Marvelmind API manual

Version 2020.04.02

Product overview

Marvelmind API library is 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).

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_api.c') to other programming languages.

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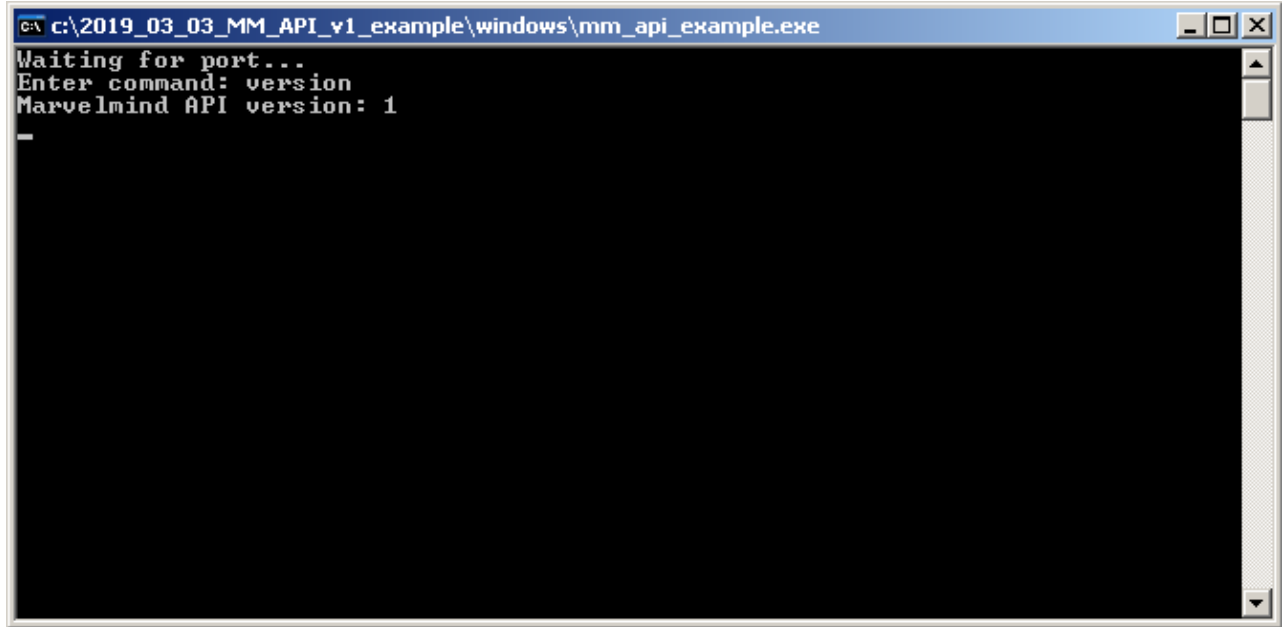
Installation for Windows

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

Installation for Linux

- Download Marvelmind 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

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.


```
c:\2019_03_03_MM_API_v1_example\windows\mm_api_example.exe
Waiting for port...
Enter command: version
Marvelmind API version: 1
```

Marvelmind
robotics

1. Marvelmind 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_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
Get version of Marvelmind API library	V1+
Try to open serial port	V1+
Try to open serial port by given name	V2+
Close serial port	V1+
Get version and CPU ID of Marvelmind device	V1+
Get list of devices	V1+
Wake device	V1+
Send device to sleep	V1+
Get telemetry data from beacon	V1+
Get latest location data	V1+
Get latest location data (with angle)	V3+
Set location of the beacon	V3+
Get latest raw distances data	V1+
Get location update rate setting	V1+
Set location update rate setting	V1+
Add submap	V1+
Delete submap	V1+
Freeze submap	V1+
Unfreeze submap	V1+
Get submap settings	V1+
Set submap settings	V1+
Get ultrasonic settings of the beacon	V1+
Set ultrasonic settings of the beacon	V1+
Erase map	V1+
Reset device to default settings	V1+
Connect beacons to axes	V2+
Read modem's configuration memory dump	V3+
Write modem's configuration memory dump	V3+
Get temperature of air setting from modem	V3+
Set temperature of air setting in modem	V3+
Software reset of the device	V3+
Check if the device type is modem	V1+
Check if the device type is beacon	V1+
Check if the device type is hedgehog	V1+

1.1. 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+

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

Marvelmind
robotics

1.2. 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+

Returned value:

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

Parameters: none

1.3. 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+

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. 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+

Returned value:

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

Parameters: none

1.5.   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+

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.

Marvelmınd
robotics

1.6. 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+

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

1.7. Wake device

Sends command to wake specified device. If wake command was sent and such device is exist, the device will connect to modem in several seconds and will appear in devices list.

Function name:	**mm_wake_device**
Declaration in C:	bool mm_wake_device (uint8_t address);
Available for API versions:	V1+

Returned value:

Type	Description
bool	true – function successfully executed, wake command was sent
	false – error in execution

Parameters:

Type	Description
uint8_t	1…254 - address of Marvelmind device to wake
	0 – wake all devices

1.8. Send device to sleep

Send to sleep existing device.

Function name:	**mm_send_to_sleep_device**
Declaration in C:	bool mm_send_to_sleep_device (uint8_t address);
Available for API versions:	V1+

Returned value:

Type	Description
bool	true – function successfully executed, sleep command was sent
	false – error in execution

Parameters:

Type	Description
uint8_t	1…254 - address of Marvelmind device to sleep
	0 – send to sleep all devices

Marvelmind
robotics

1.9.   Get telemetry data from beacon

Reads telemetry data of Marvelmind beacon.

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

Returned value:

Type	Description
bool	true – function successfully executed, telemetry is retrieved false – error in execution

Parameters:

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

Structure of data returned via pointer:

Type	Description
uint32_t	Working time of the beacon, seconds (time from reset or waking up).
int8_t	RSSI, dBm – radio signal strength
int8_t	Measured temperature, °C
uint16_t	Supply voltage, mV
16 bytes	Reserved (0)

1.10. Get latest location data

Reads latest updated 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+

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

1.11. Get latest location data (with angle)

Reads latest updated 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+

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

1.12. Set location of the beacon

Manual setup of location of the specified beacon.

Function name:	**mm_set_beacon_location**
Declaration in C:	bool mm_set_beacon_location (uint8_t address, void *pdata);
Available for API versions:	V3+

Returned value:

Type	Description
bool	true – function successfully executed, location is updated
	false – error in execution

Parameters:

Type	Description
uint8_t	Address of the beacon
void *	Pointer to buffer with location data

Structure of data by pointer (should be filled before function call):

Type	Description
int32_t	New X coordinate of the beacon, mm
int32_t	New Y coordinate of the beacon, mm
int32_t	New Z coordinate of the beacon, mm

Marvelmind
robotics

1.13.   Get latest raw distances data

Reads latest updated raw distances pack from modem.

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

Returned value:

Type	Description
bool	true – function successfully executed, raw distances 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
uint8_t	Number of raw distances data items (N). Maximum number of raw distances per request is 16: N<=16
9*N bytes	N 9-byte data structures of last raw distances, see table below

Structure of each raw distance data item:

Type	Description
uint8_t	Address of ultrasonic RX device (1…254) 0 - this data item is not filled
uint8_t	RX Head index (TBD)
uint8_t	Address of ultrasonic TX device (1…254) 0 - this data item is not filled
uint8_t	TX Head index (TBD)
uint32_t	Distance from TX device to RX device, mm
uint8_t	TBD

1.14. Get location update rate setting

Reads location update rate setting from modem.

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

Returned value:

Type	Description
bool	true – function successfully executed, update rate was retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint32_t	Location update rate setting in mHz. So 1000 is returned for 1 Hz, 16000 for 16 Hz, 50 for 0.05 Hz mode.

1.15. Set location update rate setting

Writes location update rate setting to modem.

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

Returned value:

Type	Description
bool	true – function successfully executed, update rate was changed false – error in execution

Parameters:

Type	Description
void *	Pointer to data

Structure of data by pointer (should be filled before function call):

Type	Description
uint32_t	Location update rate setting in mHz. So 1000 is returned for 1 Hz, 16000 for 16 Hz, 50 for 0.05 Hz mode. The system will use most close to specified update rate from the series: 0.05 Hz, 0.1 Hz, 0.2 Hz, 0.5Hz, 1 Hz, 2 Hz, 4 Hz, 8 Hz, 12 Hz, 16 Hz, 16+Hz.

Marvelmind
robotics

1.16. Add submap

Adds new submap.

Function name: **mm_add_submap**
Declaration in C: bool   mm_add_submap (uint8_t submapId);
Available for API versions: V1+

Returned value:

Type	Description
bool	true – function successfully executed, submap was added false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to add (0…254)

1.17. Delete submap

Delete existing submap.

Function name: **mm_delete_submap**
Declaration in C: bool   mm_delete_submap (uint8_t submapId);
Available for API versions: V1+

Returned value:

Type	Description
bool	true – function successfully executed, submap was removed false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to delete (0…254)

Marvelmind
robotics

1.18. Freeze submap

Freezes submap.

Function name: **mm_freeze_submap**
Declaration in C: bool   mm_freeze_submap (uint8_t submapId);
Available for API versions: V1+

Returned value:

Type	Description
bool	true – function successfully executed, submap is frozen false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to freeze (0…254)

1.19. Unfreeze submap

Unfreezes submap.

Function name: **mm_unfreeze_submap**
Declaration in C: bool   mm_unfreeze_submap (uint8_t submapId);
Available for API versions: V1+

Returned value:

Type	Description
bool	true – function successfully executed, submap is unfrozen false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to unfreeze (0…254)

1.20. Get submap settings

Reads submap settings from modem.

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

Returned value:

Type	Description
bool	true – function successfully executed, submap settings were retrieved false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID (0…254)
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Starting beacon trilateration
uint8_t	Starting set of beacons, beacon 1
uint8_t	Starting set of beacons, beacon 2
uint8_t	Starting set of beacons, beacon 3
uint8_t	Starting set of beacons, beacon 4
bool	true = 3D navigation enabled
bool	true = Submap is used only for Z coordinate
bool	true = manual limitation distance false = auto limitation distance
uint8_t	Maximum distance, meters (for manual limitation distances)
int16_t	Submap X shift, cm
int16_t	Submap Y shift, cm
int16_t	Submap Z shift, cm
uint16_t	Submap rotation, centidegrees
int16_t	Plane rotation quaternion, W (quaternion is normalized to 10000)
int16_t	Plane rotation quaternion, X
int16_t	Plane rotation quaternion, Y
int16_t	Plane rotation quaternion, Z
int16_t	Service zone thickness, cm
int16_t	Hedges height in 2D mode
bool	true = submap is frozen
bool	true = submap is locked
bool	true = stationary beacons are higher than mobile
bool	true = submap is mirrored
4 bytes	List of addresses of beacons in submap (0 = none)
8 bytes	List of ID's of nearby submaps (255 = none)
uint8_t	Number of service zone polygon points (P)
P*4 bytes	List of service zone polygon points structures (see below)

Structure of service zone polygon point:

Type	Description
int16_t	X, cm
int16_t	Y, cm

1.21. Set submap settings

Writes submap settings to modem.

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

Returned value:

Type	Description
bool	true – function successfully executed, submap settings were changed false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID (0…254)
void *	Pointer to data to be written (see 'get submap settings' function).

1.22. Get ultrasonic settings of the beacon

Reads ultrasonic settings from specified beacon.

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

Returned value:

Type	Description
bool	true – function successfully executed, ultrasonic settings were retrieved false – error in execution

Parameters:

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

Structure of data returned via pointer:

Type	Description
uint16_t	Frequency of ultrasound TX (not relevant for DSP RX-only beacons)
uint8_t	Number of TX periods (not relevant for DSP RX-only beacons)
bool	true= use AGC for RX false = manual gain for RX
uint16_t	Manual gain value (0…4000)
bool	true = Sensor RX1 is enabled in normal mode
bool	true = Sensor RX2 is enabled in normal mode
bool	true = Sensor RX3 is enabled in normal mode
bool	true = Sensor RX4 is enabled in normal mode
bool	true = Sensor RX5 is enabled in normal mode
bool	true = Sensor RX1 is enabled in frozen mode
bool	true = Sensor RX2 is enabled in frozen mode
bool	true = Sensor RX3 is enabled in frozen mode
bool	true = Sensor RX4 is enabled in frozen mode
bool	true = Sensor RX5 is enabled in frozen mode
uint8_t	Index of DSP RX filter (relevant only for DSP beacons) 0 = 19 kHz 1 = 25 kHz 2 = 31 kHz 3 = 37 kHz 4 = 45 kHz 5 = 56 kHz

Marvelmind robotics

1.23. Set ultrasonic settings of the beacon

Write ultrasonic settings to specified beacon.

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

Returned value:

Type	Description
bool	true – function successfully executed, ultrasonic settings were changed false – error in execution

Parameters:

Type	Description
uint8_t	Address of the beacon (1…254)
void *	Pointer to data to be written (see '<u>get ultrasonic settings</u>' function).

Marvelmind
robotics

1.24. Erase map

Erase map in modem – remove all submaps (except submap 0), reset submap 0 to initial state, remove all connected beacons from network.

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

Returned value:

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

Parameters: none

1.25. Reset device to default settings

Reset device to default settings (radio, ultrasonic etc).

Function name: **mm_set_default_settings**
Declaration in C: bool mm_set_default_settings (uint8_t address);
Available for API versions: V1+

Returned value:

Type	Description
bool	true – function successfully executed, device was reset to default settings
	false – error in execution

Parameters:

Type	Description
uint8_t	Address of the device (1…254)
	255 – reset to default the device connected via USB

Marvelmind
robotics

1.26. Connect beacons to axes

Shift map so selected beacons will be on axes.

Function name: **mm_beacons_to_axes**
Declaration in C: bool mm_beacons_to_axes (void *pdata);

Available for API versions: V2+

Returned value:

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

Structure of data by pointer:

Type	Description
uint8_t	address_0 – address of beacon which should be in the center (X=0, Y=0)
uint8_t	address_x – address of beacon which should be along X axis (Y= 0)
uint8_t	address_y – address of beacon which should be in positive direction of Y (Y>0)

1.27. Read dump of modem's configuration memory

Reads dump of modem's configuration memory. Allows saving modem's settings and stored map.

Function name: **mm_read_flash_dump**
Declaration in C: bool mm_read_flash_dump(uint32_t offset, uint32_t size, void *pdata);

Available for API versions: V3+

Returned value:

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

Parameters:

Type	Description
uint32_t	offset – offset from start of configuration memory, bytes
uint32_t	size – size of data to read, bytes
void *	pdata – pointer to user's buffer for receiving data

1.28. Write dump of modem's configuration memory

Write data dump to modem's configuration memory. Allows to restore modem's settings and map.

Function name: **mm_write_flash_dump**
Declaration in C: bool mm_write_flash_dump(uint32_t offset, uint32_t size, void *pdata);

Available for API versions: V3+

Returned value:

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

Parameters:

Type	Description
uint32_t	offset – offset from start of configuration memory, bytes For correct operation offset should be aligned to 4096 bytes page (value 0, 4096, 8192 and so on).
uint32_t	size – size of data to write, bytes
void *	pdata – pointer to user's buffer with data

Note: After writing the configuration, software reset of the modem (**mm_reset_device(255)**) is recommended to apply new settings and prevent overwriting them.

Marvelmind robotics

1.29. Restart (soft reset) of the device

Executes software reset for specified device.

Function name: **mm_reset_device**
Declaration in C: bool mm_reset_device (uint8_t address);

Available for API versions: V3+

Returned value:

Type	Description
bool	true – function successfully executed, device is resetting false – error in execution

Parameters:

Type	Description
uint8_t	Address of the device (1…254) 255 –software reset for device connected via USB

Marvelmind
robotics

1.30. Read temperature of air setting from modem

Reads temperature of air setting (in Celsius degrees) from modem.

Function name: **mm_get_air_temperature**
Declaration in C: bool mm_get_air_temperature (void *pdata);

Available for API versions: V3+

Returned value:

Type	Description
bool	true – function successfully executed, temperature is returned false – error in execution

Structure of data returned via pdata pointer:

Type	Description
int8_t	Temperature of air, Celsius degrees

1.31. Write temperature of air setting to modem

Setup temperature of air setting (in Celsius degrees) in modem.

Function name: **mm_set_air_temperature**
Declaration in C: bool mm_set_air_temperature (void *pdata);

Available for API versions: V3+

Returned value:

Type	Description
bool	true – function successfully executed, temperature was written false – error in execution

Structure of data which user should supply via pdata pointer:

Type	Description
int8_t	Temperature of air, Celsius degrees

1.32. Check whether device type is modem

Checks whether the specified device type corresponds to modem.

Function name:	**mm_device_is_modem**
Declaration in C:	bool mm_device_is_modem (uint8_t deviceType);
Available for API versions:	V1+

Returned value:

Type	Description
bool	true – specified device type corresponds to modem

Parameters:

Type	Description
uint8_t	Device type to check

1.33. Check whether device type is beacon

Checks whether the specified device type corresponds to beacon.

Function name:	**mm_device_is_beacon**
Declaration in C:	bool mm_device_is_beacon (uint8_t deviceType);
Available for API versions:	V1+

Returned value:

Type	Description
bool	true – specified device type corresponds to beacon

Parameters:

Type	Description
uint8_t	Device type to check

1.34. Check whether device type is hedgehog

Checks whether the specified device type corresponds to hedgehog.

Function name:	**mm_device_is_hedgehog**
Declaration in C:	bool mm_device_is_hedgehog (uint8_t deviceType);
Available for API versions:	V1+

Returned value:

Type	Description
bool	true – specified device type corresponds to hedgehog

Parameters:

Type	Description
uint8_t	Device type to check

2. Description of C example for Marvelmind API

C example is used for testing of Marvelmind 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;
- 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_api.c marvelmind_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. Each beacon includes data about its location and distances to other beacons.
marvelmind_pos.c marvelmind_pos.h	Reads latest location data and latest raw distances. 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. If the flag of new raw distances data is set, the program reads latest raw distances. The program compares locations and distances with data in devices list in marvelmind_devices.c and updates the data if they are changed. All changed data are printed in console

5. If the program can't get latest location data for 10 times, it <u>closes the port</u> 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
Sleep/wake	Format of command: **wake <address>** Action: Execute <u>wake</u> command. Examples: **wake 5** - send command to wake device 5 **wake 0** - send command to wake all devices
	Format of command: **sleep <address>** Action: Execute <u>sending to sleep</u> command. Examples: **sleep 5** - send to sleep device 5 **sleep 0** - send to sleep all devices
Default	Format of command: **default <address>** Action: Execute <u>reset to default settings</u> command. Examples: **default 5** - set default settings for device 5
Read telemetry	Format of command: **tele <address>** Action: Reads and prints <u>telemetry</u> data of beacon.

Marvelmind robotics

	Examples:
	tele 5 - read and print telemetry of beacon 5
Submap commands	Format of command:
	submap add <submapId>
	Action:
	Execute command to <u>add submap</u> with specified submap ID.
	Example:
	submap add 1 - add submap 1
	Format of command:
	submap delete <submapId>
	Action:
	Execute command to <u>delete submap</u> with specified submap ID.
	Example:
	submap delete 1 - delete submap 1
	Format of command:
	submap freeze <submapId>
	Action:
	Execute command to <u>freeze submap</u> with specified submap ID.
	Example:
	submap freeze 0 - freeze submap 0
	Format of command:
	submap unfreeze <submapId>
	Action:
	Execute command to <u>unfreeze submap</u> with specified submap ID.
	Example:
	submap unfreeze 0 - unfreeze submap 0
	Format of command:
	submap get <submapId>
	Action:
	Execute command to <u>get settings of submap</u> with specified submap ID.
	Example:
	submap get 0 - get and print settings of submap 0
	Format of command:
	submap testset <submapId>
	Action:
	Execute command to <u>set settings of submap</u> with specified submap ID. The program writes some predefined settings for testing of the command; please see the example code.
	Example:
	submap testset 0 - modify settings of submap 0
Map commands	Format of command:
	map erase
	Action:

	Execute <u>erase map</u> command. Example: **map erase** - erase map in modem
Update rate commands	Format of command: **rate get** Action: Execute <u>reading update rate</u> setting command. Example: **rate get** - read and print update rate setting
	Format of command: **rate set <value>** Action: Execute <u>change update rate</u> setting command. Value is given in Hz Example: **rate set 0.5** - set update rate 0.5 Hz
Ultrasound commands	Format of command: **usound get <address>** Action: Execute <u>reading ultrasonic settings</u> for specified beacon. Example: **usound get 5** - read and print ultrasound settings of beacon 5
	Format of command: **usound testset <address>** Action: Execute <u>writing ultrasonic settings</u> for specified beacon. The program writes some predefined settings for testing of the command; please see the example code. Example: **usound testset 5** - modify ultrasound settings of beacon 5
Connect to axes command	Format of command: **axes <address_0> <address_x> <address_y>** Action: Execute <u>connect beacons to axes</u> command.. Example: **axes 3 4 5** - set beacon 3 to X=0, Y=0; beacon 4 along X (Y=0) and beacon 5 above X (Y>0)

Marvelmind
robotics

Read configuration memory dump from modem	Format of command: **read_dump <offset> <size>** Action: Execute read dump of modem configuration memory command. Example: **read_dump 0 1000** - read first 1000 bytes from beginning of configuration memory
Write configuration memory test dump to modem	Format of command: **write_dump_test <offset> <size>** Action: Execute write dump of modem configuration memory command. Example: **write_dump_test 0 1000** - fills first 1000 bytes from beginning of configuration memory by test pattern
Software reset of device	Format of command: **reset <address>** Action: Execute software reset command. Example: **reset 255** - executes software reset for device connected via USB
Temperature of air commands	Format of command: **temperature get** Action: Execute reading temperature of air setting from modem Example: **temperature get** read and print ultrasound temperature of air setting
	Format of command: **temperature set <value>** Action: Execute writing temperature of air setting to modem Example: **temperature set 30** setup temperature of air setting to 30 Celsius degrees
Set location of beacon	Format of command: **setloc <address> <X > <Y > <Z>** Action: Execute set location of the beacon command. X,Y,Z are coordinates in meters. Example: **setloc 12 1.51 3.45 2.0** - sets location of beacon 12 to X= 1.51 m, Y= 3.45 m, Z= 2.0 m

Marvelmind robotics

Appendix 1. Device types

Here is the list of 'Device type ID' values for specific devices:

Device type ID	Device description
22	Beacon HW V4.5
23	Beacon HW V4.5 (hedgehog mode)
24	Modem HW V4.9
30	Beacon HW V4.9
31	Beacon HW V4.9 (hedgehog mode)
32	Beacon Mini-RX
36	Beacon Mini-TX
37	Beacon-TX-IP67
41	Beacon industrial-RX
42	Super-Beacon
43	Super-Beacon (hedgehog mode)
44	Super-Beacon industrial
45	Super-Beacon industrial (hedgehog mode)
46	Super-Modem

You can get device type id from devices list and reading device version commands.

Marvelmind
robotics