Marvelmind Boxie API

Version 2022.08.07

www.marvelmind.com

Table of contents

1. Marvelmind Boxie API

Marvelmind Boxie API is a part of Marvelmind API library, used by Marvelmind Dashboard software and provides interface to user's software. API is coming as dynamic-link library (DLL) for MS Windows and shared library for Linux (x86 and ARM platforms).

Description of navigation related Marvelmind API library functions can be found in the document about protocols and interfaces.

In addition to the API library, the package includes C example software, which was used for testing of the API and includes calls of all API functions.

The example can be used as a basis for developing of a user's software and for porting API library interface (file 'marvelmind_boxie_api.c') to other programming languages.

Before using Marvelmind Boxie API you should build the map as described in the manual, check tracking of the Marvelmind Boxie robot and try to move it with arrows in the dashboard.

After that, close the dashboard and start using the API (modem should be connected to the PC via USB, it is used for communication with the Boxie robot).

Tested on:

1. MS Windows XP; CPU: Intel Core 2 Duo
2. MS Windows 10; CPU: Intel Core i5 Duo 3.1 GHz
3. Ubuntu 16.04; CPU: Intel Core i5 3.1 GHz
4. Raspbian (2018-11-13-raspbian-stretch-full); Platform: Raspberry Pi 3 Model B+

1.1. Installation for Windows

- Download Marvelmind Boxie API software package. Copy API dll and example software to directory that you will use for the program. Windows version of the example is coming with prebuilt executable file, you can immediately run ‘mm_boxie_api_example.exe’ from the ‘windows’ directory coming in API software package.

1.2. Installation for Linux

- Download Marvelmind Boxie API software package. Copy Dashboard API to directory that you will use for the program. Note the Linux version is provided for two hardware platforms: **x86** (most of laptops based on Intel or AMD CPU) and **arm** (for example, single-board computers like Raspberry PI)

- Copy library **libdashapi.so** corresponding to your platform to the directory **/usr/local/lib** by executing command **sudo cp libdashapi.so /usr/local/lib** in terminal opened in directory with **libdashapi.so**. After that, execute **sudo ldconfig** in terminal.

- May be, you will need to give rights for your user to access serial port by adding him to **dialout** group:
  - Execute in terminal: **sudo adduser $USER dialout**
  - Add to the directory **/etc/udev/rules.d** file “**99-tty.rules**” with following content:

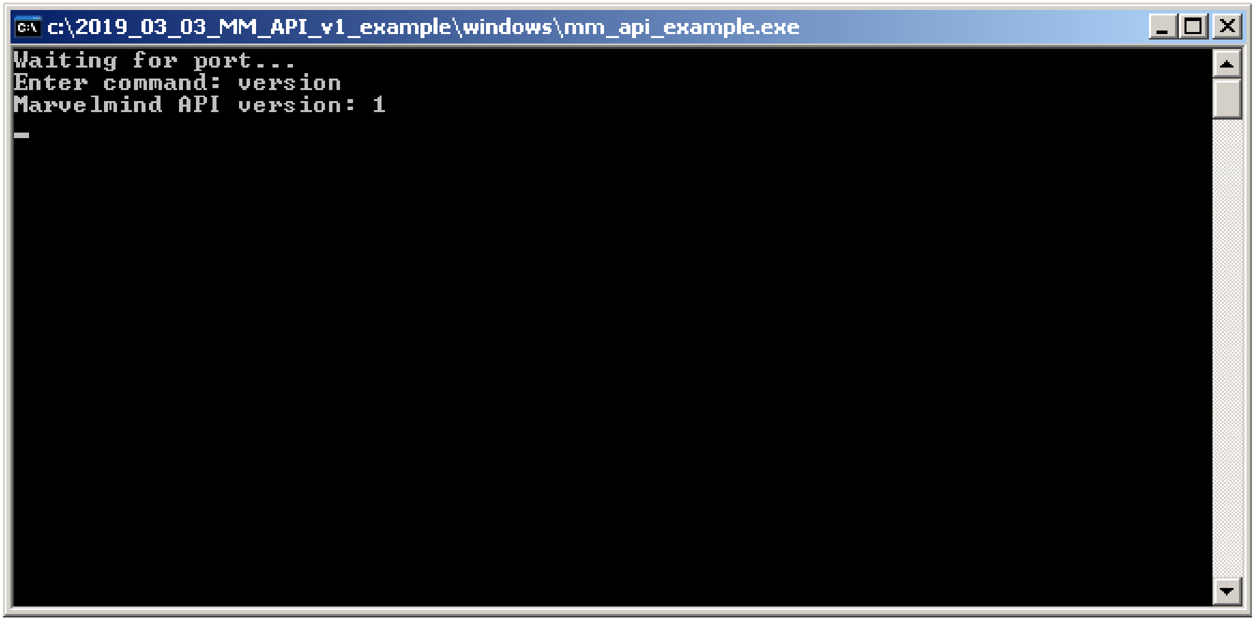
    ```
    #Marvelmind serial port rules
    KERNEL==”ttyACM0”,GROUP=”dialout”,MODE=”666”
    ```

- Build the example software – execute ‘make all’ in terminal opened in ‘source’ directory coming with the package

- Run the example by typing ‘./mm_api_example’ in terminal

1.3. Check connection to API

After running example software, press “space” button in terminal, type command ‘**version**’ and press enter. If the example software prints version of API, it can communicate with API library.



```
Waiting for port...
Enter command: version
Marvelmind API version: 1
_
```

1.4. Marvelmind Boxie API library description

API is coming as dynamic-link library (DLL) for MS Windows and shared library for Linux (x86 and ARM platforms). The library includes set of functions for monitoring and controlling Marvelmind system via modem connected to USB port of the computer. This section of document contains description of all these functions.

To provide more compatibility with different programming languages, most of complex data structures are passing via untyped pointers to memory. Functions description include offset of every data field in the memory pool. In the file ‘marvelmind_boxie_api.c’ from the example software you can see implementation of moving data between memory pools and fields in C structures.

Types of parameters in the description are shown in C syntax. Here is description of the types:

Type	Size (bytes)	Description
bool	1	Boolean type. Zero means false, non-zero means true
uint8_t	1	Unsigned integer value, 0…255
int8_t	1	Signed integer value in two’s complement format, -128…127
uint16_t	2	Unsigned integer value, 0…65535
int16_t	2	Signed integer value in two’s complement format, -32768…32767
uint32_t	4	Unsigned integer value, 0…4294967295
int32_t	4	Signed integer value in two’s complement format, -2147483648…2147483647
void *	4/8	Memory pointer (address in memory). 4 bytes for 32-bit platforms, 8 bytes for 64-bit platforms.

Each function description includes set of API versions where this function is available. New API versions will support more functions for new features in Marvelmind system. Now not all features available in Dashboard are available via API, so if you need more API functions please ask to info@marvelmind.com.

List of supported functions:

Function	API versions	License needed
Get version of Marvelmind API library	V1+	none
Get last error	V6+	none
Try to open serial port	V1+	none
Try to open serial port by given name	V2+	none
Close serial port	V1+	none
Get version and CPU ID of Marvelmind device	V1+	none
Get list of devices	V1+	none
Get latest beacons location data	V1+	none
Get latest beacons location data (with angle)	V3+	none
Immediate robot movement (like arrows in dashboard)	V7+	none
Stop the robot	V7+	none
Send robot movement program item (waypoint etc)	V7+	none
Send command to the robot	V7+	none
Read Boxie telemetry data	V7+	none
Read Boxie lidars data	V7+	none
Read Boxie location data	V7+	none

1.4.1. Get version of Marvelmind API library

Reads version of the API library. Required to ensure the needed functions are available in this version of library.

Function name:	**mm_api_version**
Declaration in C:	bool mm_api_version(void *pdata);
Available for API versions:	V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer.

Type	Description
uint32_t	Version of API library

1.4.2. Get last error

Reads status of last operation with API library to differ causes of the error.

Function name:	**mm_get_last_error**
Declaration in C:	bool mm_get_last_error(void *pdata);
Available for API versions:	V6+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer.

Type	Description
uint32_t	Status of last operation: 0: operation successfully executed 1: communication error 2: error opening serial port 3: license is required

1.4.3. Open serial port

Opens port where Marvelmind device (modem or beacon) is connected via USB (virtual serial port). You don't need to specify serial port name, because the API searching all serial ports and checks whether it corresponds to Marvelmind device or no.

Function name:	**mm_open_port**
Declaration in C:	bool mm_open_port ();
Available for API versions:	V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, port is opened false – error in execution

Parameters: none

1.4.4. Open serial port by given name

Opens port where Marvelmind device (modem or beacon) is connected via USB (virtual serial port). Function tries to open port with specified name.

Function name:	**mm_open_port_by_name**
Declaration in C:	bool mm_open_port_by_name();
Available for API versions:	V2+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, port is opened false – error in execution

Parameters:

Type	Description
void *	Pointer to serial portname – sequence of ASCII characters terminated by zero (ASCIIZ)

1.4.5. Close serial port

Closes port, if it was previously opened by **mm_open_port** function.

Function name:	**mm_close_port**
Declaration in C:	bool mm_close_port ();
Available for API versions:	V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, port is closed false – error in execution

Parameters: none

1.4.6. Get version and CPU ID of Marvelmind device

Reads version and CPU ID. Version includes information about firmware version and type of device hardware. CPU ID is the unique ID of the device item.

Function name: **mm_get_device_version_and_id**
Declaration in C: bool mm_get_device_version_and_id (uint8_t address, void *pdata);
Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, version and CPU ID data retrieved false – error in execution

Parameters:

Type	Description
uint8_t	Address of Marvelmind device (1…254)
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Major version of firmware (example: “6”, for version V6.07a)
uint8_t	Minor version of firmware (example: “7”, for version V6.07a)
uint8_t	Second minor version of firmware (example: “1”, for version V6.07a)
uint8_t	Device type ID (see appendix).
uint8_t	Firmware options (TBD).
uint32_t	CPU ID. Printing this value as hexadecimal gives CPU ID in form shown in dashboard and on the stickers on devices.

1.4.7. Get list of devices

Reads list of Marvelmind devices known to modem. The list includes list of all devices connected by radio to modem’s network, including sleeping devices.

Function name:	**mm_get_devices_list**
Declaration in C:	bool mm_get_devices_list (void *pdata);
Available for API versions:	V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, list of devices is retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Number of following devices in the list (N)
N*9 bytes	Sequence of N devices structures, described in next table

Structure of each device in the list:

Type	Description
uint8_t	Address of device
bool	true = duplicated address - more than 1 device with same address was found false = not duplicated address

bool	true = device is sleeping
	false = device not sleeping

uint8_t	Major version of firmware (example: “6”, for version V6.07a)
uint8_t	Minor version of firmware (example: “7”, for version V6.07a)
uint8_t	Second minor version of firmware (example: “1”, for version V6.07a)
uint8_t	Device type ID (see appendix).
uint8_t	Firmware options (TBD).
uint8_t	Flags: Bit 0: 1 – device connection complete – device has confirmed connection 0 – waiting for confirmation from device (like ‘Connecting…’ in dashboard). Bit 1…7 - TBD

1.4.8. Get latest location data

Reads latest updated beacons coordinates pack from modem. Also reads user payload data if available.

Function name:	**mm_get_last_locations**
Declaration in C:	bool mm_get_last_locations(void *pdata);
Available for API versions:	V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, location data was retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
18*6 bytes	6 18-byte data structures of last updated coordinates, see table below
bool	true – new raw distances are available to read
5 bytes	TBD
uint8_t	User payload data size (M)
M bytes	User payload data

Structure of each location data item:

Type	Description
uint8_t	Address of device (1…254) 0 - this data item is not filled
uint8_t	Head index (TBD)
int32_t	X coordinate, mm
int32_t	Y coordinate, mm
int32_t	Z coordinate, mm
uint8_t	Status flags (TBD)
uint8_t	Quality of positioning, 0…100%
uint8_t	TBD
uint8_t	TBD

1.4.9. Get latest beacons location data (with angle)

Reads latest updated beacons coordinates pack from modem (with angle for paired beacons). Also reads user payload data if available.

Function name:	**mm_get_last_locations2**
Declaration in C:	bool mm_get_last_locations2(void *pdata);
Available for API versions:	V3+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, location data was retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
20*6 bytes	6 20-byte data structures of last updated coordinates, see table below
bool	true – new raw distances are available to read
5 bytes	TBD
uint8_t	User payload data size (M)
M bytes	User payload data

Structure of each location data item:

Type	Description
uint8_t	Address of device (1…254) 0 - this data item is not filled
uint8_t	Head index (TBD)
int32_t	X coordinate, mm
int32_t	Y coordinate, mm
int32_t	Z coordinate, mm
uint8_t	Status flags (TBD)
uint8_t	Quality of positioning, 0…100%
uint8_t	TBD
uint8_t	TBD
uint16_t	Bit 0…11 – angle of rotation in 1/10 degree (if paired beacons feature is enabled) Bit 12 – 1 = angle not available Bit 13…15 - reserved

1.4.10. Command for immediate robot movement (like arrows in dashboard)

Send command for immediate robot movement.

Function name: **mm_robot_movement**

Declaration in C: bool mm_robot_movement (uint8_t address, void *pdata);

Available for API versions: V7+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address – address of the robot
void *	pdata - pointer to data to write

Structure of data by pointer:

Type	Description
uint8_t	**move_type** – type of the robot movement: 0 – forward 1 – backward 2 – rotate clockwise 3 – rotate counterclockwise
64 bytes	Reserved (0)

1.4.11. Command for robot stopping

Send command for stopping robot movement.

Function name: **mm_robot_stop**

Declaration in C: bool mm_robot_stop (uint8_t address);

Available for API versions: V7+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address – address of the robot

1.4.12. Send robot movement program item

Send movement program item to the robot.

Function name: **mm_set_robot_program_item**

Declaration in C: bool mm_set_robot_program_item (uint8_t address, void *pdata);

Available for API versions: V7+

License required: none
Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address – address of the robot
void *	pdata - pointer to data to write

Structure of data by pointer:

Type	Description
uint8_t	**Item_index** – index of this program item (starting from 0)
uint8_t	**Items_number** – total number of program items
uint8_t	**OpCode** – type of the action: 0 – move forward 1 – move backward 2 – rotate clockwise 3 – rotate counterclockwise 6 – move to specified waypoint
int16_t	**Param1** – first movement parameter: If OpCode is 0 or 1 – distance of movement, cm If OpCode is 2 or 3 – angle of rotation, degrees If Opcode is 6 – X coordinate of waypoint, cm
int16_t	**Param2** – second movement parameter: If Opcode is 6 – Y coordinate of waypoint, cm
int16_t	**Param3** – third movement parameter: If Opcode is 6 – Z coordinate of waypoint, cm (not used for Boxie)

1.4.13. Send command to the robot

Send command to the robot.

Function name: **mm_set_robot_command**

Declaration in C: bool mm_set_robot_command (uint8_t address, void *pdata);

Available for API versions: V7+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address – address of the robot
void *	pdata - pointer to data to write

Structure of data by pointer:

Type	Description
uint8_t	**Command_id** – command type: 4 – pause executing movement program 8 – continue executing movement program 9 – start executing movement program from the first item
int16_t	**Param1** – first parameter (not used)
int16_t	**Param2** – second parameter (not used)
int16_t	**Param3** – third parameter (not used)

1.4.14. Read Boxie telemetry data

Reads telemetry data from Boxie.

Function name: **mm_get_boxie_telemetry**

Declaration in C: bool mm_get_boxie_telemetry (uint8_t address, void *pdata);

Available for API versions: V7+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address – address of the robot
void *	pdata - pointer to data to write

Structure of data by pointer:

Type	Description
int16_t	**Vbat** – Boxie battery voltage, mV
int16_t	**Current** – Boxie supply current, mA
int16_t	**Pwr_left** – power on the left wheel, percents
int16_t	**Pwr_right** – power on the right wheel, percents
int16_t	**Odo_left** – speed of the left wheel by odometry, mm/s
int16_t	**Odo_right** – speed of the right wheel by odometry, mm/s
64 bytes	**Reserved**

1.4.15. Read Boxie lidars data

Reads lidars data from Boxie.

Function name: **mm_get_boxie_lidars**

Declaration in C: bool mm_get_boxie_lidars (uint8_t address, void *pdata);

Available for API versions: V7+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address – address of the robot
void *	pdata - pointer to data to write

Structure of data by pointer:

Type	Description
48 bytes	Data from 12 Boxie lidars, 4 bytes per lidar
64 bytes	Reserved

Structure of data for each lidar:

Type	Description
uint16_t	**Range** - range measured by lidar, mm
uint8_t	**Status** – status of the measurement: 0 – range is measured Other values – range is not measured, **Range** field is nor relevant
uint8_t	**Reserved**

1.4.16. Read Boxie location data

Reads location data from Boxie.

Function name: **mm_get_boxie_location**

Declaration in C: bool mm_get_boxie_location (uint8_t address, void *pdata);

Available for API versions: V7+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address – address of the robot
void *	pdata - pointer to data to write

Structure of data by pointer:

Type	Description
int32_t	**x_mm** – X coordinate, mm
int32_t	**y_mm** – Y coordinate, mm
int32_t	**z_mm** – Z coordinate, mm (currently not used)
int16_t	**yaw** – yaw angle, degrees
64 bytes	Reserved

4.5.4. Description of C example for Marvelmind Boxie API

C example is used for testing of Marvelmind Boxie API and can be used as basis for building of user application.

The C example is the console application. It was tested on following platforms:

- CPU: Intel Core 2 Duo, OS: MS Windows XP;
- MS Windows 10; CPU: Intel Core i5 Duo 3.1 GHz
- CPU: Intel Core i5, OS: Linux Ubuntu 16.04;
- Raspberry Pi 3 Model B+, OS: Raspbian (2018-11-13-raspbian-stretch-full)

On the Windows platform the example was built with CodeBlocks IDE and so the example includes CodeBlocks project file.

On the Linux platforms, the example was built with using make utility and so the example includes makefile for this.

The example includes following modules:

File name	Description
main.c	Module with main () function. Calls of functions of example and implements simple command line interface.
marvelmind_example.c marvelmind_example.h	marvelmindStart() – initialization of the example marvelmindFinish() – called after finishing work with API marvelmindCycle() – frequently called from main loop Also, module includes several function for processing commands entered by user.
marvelmind_boxie_api.c marvelmind_boxie_api.h	marvelmindAPILoad() – loads API library marvelmindAPIFree() – frees memory used by API library All functions of communication with API library.
marvelmind_devices.c marvelmind_devices.h	Supports list of beacons retrieved from modem by calling 'get devices list' command
marvelmind_pos.c marvelmind_pos.h	Reads latest beacons location data. Updates these data in the devices list.
marvelmind_utils.c marvelmind_utils.h	Some helper functions used by other modules.

How the example works:

1. Try to open serial port until success
2. When port is opened, the program reads version of device connected via USB. If this is modem, the program continues to execute next steps
3. When connected to modem, the program reads the devices list with 1 Hz rate. The devices list is compared with currently stored in marvelmind_devices.c module and the list in marvelmind_devices.c is updated, if any changes are detected. All changes are printed in console
4. When connected to modem, the program reads the latest location data with 20 Hz rate.
5. If the program can't get latest location data for 10 times, it closes the port and returns to step 1 – tries to open the port again. Reopening of the port is needed for cases when modem was disconnected and connected back to USB

6. If user press ‘**space**’ button, the program shows ‘Enter command: ‘ message and waits for user command. Most of API functions are called by user command, see below for details

User commands:

If user press ‘**space**’ button when program is running, the program shows message ‘**Enter command:** ‘. User should type command on keyboard and press **enter**.

The table below contains format of all user commands:

Commands group	Description
API version	Format of command: **version** Action: Prints version of API library
Exit from program	Format of command: **quit** Action: Finishes program execution
Boxie commands	Format of command: **boxie move <address> <move_type>** Action: Execute command to move the robot with specified movement type. Example: **boxie move 110 0** - move forward Boxie robot n110
	Format of command: **boxie stop <address>** Action: Execute command to stop the robot. Example: **boxie stop 110** - stop Boxie robot n110
	Format of command: **boxie start <address>** Action: Execute command to start the robot movement by the specified program. Example: **boxie start 110** - start execution of movement program for Boxie robot n110
	Format of command: **boxie pause <address>** Action: Execute command to pause the robot movement by the specified program. Example: **boxie pause 110** - pause the Boxie robot n110

	Format of command: **boxie continue <address>** Action: Execute command to continue the robot previously paused movement by the specified program. Example: **boxie continue 110** - continue movement the Boxie robot n110
	Format of command: **boxie program <address>** Action: Starts creating the movement program. Example: **boxie program 110** - start creating program for the Boxie robot n110 After this a new command prompt is appeared: **Robot program command:** Following commands are supported: **end** – finish creating program and send it to the robot **forward <distance in cm>** - forward movement **backward <distance in cm>** - backward movement **clockwise <angle in degrees>** - rotate clockwise **counterclockwise <angle in degrees>** - rotate counterclockwise **waypoint <x, cm> <y, cm>** - move to specified waypoint X, Y
	Format of command: **boxie tele <address>** Action: Execute command to read telemetry data and print the data Example: **boxie tele 110** - read and print telemetry of the Boxie robot n110
	Format of command: **boxie lidars <address>** Action: Execute command to read lidars data and print the data Example: **boxie lidars 110** - read and print lidars data of the Boxie robot n110
	Format of command: **boxie location <address>** Action: Execute command to read location data and print the data Example: **boxie location 110** - read and print location data of the Boxie robot n110