the `DLN_MSG_ID_I2C_MASTER_PULLUP_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C master port to be configured.

DLN_I2C_MASTER_PULLUP_ENABLE Response

Go to command

The adapter sends the **DLN_I2C_MASTER_PULLUP_ENABLE** response after the command execution. The `result` field informs a user if a pull-up resistor has been successfully activated for the specified I2C master port.

```
typedef struct
{
```

```
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_MASTER_PULLUP_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_PULLUP_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_GPIO_I2C_MASTER_PULLUP_ENABLE_RSP response it is set to `0x0309`. The `DLN_MSG_ID_I2C_MASTER_PULLUP_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the embedded pull-up resistor has been successfully activated for the specified I2C master port.

6.2.2.11. DLN_I2C_MASTER_PULLUP_DISABLE

DLN_I2C_MASTER_PULLUP_DISABLE Command

Go to response

The **DLN_I2C_MASTER_PULLUP_DISABLE** command is used to disable the embedded pull-uo resistor for the specified I2C master port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_MASTER_PULLUP_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_PULLUP_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_PULLUP_DISABLE_CMD command, it must be set to `0x030A`. You can use the `DLN_MSG_ID_I2C_MASTER_PULLUP_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).

- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C master port to be configured.

DLN_I2C_MASTER_PULLUP_DISABLE Response

Go to command

The adapter sends the **DLN_I2C_MASTER_PULLUP_DISABLE** response after the command execution. The `result` field informs a user if the embedded pull-up resistor has been successfully disabled for the specified I2C master port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_MASTER_PULLUP_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_PULLUP_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_PULLUP_DISABLE_RSP response it is set to `0x030A`. The `DLN_MSG_ID_GPIO_PIN_PULLUP_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the embedded pull-up resistor has been successfully deactivated for the specified I2C master port.

6.2.2.12. DLN_I2C_MASTER_PULLUP_IS_ENABLED

DLN_I2C_MASTER_PULLUP_IS_ENABLED Command

Go to response

The **DLN_I2C_MASTER_PULLUP_IS_ENABLED** command is used to retrieve information on whether the embedded pull-up resistor is currently activated for the specified I2C master port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_MASTER_PULLUP_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and it is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_MASTER_PULLUP_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_PULLUP_IS_ENABLED_CMD command, it must be set to `0x030B`. You can use the `DLN_MSG_ID_I2C_MASTER_PULLUP_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65535).
- **handle** - A handle to the DLN-series adapter.

port
An I²C master port to retrieve the information from.

DLN_I2C_MASTER_PULLUP_IS_ENABLED Response

Go to command

The adapter sends the **DLN_I2C_MASTER_PULLUP_IS_ENABLED** response after the command execution. The response informs a user whether the embedded pull-up resistor is currently enabled for the specified I²C master port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
} __PACKED_ATTR DLN_I2C_MASTER_PULLUP_IS_ENABLED_RSP;
```

Parameters

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_MASTER_PULLUP_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_MASTER_PULLUP_IS_ENABLED_RSP response it is set to `0x030B`. The `DLN_MSG_ID_I2C_MASTER_PULLUP_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the current setting for the embedded pull-up resistor for the specified I²C master porrt has been successfully retrieved.

port
Contains a number of the I²C master port.

enabled
Contains the current configuration of the I²C master port. The following values are available:

- `1` - the embedded pull-up resistor is currently enabled for the specified I²C master port;
- `0` - the embedded pull-up resistor is currently disabled for the specified I²C master port.

6.3. I²C Slave Module

6.3.1. I²C Slave Events

A DLN-series adapter can be configured to send I²C slave events. They are generated when certain predefined conditions are met. The I²C slave events are configured using the DlnI2cSlaveSetEvent() function. The `eventType` parameter defines conditions for event generation. You can retrieve current event generation conditions by calling the DlnI2cSlaveGetEvent() function. There are three types of events:

- `DLN_I2C_SLAVE_EVENT_NONE` - no events are generated.
- `DLN_I2C_SLAVE_EVENT_READ` - events are generated when master device starts reading data sent by a slave via I²C bus.
- `DLN_I2C_SLAVE_EVENT_WRITE` - events are generated when master device starts sending data to a slave via I²C bus.
- `DLN_I2C_SLAVE_EVENT_READ_WRITE` - events are generated when the I²C master device starts sending data to a slave or reading data sent by the slave via I²C bus.

6.3.1.1. DLN_I2C_SLAVE_EVENT_NONE

No events are generated.

In order to choose this event type, call the DlnI2cSlaveSetEvent() and set `0` or `SPI_SLAVE_EVENT_NONE` as the `eventType` parameter.

6.3.1.2. DLN_I2C_SLAVE_EVENT_READ

Events are generated when a master device starts reading data sent by a slave via I²C bus. This event is described using the DLN_I2C_SLAVE_READ_EV structure.

The information sent to your application includes event counter, event type, I²C slave port number, slave address and the size of the message buffer.

In order to choose this event type, call the DlnI2cSlaveSetEvent() function and set `1` or `SPI_SLAVE_EVENT_READ` as the `eventType` parameter.

6.3.1.3. DLN_I2C_SLAVE_EVENT_WRITE

Events are generated when a master device starts sending data to a slave via I²C bus. This event is described using the DLN_I2C_SLAVE_WRITE_EV structure.

Unlike the DLN_I2C_SLAVE_EVENT_READ event, this event also contains actual information received from master via I²C bus.

In order to choose this event type, call the DlnI2cSlaveSetEvent() function and set `2` or `SPI_SLAVE_EVENT_WRITE` as the `eventType` parameter.

6.3.1.4. DLN_I2C_SLAVE_EVENT_READ_WRITE

Events are generated in both cases - when a master device starts reading or sending data via I²C interface.

Depending on actual case, either DLN_I2C_SLAVE_READ_EV or DLN_I2C_SLAVE_WRITE_EV structure will be used to store the event description.

In order to choose this event type, call the DlnI2cSlaveSetEvent() function and set 3 or SPI_SLAVE_EVENT_READ_WRITE as the eventType parameter.

6.3.2. Structures

6.3.2.1. DLN_I2C_SLAVE_READ_EV

This structure is used to store the description of DLN_I2C_SLAVE_READ_EV events.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t eventCount;
   uint8_t eventType;
   uint8_t port;
   uint8_t slaveAddress;
   uint16_t size;
} __PACKED_ATTR DLN_I2C_SLAVE_READ_EV;
```

Members

header
Defines the DLN message header DLN_MSG_HEADER. This structure's header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_i2C_SLAVE_READ_EV structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_READ_EV response it is set to 0x0C10. The DLN_MSG_ID_I2C_SLAVE_READ_EV constant can be used.
- **echoCounter**
- **handle** - A handle to the DLN-series adapter.

eventCount
Contains the number of generated events after configuration setting.

eventType
Contains the condition for event generation on the I²C-slave port. The following values are available:

- 0 or DLN_I2C_SLAVE_EVENT_NONE - no events are generated for the current port.
- 1 or DLN_I2C_SLAVE_EVENT_READ - events are generated when master device starts reading data sent by a slave via I²C bus.
- 2 or DLN_I2C_SLAVE_EVENT_WRITE - events are generated when master device starts sending data to a slave via I²C bus.
- 3 or DLN_I2C_SLAVE_EVENT_READ_WRITE - events are generated when the I²C master device starts sending data to a slave or reading data sent by the slave via I²C bus.

port
A number of the I²C slave port.

slaveAddress
A unique 7-bit number, factory assigned to each I²C slave device.

size
A size of the buffer that stores the event data. This parameter is specified in bytes. The maximum value us 256 bytes.

6.3.2.2. DLN_I2C_SLAVE_WRITE_EV

This structure is used to store the description of `DLN_I2C_SLAVE_WRITE_EV` events.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint16_t eventCount;
   uint8_t eventType;
   uint8_t port;
   uint8_t slaveAddress;
   uint16_t size;
   uint8_t buffer[DLN_I2C_SLAVE_BUFFER_SIZE];
} __PACKED_ATTR DLN_I2C_SLAVE_WRITE_EV;
```

Members:

header
Defines the DLN message header DLN_MSG_HEADER. This structure's header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_i2C_SLAVE_WRITE_EV structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_WRITE_EV response it is set to `0x0C11`. The `DLN_MSG_ID_I2C_SLAVE_WRITE_EV` constant can be used.
- **echoCounter**
- **handle** - A handle to the DLN-series adapter.

eventCount
Contains the number of generated events after configuration setting.

eventType
Contains the condition for event generation on the I²C-slave port. The following values are available:

- `0` or `DLN_I2C_SLAVE_EVENT_NONE` - no events are generated for the current port.
- `1` or `DLN_I2C_SLAVE_EVENT_READ` - events are generated when the master device starts reading data sent by the slave via I²C bus.
- `2` or `DLN_I2C_SLAVE_EVENT_WRITE` - events are generated when the I²C master device starts sending data to a slave via I²C bus.
- `3` or `DLN_I2C_SLAVE_EVENT_READ_WRITE` - events are generated when the I²C master device starts sending data to a slave or reading data sent by the slave via I²C bus.

port
A number of the I²C slave port.

slaveAddress
A unique 7-bit number, factory assigned to each I²C slave device.

size
A size of the buffer that stores the event data. This parameter is specified in bytes. The maximum value us 256 bytes.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer contains the data received from the master via I²C bus.

6.3.3. Functions

6.3.3.1. DlnI2cSlaveGetPortCount()

```
DLN_RESULT DlnI2cSlaveGetPortCount(
   HDLN handle,
   uint8_t* count
);
```

The `DlnI2cSlaveGetPortCount()` function retrieves the total number of I²C slave ports available in your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. This integer will be filled with the number of available I²C slave ports after the function execution.

Table 6.1. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.2. DlnI2cSlaveEnable()

```
DLN_RESULT DlnI2cSlaveEnable(
   HDLN handle,
   uint8_t port,
   uint16_t* conflict
);
```

The `DlnI2cSlaveEnable()` function activates corresponding I²C slave port on your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to be enabled .

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used by the I²C slave module. To fix this a user has to disconnect a pin from a module that it has been assigned to and call the DlnI2cSlaveEnable() function once again. If there are any more conflicting pins, the next conflicted pin number will be returned.

Table 6.2. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.3. DlnI2cSlaveDisable()

```
DLN_RESULT DlnI2cSlaveDisable(
   HDLN handle,
   uint8_t port,
   uint8_t waitForTransferCompletion
);
```

The `DlnI2cSlaveDisable()` function deactivates corresponding I²C slave port on your DLN-Series adapter.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to be disabled.

waitForTransferCompletion
Used to choose whether the device should wait for current data transfers to complete before disabling an I²C slave. The following values are available:

- `1` or `DLN_I2C_SLAVE_WAIT_FOR_TRANSFERS` - wait until transfers are completed;
- `0` or `DLN_I2C_MASTER_CANCEL_TRANSFERS` - cancel all pending data transfers and deactivate the module.

Table 6.3. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.4. DlnI2cSlaveIsEnabled()

```
DLN_RESULT DlnI2cSlaveIsEnabled(
HDLN handle,
uint8_t port,
uint8_t* enabled
);
```

The `DlnI2cSlaveIsEnabled()` function retrieves information whether the specified I²C slave port of your DLN-series adapter is activated.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with information whether the specified I²C slave port is activated after the function execution. There are two possible values:

- `0` or `DLN_I2C_SLAVE_DISABLED` - the port is not configured as I²C slave.
- `1` or `DLN_I2C_SLAVE_ENABLED` - the port is configured as I²C slave.

Table 6.4. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.5. DlnI2cGetAddressCount()

```
DLN_RESULT DlnI2cSlaveGetAddressCount(
   HDLN handle,
   uint8_t port,
   uint8_t* count
);
```

The `DlnI2cSlaveGetAddressCount()` function retrieves the number of I²C slave addresses that can be assigned to the specified I²C slave module of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter

port
A number of the I²C slave port to retrieve the information from.

count
A pointer to an unsigned 8-bit integer. The integer will be filled with the number of slave addresses that can be assigned to the I²C slave module of your DLN-series adapter after the function execution.

Table 6.5. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Max address number	N/A	1	N/A
Defined in	dln_i2c_slave.h		

6.3.3.6. DlnI2cSlaveSetAddress()

```
DLN_RESULT DlnI2cSlaveSetAddress(
   HDLN handle,
   uint8_t port,
   uint8_t slaveAddressNumber,
   uint8_t address
);
```

The `DlnI2cSlaveSetAddress()` function assigns an I²C slave address to the specified I²C slave module of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
An I²C slave port to be configured.

slaveAddressNumber
A number of the I²C slave address to be assigned.

address
An I²C slave address to be assigned to the specified I²C slave module of your DLN-series adapter.

Table 6.6. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Max address number	N/A	1	N/A
Defined in	dln_i2c_slave.h		

6.3.3.7. DlnI2cSlaveGetAddress()

```
DLN_RESULT DlnI2cSlaveGetAddress(
   HDLN handle,
   uint8_t port,
   uint8_t slaveAddressNumber,
   uint8_t* address
);
```

The `DlnI2cSlaveGetAddress()` function retrieves one of the slave addresses, assigned to the I²C slave module of your DLN-series adapter.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to retrieve the information from.

slaveAddressNumber
A number of the I²C slave address to be retrieved.

address
A pointer to an unsigned 8-bit integer. The integer will be filled with the currently assigned I²C slave address with the specified number after the function execution.

Table 6.7. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.8. DlnI2cSlaveGeneralCallEnable()

```
DLN_RESULT DlnI2cSlaveGeneralCallEnable(
   HDLN handle,
   uint8_t port,
);
```

The `DlnI2cSlaveGeneralCallEnable()` function allows addressing all I²C slave devices on the circuit by sending the `0` slave address.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to be configured.

Table 6.8. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.9. DlnI2cSlaveGeneralCallDisable()

```
DLN_RESULT DlnI2cSlaveGeneralCallDisable(
   HDLN handle,
   uint8_t port,
);
```

The `DlnI2cSlaveGeneralCallDisable()` function disables addressing all I²C slave devices on the circuit by sending the `0` slave address.

Parameters

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to be configured.

Table 6.9. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.10. DlnI2cSlaveGeneralCallIsEnabled()

```
DLN_RESULT DlnI2cSlaveGeneralCallIsEnabled(
   HDLN handle,
   uint8_t port,
   uint8_t* enabled
);
```

The `DlnI2cSlaveGeneralCallIsEnabled()` function retrieves information whether addressing all I²C slave devices on the circuit by sending the `0` slave address is activated.

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to retrieve the information from.

enabled
A pointer to an unsigned 8-bit integer. The integer will be filled with information whether adressing all I²C slave devices on the circuit by sending the `0` slave address is enabled after the function execution. There are two possible values:

- `0` or `DLN_I2C_SLAVE_GENERAL_CALL_DISABLED` - addressing all I²C slave devices on the circuit by sending the `0` slave address is disabled.
- `1` or `DLN_I2C_SLAVE_GENERAL_CALL_ENABLED` - addressing all I²C slave devices on the circuit by sending the `0` slave address is enabled.

Table 6.10. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.11. DlnI2cSlaveLoadReply()

The `DlnI2cSlaveLoadReply()` function loads information to be transferred to an I²C master device.

```
DLN_RESULT DlnI2cSlaveLoadReply(
   HDLN handle,
   uint8_t port,
   uint16_t size,
   uint8_t* buffer
```

```
);
```

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to be used.

size
A size of the message buffer. This parameter is specified in bytes. The value must fall within the range from 1 to 256 bytes.

buffer
A pointer to an array of unsigned 8-bit integers. Each of the elements is an 8-bit value. The buffer must contain the information to be sent to an I²C master during the function execution (in the range from 1 to 256 bytes).

Table 6.11. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.12. DlnI2cSlaveSetEvent()

The `DlnI2cSlaveSetEvent()` function configures the event generation conditions for the I²C slave port of the DLN-series adapter.

```
DLN_RESULT DlnI2cSlaveSetEvent(
   HDLN handle,
   uint8_t port,
   uint8_t slaveAddressNumber,
   uint8_t eventType,
);
```

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to be configured.

slaveAddressNumber
Must contain the number of the I²C slave address.

eventType
Must contain the condition for event generation on the I²C slave port. The following values are available:

- `0` or `I2C_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `I2C_SLAVE_EVENT_READ` - events are generated when a master device starts reading data sent by a slave via I²C bus.

- 2 or `I2C_SLAVE_EVENT_WRITE` - events are generated when a master device starts sending data to a slave via I²C bus.
- 3 or `I2C_SLAVE_EVENT_READ_WRITE` - Events are generated when a master device starts either reading or sending data via I²C interface.

Table 6.12. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.3.13. DlnI2cSlaveGetEvent()

The `DlnI2cSlaveGetEvent()` function retrieves currently set event generation conditions for the specified I²C slave port.

```
DLN_RESULT DlnI2cSlaveGetEvent(
   HDLN handle,
   uint8_t port,
   uint16_t slaveAddressNumber
   uint8_t* eventType,
);
```

Parameters:

handle
A handle to the DLN-series adapter.

port
A number of the I²C slave port to be used.

slaveAddressNumber
A number of the I²C slave address, assigned to your DLN-series adapter.

eventType
A pointer to an unsigned 8-bit integer. The integer will be filled with current event generation conditions for the specified SPI slave port. The followng values are available:

- `0` or `I2C_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `I2C_SLAVE_EVENT_READ` - events are generated when a master device starts reading data sent by a slave via I²C bus.
- 2 or `I2C_SLAVE_EVENT_WRITE` - events are generated when a master device starts sending data to a slave via I²C bus.
- 3 or `I2C_SLAVE_EVENT_READ_WRITE` - Events are generated when a master device starts either reading or sending data via I²C interface.

Table 6.13. Function summary

	DLN-4M	DLN-4S	DLN-2
Supporting adapters	✗	✓	✗
Required firmware	N/A	All versions	N/A
Defined in	dln_i2c_slave.h		

6.3.4. Commands and Responses

6.3.4.1. DLN_I2C_SLAVE_GET_PORT_COUNT

DLN_I2C_SLAVE_GET_PORT_COUNT Command

Go to response

The **DLN_I2C_SLAVE_GET_PORT_COUNT** command is used to retrieve the number of I2C slave ports available in your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_I2C_SLAVE_GET_PORT_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_GET_PORT_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_PORT_COUNT command it must be set to `0x0C00`. You can use the `DLN_MSG_ID_I2C_SLAVE_GET_PORT_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_I2C_SLAVE_GET_PORT_COUNT Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_GET_PORT_COUNT** response after the command execution. The response contains the available number of I2C slave ports.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_I2C_SLAVE_GET_PORT_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_GET_PORT_COUNT_RSP structure.

- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_PORT_COUNT response it is set to `0x0C00`. The `DLN_MSG_ID_I2C_SLAVE_GET_PORT_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the available number of I²C slave ports has been successfully obtained.

count
Contains the available number of I²C slave ports.

6.3.4.2. DLN_I2C_SLAVE_ENABLE

DLN_I2C_SLAVE_ENABLE Command

Go to response

The **DLN_I2C_SLAVE_ENABLE** command is used to activate the corresponding I²C slave port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_SLAVE_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_ENABLE command it must be set to `0x0C01`. You can use the `DLN_MSG_ID_I2C_SLAVE_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C slave port to retrieve the information from.

DLN_I2C_SLAVE_ENABLE Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_ENABLE** response after the command execution. The `result` field informs a user if the specified I²C slave port has been successfully activated.

```
typedef struct
```

```
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint16_t conflict;
} __PACKED_ATTR DLN_I2C_SLAVE_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_ENABLE response it is set to `0x0C01`. The `DLN_MSG_ID_I2C_SLAVE_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified I2C slave port has been successfully activated.

conflict
A pointer to an unsigned 16-bit integer. This integer can be filled with a number of the conflicted pin, if any.

A conflict arises if a pin is already assigned to another module of the DLN-series adapter and cannot be used by the I2C module. To fix this a user has to disconnect a pin from a module that it has been assigned to and send the DLN_I2C_SLAVE_ENABLE command once again. In case there still are conflicted pins, only the number of the next one will be returned.

6.3.4.3. DLN_I2C_SLAVE_DISABLE

DLN_I2C_SLAVE_DISABLE Command

Go to response

The **DLN_I2C_SLAVE_DISABLE** command is used to deactivate the specified I2C slave port of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_SLAVE_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_DISABLE command it must be set to `0x0C02`. You can use the `DLN_MSG_ID_I2C_SLAVE_DISABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C slave port to be deactivated.

DLN_I2C_SLAVE_DISABLE Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_DISABLE** response after the function execution. The result field informs a user if the specified I²C slave port has been successfully deactivated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_SLAVE_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_DISABLE response it is set to `0x0C02`. The `DLN_MSG_ID_I2C_SLAVE_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the specified I²C slave port has been successfully deactivated.

6.3.4.4. DLN_I2C_SLAVE_IS_ENABLED

DLN_I2C_SLAVE_IS_ENABLED Command

Go to response

The **DLN_I2C_SLAVE_IS_ENABLED** command is used to retrieve information, whether the specified I²C slave port is activated.

```
typedef struct
{
```

```
   DLN_MSG_HEADER header;
   uint8_t port;
}__PACKED_ATTR DLN_I2C_SLAVE_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_IS_ENABLED command it must be set to `0x0C03`. You can use the `DLN_MSG_ID_I2C_SLAVE_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C slave port to retrieve the information from.

DLN_I2C_SLAVE_IS_ENABLED Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_IS_ENABLED** response after the command execution. The response informs a user if the specified I2C slave port is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_I2C_SLAVE_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_IS_ENABLED response it is set to `0x0C03`. The `DLN_MSG_ID_I2C_SLAVE_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- DLN_RES_SUCCESS - The state of the I2C slave port has been successfully retrieved.

enabled
Informs whether the specified I2C slave port is activated. There are two possible values:

- 0 or DLN_I2C_SLAVE_DISABLED - the I2C slave port is activated.
- 1 or DLN_I2C_SLAVE_ENABLED - the I2C slave port is deactivated.

6.3.4.5. DLN_I2C_SLAVE_GET_ADDRESS_COUNT

DLN_I2C_SLAVE_GET_ADDRESS_COUNT Command

Go to response

The **DLN_I2C_SLAVE_GET_ADDRESS_COUNT** command is used to retrieve the number of I2C slave addresses that can be assigned to the specified I2C slave module of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_SLAVE_GET_ADDRESS_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_GET_ADDRESS_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_ADDRESS_COUNT command it must be set to 0x0C04. You can use the DLN_MSG_ID_I2C_SLAVE_GET_ADDRESS_COUNT constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same echoCounter value. This field can be any numerical value from 0 to 0xFFFF (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C slave port to retrieve the information from.

DLN_I2C_SLAVE_GET_ADDRESS_COUNT Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_GET_ADDRESS_COUNT** response after the command execution. The response will contain the number of I2C slave addresses that can be assigned to the specified I2C slave module of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
```

```
} __PACKED_ATTR DLN_I2C_SLAVE_GET_ADDRESS_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_GET_ADDRESS_COUNT_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_ADDRESS_COUNT response it is set to `0x0C04`. The `DLN_MSG_ID_I2C_SLAVE_GET_ADDRESS_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The number of I²C slave addresses that can be assigned to the DLN-series adapter has been successfully retrieved.

count
Contains the number of slave addresses that can be assigned to the I²C slave module of your DLN-series adapter.

6.3.4.6. DLN_I2C_SLAVE_SET_ADDRESS

DLN_I2C_SLAVE_SET_ADDRESS Command

Go to response

The **DLN_I2C_SLAVE_SET_ADDRESS** command is used to assign an I²C slave address to the specified I²C slave module of DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t slaveAddressNumber;
   uint8_t address;
} __PACKED_ATTR DLN_I2C_SLAVE_SET_ADDRESS_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_SET_ADDRESS_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_SET_ADDRESS_COUNT command it must be set to `0x0C05`. You can use the `DLN_MSG_ID_I2C_SLAVE_SET_ADDRESS` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C slave port to be configured.

slaveAddressNumber
A number of the I²C slave address to be assigned.

address
An I²C slave address to be assigned to the specified I²C slave module of your DLN-series adapter.

DLN_I2C_SLAVE_SET_ADDRESS Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_SET_ADDRESS** response after the command execution. The `result` field informs a user whether the I²C slave address has been successfully assigned to the I²C slave module of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_SLAVE_SET_ADDRESS_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_SET_ADDRESS_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_SET_ADDRESS response it is set to `0x0C05`. The `DLN_MSG_ID_I2C_SLAVE_SET_ADDRESS` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The new I²C slave address has been successfully assigned.

6.3.4.7. DLN_I2C_SLAVE_GET_ADDRESS

DLN_I2C_SLAVE_GET_ADDRESS Command

Go to response

The **DLN_I2C_SLAVE_GET_ADDRESS** command is used to retrieve one of the slave addresses, assigned to the I²C slave module of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t slaveAddressNumber;
} __PACKED_ATTR DLN_I2C_SLAVE_GET_ADDRESS_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_GET_ADDRESS_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_ADDRESS_COUNT command it must be set to `0x0C06`. You can use the `DLN_MSG_ID_I2C_SLAVE_GET_ADDRESS` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C slave port to retrieve the information from.

slaveAddressNumber
A number of the I2C slave address to be retrieved.

DLN_I2C_SLAVE_GET_ADDRESS Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_GET_ADDRESS** response after the command execution. The response contains the I2C slave address with the specified number, currently assigned to I2C slave module of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t address;
} __PACKED_ATTR DLN_I2C_SLAVE_GET_ADDRESS_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_GET_ADDRESS_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_ADDRESS response it is set to `0x0C06`. The `DLN_MSG_ID_I2C_SLAVE_GET_ADDRESS` constant can be used.

- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The I²C slave address with the specified number has been successfully retrieved.

address
The currently assigned I²C slave address with the specified number.

6.3.4.8. DLN_I2C_SLAVE_GENERAL_CALL_ENABLE

DLN_I2C_SLAVE_GENERAL_CALL_ENABLE Command

Go to response

The **DLN_I2C_SLAVE_GENERAL_CALL_ENABLE** command is used to allow addressing all I²C slave devices on the circuit by sending the `0` slave address.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_SLAVE_GENERAL_CALL_ENABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_GENERAL_CALL_ENABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GENERAL_CALL_ENABLE command it must be set to `0x0C07`. You can use the `DLN_MSG_ID_I2C_SLAVE_GENERAL_CALL_ENABLE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C slave port to be configured.

DLN_I2C_SLAVE_GENERAL_CALL_ENABLE Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_GENERAL_CALL_ENABLE** response after the command execution. The `result` field informs a user if addressing all I²C slave devices on the circuit using i²C slave address `0` has been successfully enabled.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_SLAVE_GENERAL_CALL_ENABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_GENERAL_CALL_ENABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_ENABLE response it is set to `0x0C07`. The `DLN_MSG_ID_I2C_SLAVE_GENERAL_CALL_ENABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - addressing using slave address `0` has been successfully enabled for the specified I2C slave port.

6.3.4.9. DLN_I2C_SLAVE_GENERAL_CALL_DISABLE

DLN_I2C_SLAVE_GENERAL_CALL_DISABLE Command

Go to response

The **DLN_I2C_SLAVE_GENERAL_CALL_DISABLE** command is used to disable addressing all I2C slave devices on the circuit by sending the `0` slave address.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
} __PACKED_ATTR DLN_I2C_SLAVE_GENERAL_CALL_DISABLE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_GENERAL_CALL_DISABLE_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GENERAL_CALL_DISABLE command it must be set to `0x0C08`. You can use the `DLN_MSG_ID_I2C_SLAVE_GENERAL_CALL_DISABLE` constant.

- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C slave port to be configured.

DLN_I2C_SLAVE_GENERAL_CALL_DISABLE Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_GENERAL_CALL_DISABLE** response after the function execution. The result field informs a user if addressing all I²C slave devices on the circuit using slave address `0` has been successfully deactivated for the specified I²C slave port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_SLAVE_GENERAL_CALL_DISABLE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_GENERAL_CALL_DISABLE_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GENERAL_CALL_DISABLE response it is set to `0x0C08`. The `DLN_MSG_ID_I2C_SLAVE_GENERAL_CALL_DISABLE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - addressing all I²C slave devices on the circuit using slave address `0` has been successfully deactivated.

6.3.4.10. DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED

DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED Command

Go to response

The **DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED** command is used to retrieve information whether addressing all I²C slave devices on the circuit by sending the `0` slave address is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
```

```
   uint8_t port;
}__PACKED_ATTR DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED command it must be set to `0x0C08`. You can use the `DLN_MSG_ID_I2C_SLAVE_GENERAL_CALL_IS_ENABLED` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C slave port to retrieve the information from.

DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED** response after the command execution. The response informs a user whether addressing all I2C slave devices on the circuit by sending the `0` slave address is activated.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t enabled;
}__PACKED_ATTR DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GENERAL_CALL_IS_ENABLED response it is set to `0x0C08`. The `DLN_MSG_ID_I2C_SLAVE_GENERAL_CALL_IS_ENABLED` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The state of the I²C slave port has been successfully retrieved.

enabled
Informs whether ms a user whether adressing all I²C slave devices on the circuit by sending the `0` slave address is enabled. There are two possible values:

- `0` or `DLN_I2C_SLAVE_GENERAL_CALL_DISABLED` - addressing all I²C slave devices on the circuit by sending the `0` slave address is disabled.
- `1` or `DLN_I2C_SLAVE_GENERAL_CALL_ENABLED` - addressing all I²C slave devices on the circuit by sending the `0` slave address is enabled.

6.3.4.11. DLN_I2C_SLAVE_LOAD_REPLY

DLN_I2C_SLAVE_LOAD_REPLY Command

Go to response

The **DLN_I2C_SLAVE_LOAD_REPLY** Command is used to load information to be transferred to an I²C-master device.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint16_t size;
   uint8_t buffer[DLN_I2C_SLAVE_BUFFER_SIZE];
} __PACKED_ATTR DLN_I2C_SLAVE_LOAD_REPLY_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_LOAD_REPLY_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_LOAD_REPLY command it must be set to `0x0C0A`. You can use the `DLN_MSG_ID_I2C_SLAVE_LOAD_REPLY` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C slave port to be used.

size
A size of the message buffer. This parameter is specified in bytes. The value must fall within the range of 1 to 256 bytes.

buffer
A 256-element array. Each of the elements is an 8-bit value. The buffer must contain the information to be sent to the I²C master.

DLN_I2C_SLAVE_LOAD_REPLY Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_LOAD_REPLY** response after the command execution. The response informs a user if the data to be sent to an I2C master were successfully loaded.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2#_SLAVE_LOAD_REPLY_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_LOAD_REPLY_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_LOAD_REPLY response it is set to `0x0C0A`. The `DLN_MSG_ID_I2C_SLAVE_LOAD_REPLY` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The information has been successfully loaded.

6.3.4.12. DLN_I2C_SLAVE_SET_EVENT

DLN_I2C_SLAVE_SET_EVENT Command

Go to response

The **DLN_I2C_SLAVE_SET** command is used to configure the event generation conditions for the I2C slave module of the DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t slaveAddressNumber;
   uint8_t eventType;
} __PACKED_ATTR DLN_I2C_SLAVE_SET_EVENT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_SET_EVENT_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_SET_EVENT command it must be set to `0x0B0B`. You can use the `DLN_MSG_ID_I2C_SLAVE_SET_EVENT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I²C slave port to be configured.

slaveAddressNumber
Must contain the number of the I²C slave address.

eventType
Must contain the condition for event generation on the I²C slave port. The following values are available:

- `0` or `I2C_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `I2C_SLAVE_EVENT_READ` - events are generated when a master device starts reading data sent by a slave via I²C bus.
- `2` or `I2C_SLAVE_EVENT_WRITE` - events are generated when a master device starts sending data to a slave via I²C bus.
- `3` or `I2C_SLAVE_EVENT_READ_WRITE` - Events are generated when a master device starts either reading or sending data via I²C interface.

DLN_I2C_SLAVE_SET_EVENT Response

The adapter sends the **DLN_I2C_SLAVE_SET_EVENT** Response after the command execution. The `result` field informs a user if the I2C-slave event generation conditions have been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_I2C_SLAVE_SET_EVENT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_SET_EVENT_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_SET_EVENT response it is set to `0x0B0B`. The `DLN_MSG_ID_I2C_SLAVE_SET_EVENT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The I2C event generation conditions have been successfully set.

6.3.4.13. DLN_SPI_SLAVE_GET_EVENT

DLN_I2C_SLAVE_GET_EVENT Command

Go to response

The **DLN_I2C_SLAVE_GET_EVENT** command is used to retrieve currently set event generation conditions for the specified I2C slave port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t port;
   uint8_t slaveAddressNumber;
} __PACKED_ATTR DLN_I2C_SLAVE_GET_EVENT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_I2C_SLAVE_GET_EVENT_CMD structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_EVENT command it must be set to `0x0B0#`. You can use the `DLN_MSG_ID_I2#_SLAVE_GET_EVENT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

port
A number of the I2C slave port to retrieve the information from.

slaveAddressNumber
A number of the I2C slave address, assigned to your DLN-series adapter.

DLN_I2C_SLAVE_GET_EVENT Response

Go to command

The adapter sends the **DLN_I2C_SLAVE_GET_EVENT** response after the command execution. The response contains current event generation conditions for the specified I2C slave port.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t eventType;
} __PACKED_ATTR DLN_I2C_SLAVE_GET_EVENT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_I2C_SLAVE_GET_EVENT_RSP structure.
- **msgId** - Defines the message. For the DLN_I2C_SLAVE_GET_EVENT response it is set to `0x0B0C`. The `DLN_MSG_ID_I2C_SLAVE_GET_EVENT` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - The event generation conditions have been successfully retrieved.

eventType
Contains the condition for event generation on the I2C slave port. The following values are available:

- `0` or `I2C_SLAVE_EVENT_NONE` - no events are generated for the current port;
- `1` or `I2C_SLAVE_EVENT_READ` - events are generated when a master device starts reading data sent by a slave via I2C bus.
- `2` or `I2C_SLAVE_EVENT_WRITE` - events are generated when a master device starts sending data to a slave via I2C bus.
- `3` or `I2C_SLAVE_EVENT_READ_WRITE` - Events are generated when a master device starts either reading or sending data via I2C interface.

Chapter 7. LEDs module

All DLN-series adapters are fitted with a single status LED and several user-controlled LEDs (depends on the model).

The status LED shows current state of your device.

Each of the user-controlled LEDs can be adjusted to a specific state using DlnLedSetState() function.

You can find out the number of available user-controlled LEDs using DlnLedGetCount() function

> **Warning**
>
> A **DLN-4** device can be purchased with an optional enclosure. Such enclosures have no openings for LEDs. Therefore, you will not be able to see the LEDs through the enclosure.

7.1. Available LED states

DLN_LED_STATE - is a user-defined type, whose value defines current behavior of a LED.

Below is the list of its possible values:

Table 7.1. LED States

State	LED Behavior
DLN_LED_STATE_OFF	Off
DLN_LED_STATE_ON	On
DLN_LED_SLOW_BLINK	Blinking slowly
DLN_LED_FAST_BLINK	Blinking rapidly
DLN_LED_DOUBLE_BLINK	Blinking in a double-blink pattern
DLN_LED_TRIPLE_BLINK	Blinking in a triple-blink pattern

7.2. Functions

This section describes the LED module functions. They are used to control and monitor user-controlled LEDs, used in the device.

Actual control of the device is performed by use of commands and responses. Each function utilizes respective commands and responses. You can send such commands directly if necessary.

- DlnLedGetCount() - retrieves the total number of user-controlled LEDs available in the device;
- DlnLedSetState() - sets a new state of the user-controlled LED;
- DlnLedGetState() - retrieves the current state of the LED.

7.2.1. DlnLedGetCount()

```
DLN_RESULT DlnLedGetCount(
   HDLN handle,
   uint8_t* count
);
```

The `DlnLedGetCount()` function retrieves the total number of user-controlled LEDs available in the device.

Parameters:

handle
A handle to the DLN-series adapter.

count
A pointer to an unsigned 8-bit integer. This integer will store the number of user-controlled LEDs after function execution.

This function is defined in the `dln_led.h` file.

7.2.2. DlnLedSetState()

```
DLN_RESULT DlnLedSetState(
   HDLN handle,
   uint8_t ledNumber,
   DLN_LED_STATE state
);
```

The `DlnLedSetState` function sets a new state of the user-controlled LED.

Parameters:

handle
A handle to the DLN-series adapter.

ledNumber
A LED, whose state we need to change.

state
A LED state to be set.

This function is defined in the `dln_led.h` file.

7.2.3. DlnLedGetState()

```
DLN_RESULT DlnLedGetState(
   HDLN handle,
   uint8_t ledNumber
   DLN_LED_STATE* state
);
```

The `DlnLedGetState()` function retrieves the current state of the LED.

Parameters:

handle
A handle to the DLN-series adapter.

ledNumber
A LED to obtain a state from.

state
A pointer to a `DLN_LED_STATE` structure to store the obtained LED state.

This function is defined in the `dln_led.h` file.

7.3. Commands and responses

7.3.1. DLN_LED_GET_COUNT

DLN_LED_GET_COUNT Command

Go to Response

DLN_LED_GET_COUNT command is used to retrieve the number of user-controlled LEDs from a device.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_LED_GET_COUNT_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_LED_GET_COUNT_CMD structure.
- **msgId** - Defines the message. For the DLN_LED_GET_COUNT command it must be set to `0x0800`. You can use the `DLN_MSG_ID_LED_GET_COUNT` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either the handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

DLN_LED_GET_COUNT Response

Go to Command

The adapter sends **DLN_LED_GET_COUNT** response after the command execution. The response contains the number of user-controlled LEDs.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t count;
} __PACKED_ATTR DLN_LED_GET_COUNT_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_LED_GET_COUNT_RSP structure.

- **msgId** - Defines the message. For the DLN_LED_GET_COUNT response it is set to `0x0800`. The `DLN_MSG_ID_LED_GET_COUNT` constant can be used.
- **echoCounter** - Links a command to a response. The echoCounter value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either the handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the number of LEDs has been successfully obtained.

count
Contains the number of user-controlled LEDs.

7.3.2. DLN_LED_SET_STATE

DLN_LED_SET_STATE Command

Go to Response

The DLN_LED_SET_STATE command is used to set LED states.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t ledNumber;
   DLN_LED_STATE state;
} __PACKED_ATTR DLN_LED_SET_STATE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_LED_SET_STATE_CMD structure.
- **msgId** - Defines the message. For the DLN_LED_SET_STATE command it must be set to `0x0801`. You can use the `DLN_MSG_ID_LED_SET_STATE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You may specify either the handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

ledNumber
Contains the number of the LED, whose state we are changing.

state
Contains the value of the state to be set.

DLN_LED_SET_STATE_Response

Go to Command

The adapter sends the **DLN_LED_SET_STATE** response after the command execution. The `result` field informs a user if a LED state has been successfully set.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_LED_SET_STATE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_LED_SET_STATE_RSP structure.
- **msgId** - Defines the message. For the DLN_LED_SET_STATE response it is set to `0x0801`. The `DLN_MSG_ID_LED_SET_STATE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either the handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the state of a LED has been successfully set;
- `DLN_RES_INVALID_LED_NUMBER` - invalid LED number has been specified;
- `DLN_RES_INVALID_LED_STATE` - invalid LED state has been specified.

7.3.3. DLN_LED_GET_STATE

DLN_LED_GET_STATE Command

Go to Response

The **LED_GET_STATE** command is used to obtain current state of a LED.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint8_t ledNumber;
} __PACKED_ATTR DLN_LED_GET_STATE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_LED_GET_STATE_CMD structure.
- **msgId** - Defines the message. For the DLN_LED_GET_STATE command it must be set to `0x0802`. You can use the `DLN_MSG_ID_LED_GET_STATE` constant.
- **echoCounter** - Can be used to link a command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter. You can specify either a handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

ledNumber
Contains the number of the LED, whose state is to be retrieved.

DLN_LED_GET_STATE Response

Go to Command

The adapter sends **DLN_LED_GET_STATE** response after the command execution. The response will contain the retrieved state of the LED.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   DLN_LED_STATE state;
} __PACKED_ATTR DLN_LED_GET_STATE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_LED_GET_STATE_RSP structure.
- **msgId** - Defines the message. For the DLN_LED_GET_STATE response it is set to `0x0802`. The `DLN_MSG_ID_LED_GET_STATE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter. Can be either the handle to a specific device (stream) or the `HDLN_ALL_DEVICES` value.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the state of a LEDs has been successfully obtained;
- `DLN_RES_INVALID_LED_NUMBER` -invalid LED number has been specified;

state
Contains the obtained value of the state.

7.4. Types

Chapter 8. Bootloader Module

The bootloader module is installed in every DLN-series adapter. It is used to update firmware in order to apply new features, updates and fixes.

Since DLN-series adapters are operated via the Application Programming Interface (API), a user can also update firmware for remote devices.

> **Warning**
>
> Firmware for DLN-series adapters is supplied encoded. Each DLN-series adapter is only compatible with specifically designed firmware. Do not try to install an incompatible firmware.

8.1. Device modes

There are three modes of a DLN-series adapter.:

Bootloader

This mode is activated in order to use bootloader functions. Only generic and bootloader functions are available in this mode. The Bootloader mode can be accesses either by sending the DLN_BOOT_ENTER_BOOTLOADER command or by setting the respective jumper on the DLN-series adapter board. The jumper must be set before the device is powered up. In case the device is not fitted with external power source, the jumper must be set before connecting the device to a USB port.

Application

When in this mode, the DLN-series adapter functions normally. However, some bootloader functions (DlnBootReadFlash() and DlnBootWriteFlash()) become unavailable.

Update in progress

This mode is activated automatically once the device receives the DLN_BOOT_WRITE_FLASH command. In case of a power failure, the DLN-series adapter will be rebooted and remain in this mode

8.2. Structures

8.2.1. DLN_BOOT_FLASH_DESC

```
typedef struct
{
   uint32_t appStartAddr;
   uint32_t appMaxSize;
   uint32_t pageSize;
} __PACKED_ATTR DLN_BOOT_FLASH_DESC;
```

This structure is used to store configuration of the device internal flash memory.

Members:

appStartAddr
A bit field, consisting of 32 bits. It is used to store the starting address of the device firmware.

appMaxSize
A bit field, consisting of 32 bits. It is used to store the maximum size of the device firmware.

pageSize
A bit field, consisting of 32 bits.

8.3. Functions

This section describes the Bootloader functions. They are used to control and monitor the Bootloader module of a DLN-series adapter.

Actual control of the device is performed by use of commands and responses. Each function utilizes respective commands and responses. You can send such commands directly if necessary.

- DlnBootGetMode() - retrieves current mode of your DLN-series adapter.
- DlnBootEnterBootloader() - enters the Bootloader mode.
- DlnBootExitBootloader() - exits the Bootloader mode and continues in the Application mode
- DlnBootGetFlashDesc() - retrieves information about the device internal flash memory.
- DlnBootWriteFlash() - writes new firmware into the device internal memory.
- DlnBootReadFlash() - reads data on the DLN-series adapter current firmware.

8.3.1. DlnBootGetMode()

```
DLN_RESULT DlnBootGetMode(
   HDLN handle,
   DLN_BOOT_MODE* mode
);
```

The `DlnBootGetMode()` function retrieves current mode of your DLN-series adapter.

Parameters

handle
A handle to the DLN-series adapter.

mode
A pointer to the 32-bit `DLN_BOOT_MODE` variable. This variable will be used to store current device mode after the function execution.

This function is defined in the `dln_boot.h` file.

8.3.2. DlnBootEnterBootloader()

```
DLN_RESULT DlnBootEnterBootloader(
   HDLN handle
);
```

The `DlnBootEnterBootloader()` function enters the Bootloader mode.

Parameters:

handle
A handle to the DLN-series adapter.

This function is defined in the `dln_boot.h` file.

8.3.3. DlnBootExitBootloader()

```
DLN_RESULT DlnBootExitBootloader(
   HDLN handle
);
```

The `DlnExitBootloader()` function exits the Bootloader mode and continues in the Application mode.

Parameters:

handle
A handle to the DLN-series adapter.

This function is defined in the `dln_boot.h` file.

8.3.4. DlnBootGetFlashDesc()

```
DLN_RESULT DlnBootGetFlashDesc(
   HDLN handle,
   DLN_BOOT_FLASH_DESC* desc
);
```

The `DlnBootGetFlashDesc()` function retrieves information about the device internal flash memory.

Parameters:

handle
A handle to the DLN-series adapter.

desc
A pointer to the DLN_BOOT_FLASH_DESC structure, which will be filled with data after the function execution.

This function is defined in the `dln_boot.h` file.

8.3.5. DlnBootWriteFlash()

```
DLN_RESULT DlnBootWriteFlash(
   HDLN handle,
   uint32_t address,
   uint16_t size,
   uint8_t* buffer
);
```

The `DlnBootWriteFlash()` function writes new firmware into the device internal memory.

This function is only available, when the device functions in the Bootloader mode.

handle
A handle to the DLN-series adapter.

address
The address to start writing from.

size
The size of the firmware.

buffer

This function is defined in the `dln_boot.h` file.

8.3.6. DlnBootReadFlash()

```
DLN_RESULT DlnBootReadFlash(
   HDLN handle,
   uint32_t address,
   uint16_t size,
   uint8_t* buffer
);
```

The `DlnBootReadFlash()` function reads data on the DLN-series adapter current firmware.

This function is only available, when the device functions in the Bootloader mode.

Parameters:

handle
A handle to the DLN-series adapter.

address
Address to start reading from.

size
Size of the data to read.

buffer
A pointer to an unsigned 8 bit integer. This integer will be filled with the firmware data after the function execution.

This function is defined in the `dln_boot.h` file.

8.4. Commands and Responses

8.4.1. DLN_BOOT_GET_MODE

DLN_BOOT_GET_MODE Command

Go to response

The **DLN_BOOT_GET_MODE** command is used to obtain current mode of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_BOOT_GET_MODE_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_BOOT_GET_MODE_CMD structure.
- **msgId** - Defines the message. For the DLN_BOOT_GET_MODE command it must be set to `0x0900`. You can use the `DLN_MSG_ID_BOOT_GET_MODE` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_BOOT_GET_MODE Response

Go to command

The adapter sends the **DLN_BOOT_GET_MODE** response after the command execution. The response contains the current mode of your DLN-series adapter.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   DLN_BOOT_MODE mode;
} __PACKED_ATTR DLN_BOOT_GET_MODE_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_BOOT_GET_MODE_RSP structure.
- **msgId** - Defines the message. For the DLN_BOOT_GET_MODE response it is set to `0x0900`. The `DLN_MSG_ID_BOOT_GET_MODE` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the DLN-series adapter mode has been successfully retrieved.

mode
Current mode of the DLN-series adapter.

8.4.2. DLN_BOOT_ENTER_BOOTLOADER

DLN_BOOT_ENTER_BOOTLOADER Command

Go to response

The **DLN_BOOT_ENTER_BOOTLOADER** command is used in order to switch on the Bootloader mode.

```
typedef struct
{
```

```
    DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_BOOT_ENTER_BOOTLOADER_CMD;
```

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_BOOT_ENTER_BOOTLOADER_CMD structure.
- **msgId** - Defines the message. For the DLN_BOOT_ENTER_BOOTLOADER command it must be set to `0x0901`. You can use the `DLN_MSG_ID_BOOT_ENTER_BOOTLOADER` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_BOOT_ENTER_BOOTLOADER Response

Go to command

The adapter sends the **DLN_BOOT_ENTER_BOOTLOADER** response after the command execution. The `result` field informs a user if the bootloader mode has been successfully switched on.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_BOOT_ENTER_BOOTLOADER_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_BOOT_ENTER_BOOTLOADER_RSP structure.
- **msgId** - Defines the message. For the DLN_BOOT_ENTER_BOOTLOADER response it is set to `0x0901`. The `DLN_MSG_ID_BOOT_ENTER_BOOTLOADER` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the bootloader mode has been successfully switched on.

8.4.3. DLN_BOOT_EXIT_BOOTLOADER

DLN_BOOT_EXIT_BOOTLOADER Command

Go to response

The **DLN_BOOT_EXIT_BOOTLOADER** command is used to switch off the Bootloader mode and continue in Application mode.

```
typedef struct
{
   DLN_MSG_HEADER header;
} __PACKED_ATTR DLN_BOOT_EXIT_BOOTLOADER_CMD;
```

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_BOOT_EXIT_BOOTLOADER_CMD structure.
- **msgId** - Defines the message. For the DLN_BOOT_EXIT_BOOTLOADER command it must be set to `0x0902`. You can use the `DLN_MSG_ID_BOOT_EXIT_BOOTLOADER` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

DLN_BOOT_EXIT_BOOTLOADER Response

Go to command

The adapter sends the **DLN_BOOT_EXIT_BOOTLOADER** response after the command execution. The `result` field informs a user if the bootloader mode has been successfully switched off.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_BOOT_EXIT_BOOTLOADER_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_BOOT_EXIT_BOOTLOADER_RSP structure.
- **msgId** - Defines the message. For the DLN_BOOT_EXIT_BOOTLOADER response it is set to `0x0902`. The `DLN_MSG_ID_BOOT_EXIT_BOOTLOADER` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- DLN_RES_SUCCESS - the bootloader mode has been successfully switched off.

8.4.4. DLN_BOOT_GET_FLASH_DESC

DLN_BOOT_GET_FLASH_DESC Command

Go to response

The **DLN_BOOT_GET_FLASH_DESC** command is used to retrieve information about the device internal flash memory.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint32_t reserved;
} __PACKED_ATTR DLN_BOOT_GET_FLASH_DESC_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_BOOT_GET_FLASH_DESC_CMD structure.
- **msgId** - Defines the message. For the DLN_BOOT_GET_FLASH_DESC command it must be set to `0x0903`. You can use the `DLN_MSG_ID_BOOT_GET_FLASH_DESC` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

reserved
This parameter is reserved for future use.

DLN_BOOT_GET_FLASH_DESC Response

Go to command

The adapter sends the **DLN_BOOT_GET_FLASH_DESC** response after the command execution. The response contains information about the device internal flash memory.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   DLN_BOOT_FLASH_DESC desc;
} __PACKED_ATTR DLN_BOOT_GET_FLASH_DESC_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_BOOT_GET_FLASH_DESC_RSP structure.
- **msgId** - Defines the message. For the DLN_BOOT_GET_FLASH_DESC response it is set to `0x0903`. The `DLN_MSG_ID_BOOT_GET_FLASH_DESC` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the information about the device internal flash memory has been successfully retrieved.

desc
Contains the DLN_BOOT_FLASH_DESC structure.

8.4.5. DLN_BOOT_WRITE_FLASH

DLN_BOOT_WRITE_FLASH Command

Go to response

The **DLN_BOOT_WRITE_FLASH** command is used to write new firmware into the device internal memory.

This command is only available, when the device functions in the Bootloader mode. Once the DLN-series adapter receives this command, it is set to Update in progress mode.

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint32_t address;
   uint16_t size;
   uint8_t buffer[256];
} __PACKED_ATTR DLN_BOOT_WRITE_FLASH_CMD;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_BOOT_WRITE_FLASH_CMD structure.
- **msgId** - Defines the message. For the DLN_BOOT_WRITE_FLASH command it must be set to `0x0904`. You can use the `DLN_MSG_ID_BOOT_WRITE_FLASH` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

address
The address to start writing from.

size
The size of the firmware.

buffer
A 256-element array. Each of the elements is an 8-bit value.

DLN_BOOT_WRITE_FLASH Response

Go to command

The adapter sends the **DLN_BOOT_WRITE_FLASH** response after the command execution. The `result` field informs a user if the firmware has been successfully written.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
} __PACKED_ATTR DLN_BOOT_WRITE_FLASH_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_BOOT_WRITE_FLASH_RSP structure.
- **msgId** - Defines the message. For the DLN_BOOT_WRITE_FLASH response it is set to `0x0904`. The `DLN_MSG_ID_BOOT_WRITE_FLASH` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the firmware has been successfully written.

8.4.6. DLN_BOOT_READ_FLASH

DLN_BOOT_READ_FLASH Command

Go to response

The **DLN_BOOT_READ_FLASH** command is used to read the firmware, currently installed in the DLN-series adapter.

This command is only available when the DLN-series adapter functions in the Bootloader mode

```
typedef struct
{
   DLN_MSG_HEADER header;
   uint32_t address;
   uint16_t size;
} __PACKED_ATTR DLN_BOOT_READ_FLASH_CMD;
```

header
Defines the DLN message header DLN_MSG_HEADER. The header contains several predefined fields and is used to identify and route messages. When sending the command, a user must fill the following fields:

- **size** - The size of the message. Must be equal to the size of the DLN_BOOT_READ_FLASH_CMD structure.
- **msgId** - Defines the message. For the DLN_BOOT_READ_FLASH command it must be set to `0x0905`. You can use the `DLN_MSG_ID_BOOT_WRITE_FLASH` constant.
- **echoCounter** - Can be used to link the command to a response. The response will have the same `echoCounter` value. This field can be any numerical value from `0` to `0xFFFF` (65536).
- **handle** - A handle to the DLN-series adapter.

address
An address to start reading from.

size
A size of the data to be read.

DLN_BOOT_READ_FLASH Response

Go to command

The adapter sends the **DLN_BOOT_READ_FLASH** response after the command execution. The response contains data about current device firmware.

```
typedef struct
{
   DLN_MSG_HEADER header;
   DLN_RESULT result;
   uint8_t buffer[256];
} __PACKED_ATTR DLN_BOOT_READ_FLASH_RSP;
```

Parameters:

header
Defines the DLN message header DLN_MSG_HEADER. The response header contains the following fields:

- **size** - The size of the message. It is equal to the size of the DLN_BOOT_READ_FLASH_RSP structure.
- **msgId** - Defines the message. For the DLN_BOOT_READ_FLASH response it is set to `0x0905`. The `DLN_MSG_ID_BOOT_READ_FLASH` constant can be used.
- **echoCounter** - Links a command to a response. The `echoCounter` value is copied from the respective command header.
- **handle** - A handle to the DLN-series adapter.

result
Contains the result of the command execution. The following values are available:

- `DLN_RES_SUCCESS` - the firmware data have been successfully read.

buffer
A 256-element array. Each of the elements is an 8-bit value.

Chapter 9. Pulse Counter Module

Chapter 10. Demo Applications

Each DLN-series adapter is supplied with several applications. These applications are completed with their source code written on several languages and can be used as examples of interaction with the device.

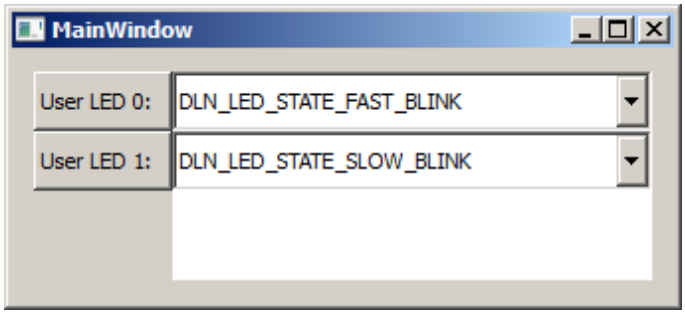
10.1. LEDs GUI

The **LEDs GUI** example shows how to configure and monitor the user-contolled LEDs, incorporated in DLN-series adapters.

The source code of the **LEDs GUI** application is available for the following programming languages and build environments:

- Qt C++
- Microsoft Visual C++

Interface



The main window of the application consists of two parts:

- the list of available LEDs (left part);

  The application automatically detects the number of LEDs available upon launch (e.g. DLN-4 has two user-controlled LEDs).

- a drop-down menu, containing available states for each LED (right part).

Operation

When a user selects one of the states for a LED from the drop-down menu, the LED begins acting in a respective manner.

Implementation Summary

When the application is launched, it uses the DlnLedGetCount() function to retrieve the number of user-controlled LEDs and creates the respective number of rows in the table widget. Then it calls the DlnLedGetState() function for each of the LEDs to get its current state and configure the corresponding Combo Box.

When a user selects a new LED state from the drop-down menu, the application uses the DlnLedSetState() function to apply the change.

See also: DlnLedGetCount(), DlnLedGetState() and DlnLedSetState().

10.2. Get Version

The **Get Version** example shows how to determine the following data about the DLN adapter:

- **Hardware type** - The type of the device (e.g. DLN-4);

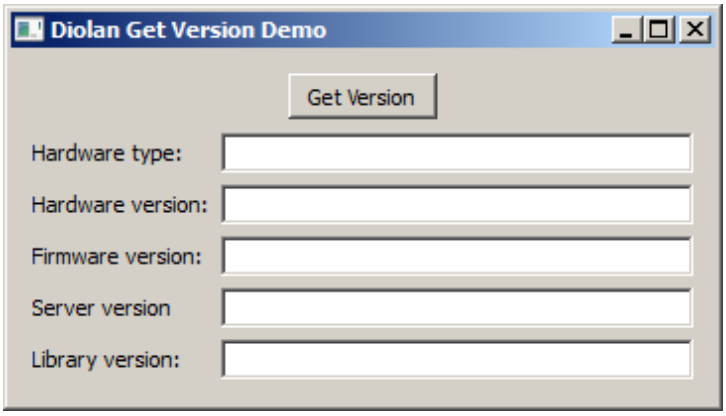
- **Hardware version** - The version of the hardware, used in the device;
- **Firmware version** - The version of the firmware, installed in the device;
- **Server version** - The version of the server;
- **Library version** - The version of the DLN-library.

The source code of the **Get Version**application is available for the following programming languages and build environments:

- Qt C++
- Microsoft Visual C++

> **Warning**
>
> In case there are several devices connected to a computer, this application will randomly choose the one to connect to. Therefore, we advise you to leave only one device connected using this application.

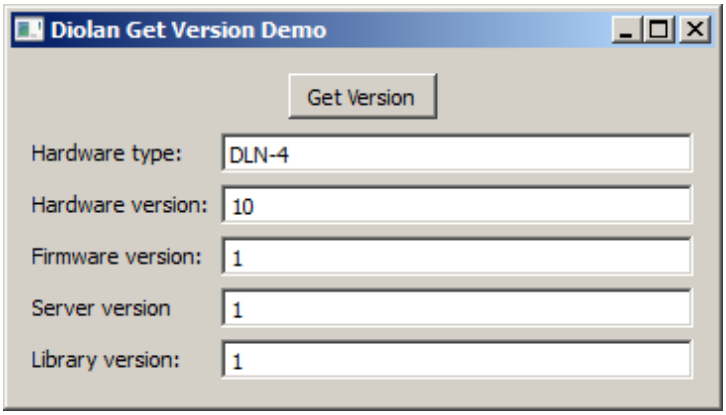
Interface



The main window contains one button, called **Get Version**, and five text fields.

Operation

Whenever a user presses the button, the application establishes a link with the device, obtains its specific data and fills the text fields.



Implementation Summary

When a user clicks the **Get Version** button, the application opens a device through DlnOpenDevice() function and calls the DlnGetVersion() function to obtain the DLN_VERSION structure as a parameter.

This structure contains such fields as `hardwareType`, `hardwareVersion`, `firmwareVersion`, `serverVersion` and `libraryVersion`. The retrieved data are then displayed in the Main window fields.

See also: DlnGetVersion(), DlnOpenDevice().

10.3. Device ID GUI

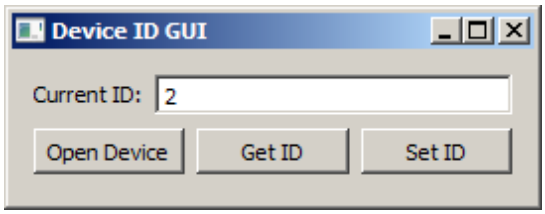
The **Device ID GUI** example shows how to set or obtain the assigned ID of the connected DLN-series adapter. The ID allows to distinguish a device in case there are several of them connected.

The source code of the **Device ID GUI** application is available for the following programming languages and build environments:

- Qt C++
- Microsoft Visual C++

> **Warning**
>
> In case there are several devices connected to a computer, this application will randomly choose the one to connect to. Therefore, we advise you to leave only one device connected using this application.

Interface



The interface of the application contains one text field, called Current ID, and three buttons:

- **Open Device** - Used to open the connected device
- **Get ID** - Used to obtain an ID, assigned to the device;
- **Set ID** - Used to assign another ID to the connected device.

Operation

Whenever a user presses the **Open Device** button, the application establishes a link with the device and retrieves its current ID.

Pressing the **Get ID** button retrieves the current ID of the opened device, which is displayed in the edit field.

In order to assign a new ID to the device, a user must enter the ID into the **Current ID** box and press the **Set ID** button. The new ID is assigned to the device and can be retrieved using the **Get ID** button.

Implementation Summary

When the application is launched, it uses the DlnOpenDevice() function to establish connection with the device. Then, the DlnGetDeviceId() function is used to get the current ID of the device, which is displayed in the edit field.

The DlnGetDeviceId() function is also called when a user presses the **Get ID** button.

When a user inputs a new ID to the edit field and presses the **Set ID** button, the application calls the DlnSetDeviceId() function to set the new device ID.

When the application is closed, it uses the DlnCloseHandle() function to close connection to the device.

See also: DlnOpenDevice(), DlnCloseHandle(), DlnGetDeviceId() and DlnSetDeviceId().

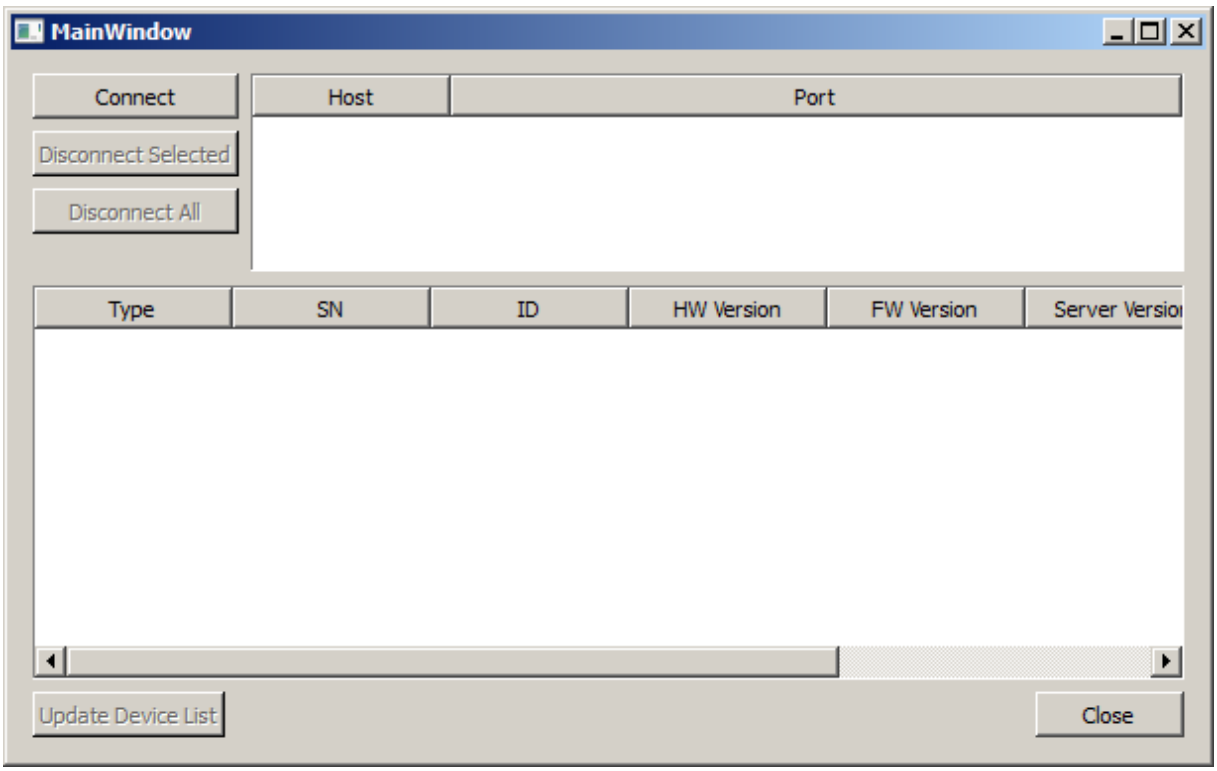
10.4. Device List GUI

Device List GUI is an application used to control and monitor the DLN-series adapters, connected to the computer.

The source code of the **Device List GUI** application is available for the following programming languages and build environments:

- Qt C++
- Microsoft Visual C++

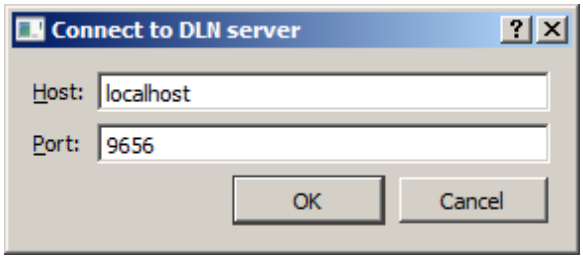
Interface



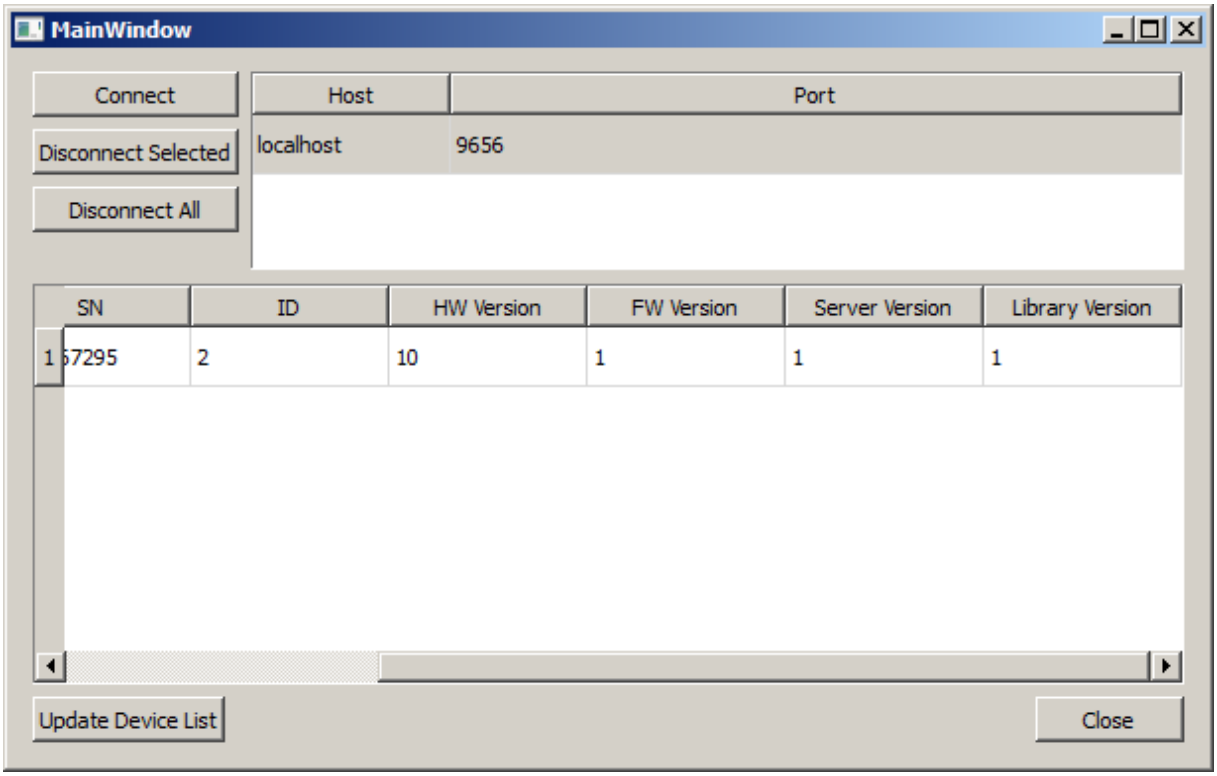
This program's interface consists of two tables and five buttons, namely:

- The upper table contains the list of all servers, connection to which has been established. It also specifies the number of the port, used for each connection;
- The lower table contains the list of all devices, connected to the servers, the connection to which has established;
- The **Connect** button is used to connect to a local or remote server;
- The **Disconnect selected** button disconnects a selected server;
- The **Disconnect all** button breaks connection with all connected servers;
- The **Close** button closes the window and exits the application.

Operation



Whenever a user presses the **Connect** button, a new window opens, prompting the user to choose a server and a port number to establish a connection. The user should enter the server's URL in the **Host** field and the port number to the *Port* field. In order to connect to a local server, please enter `localhost` to the **Host** field. Having entered the required data, click **OK** to connect or **Cancel** to close the window.



The application establishes connection with the specified port and displays all the devices, connected to the port, in the lower table. The table also contains the information about connected devices, namely **Type**, **Serial number**, **ID**, **Hardware version**, **Firmware version**, **Server version** and **Library version**. The server address and the port number are shown in the upper table.

In order to disconnect a server, press the **Disconnect Selected** button.

Warning

You should select a server to be disconnected in the upper table first.

You can also press the **Disconnect All** button to disconnect all servers.

Implementation Summary

When the **Connect** button is pressed, the application calls the DlnConnect() function. After that, the `DlnConnect()` function obtains the list of available devices.

When a user presses the **Disconnect Selected** or **Disconnect All** button, the DlnDisconnect() or DlnDisconnectAll()function is called respectively.

See also: DlnConnect(), DlnDisconnectAll(), DlnDisconnect().

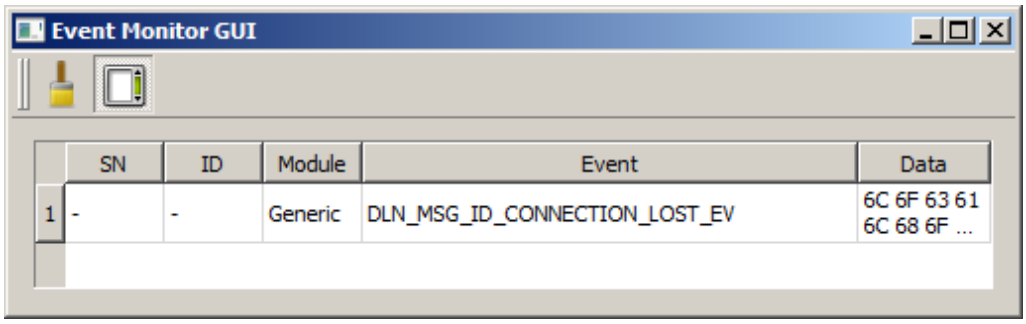
10.5. Event Monitor

Event Monitor is an application used to monitor events, sent by connected DLN-series adapters.

The source code of the **Event Monitor** application is available for the following programming languages and build environments:

- Qt C++
- Microsoft Visual C++

Interface



The main window of the application contains one table widget and two buttons:

- **Clear the events log** () - Clears all the entries from the table widget;
- **Scroll the events log when the new event arrives** () - Enables autoscroll of the events log.

Implementation Summary

Whenever launched, the application connects to a server and establishes a link to a random device. All the events, sent by the device are listed in the table widget. For each event the following information is available:

- **SN** - the serial number of the device;
- **ID** - the ID of the device;
- **Module** - the module of the device, where the event was generated (Generic, GPIO, LED etc.);
- **Event** - the name of the event;
- **Data** - the content of the event in hex code.

A user can press the **Clear the events log** () button to clear all listed events from the table widget.

The second button - **Scroll the events log when the new event arrives** () - enables autoscroll of the events list, so that the last received event is always displayed on the screen.

The application monitors the following events:

- `DLN_MSG_ID_CONNECTION_LOST_EV` - Connection to a server lost;

- `DLN_MSG_ID_DEVICE_ADDED_EV` - Device added;
- `DLN_MSG_ID_DEVICE_REMOVED_EV` - Device removed.

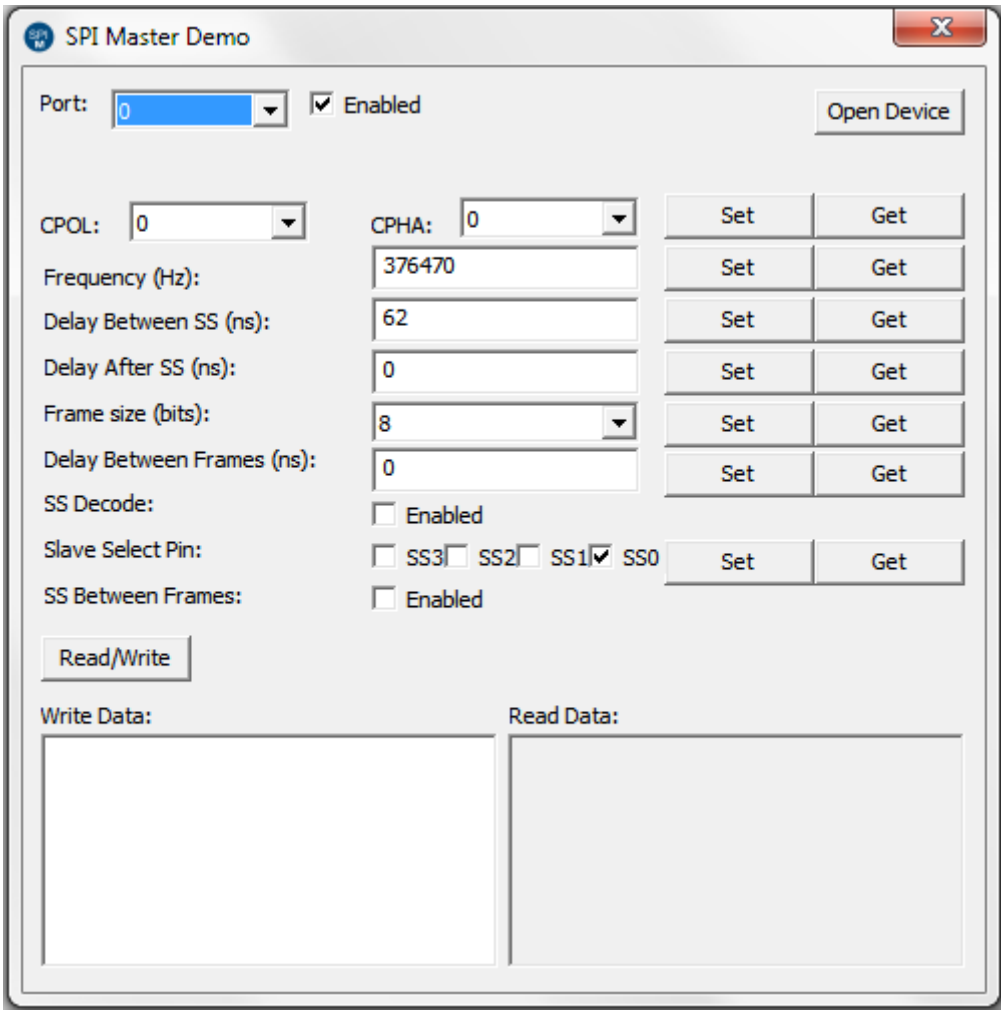
10.6. SPI Master Demo

The **SPI** (Serial Peripheal Interface) **Master Demo** example allows to control and monitor SPI master module, incorporated in DLN-series adapters.

The source code of the **SPI Master Demo** application is available for the following programming languages and build environments:

- Qt C++
- Microsoft Visual C++

Interface



The main window of the application contains:

- **Port** combobox - used to select an SPI master port of the DLN-series adapter from a dropdown list, containing available port numbers.
- **Enabled** checkbox - used to enable (checked) or disable (unchecked) the SPI master module of the DLN-series adapter.
- **Open Device** button - used to open the connected DLN-series adapter.

- **CPOL** field - used to configure a CPOL parameter by selecting one of the available values from a dropdown list. This field can also display the currently set CPOL value.
- **CPHA** field - used to configure a CPHA parameter by selecting one of the available values from a dropdown list. This field can also display the currently set CPOL value.
- **Frequency** field - used to configure SPI master frequency in Hz or display the currently set value.
- **Delay Between SS** field - used to configure a minimum delay between release of an SS line and assertion of another SS line. This field can also display the currently set delay value.
- **Frame size** field - used to configure the size of a single SPI data frame or display the currently set value.
- **Delay between frames** field - used to configure a delay between data frames, exchanged with a single slave device. This field also displays the currently set delay value.
- **SS Decode** checkbox - used to enable SS line decoding, which allows to connect more slave devices to the DLN-series adapter.
- **Slave Select Pin** checkboxes - used to select one or more SS lines to be activated.
- **SS Between Frames** checkbox - used to enable (checked) or disable (unchecked) the release of an SS line between data frames exchanged with a slave device.
- **Set** buttons - used to apply the value from the corresponding field.
- **Get** buttons - used to retrieve the corresponding value and display it in the respective field.
- **Read/Write** button - used to send and recieve data via SPI Master port.
- **Write Data** text field - fill this field with data to be send via SPI Master port.
- **Read Data** text field - this field is filled with data recieved from the SPI Slave device after SPI transaction.

10.7. Pulse Counter Demo

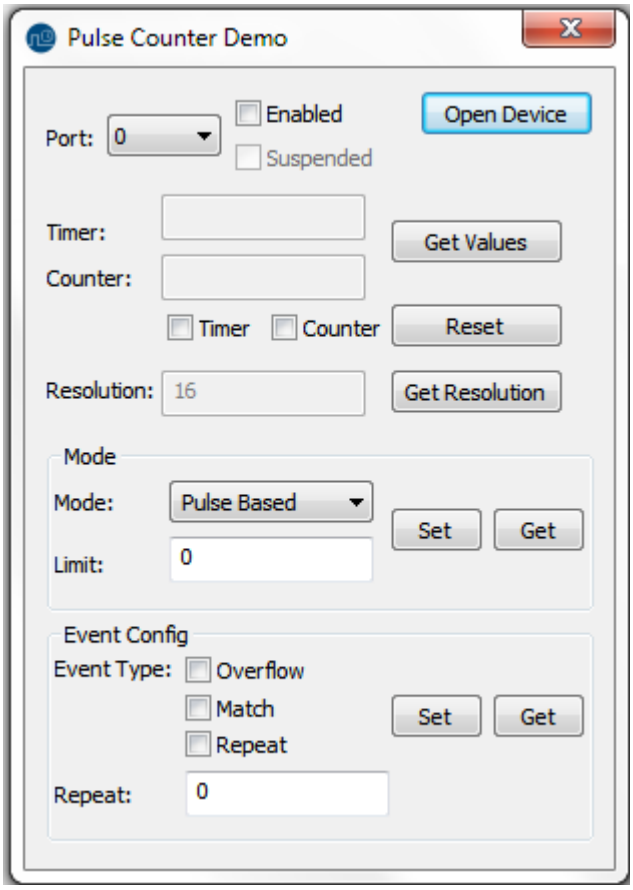
The **Pulse Counter Demo** example allows to control and monitor Pulse Counter module, incorporated in DLN-series adapters.

The source code of the **Pulse Counter Demo** application is available for the following programming languages and build environments:

- Qt C++
- Microsoft Visual C++

Interface

Figure 10.1. Pulse Counter Demo Main Window



The main window of the application contains:

- **Port** selection field - used to select an SPI master port of the DLN-series adapter from a dropdown list, containing available port numbers.
- **Enabled** checkbox - used to enable (checked) or disable (unchecked) the selected pulse counter port of the DLN-series adapter.
- **Suspended** checkbox - used to suspend (checked) or resume (unchecked) the pulse counter port of the DLN-series adapter. This parameter is only available for DLN-2 adapter.
- **Timer** field - used to display the current timer value.
- **Counter** field - used to display the current pulse counter value.
- **Get Values** button - used to request the current values of timer and pulse counter which are displayed in the proper fields.
- **Timer** checkbox - used to check timer for reset.
- **Counter** checkbox - used to check pulse counter for reset.
- **Reset** button - used to reset timer or/and pulse counter.
- **Resolution** field - used to display the current resolution field.
- **Get Resolution** button - used to retrieve the current resolution value.

- **Mode** combobox - used to select the mode of pulse counter operation. There are 3 modes available: free run, time based, pulse based.
- **Limit** field - used to set pulse counter limit.
- **Set** Mode button - used to set pulse counter mode and limit.
- **Get** Mode button - used to get current values of pulse counter mode and limit.
- **Event Type** checkboxes - used to select event types to be monitored. There are 3 types of events: Overflow, Match, Repeat. You can check them all simultaneously or check nothing to stop events.
- **Repeat** field - used to set repeat value for events in ms.
- **Set** Event Config field -
- **Get** Event Config field -