Blue Creation

Ministak FAQ

REF: MIN-FAQ-001

Author: Genis Aguilar Lopez

This document provides a guide to writing an application that uses the Ministak library. It covers the first steps and the most common issues developers go through when writing their applications. Reader is expected to have some degree of knowledge on Bluetooth as the document refers to a number of concepts explained in the Bluetooth Core specification. You are encouraged to report any errors and to suggest any new contents worth being added.

Contents

How do I initialize Ministak?

There are some functions you must call before Ministak gets initialized:

- hal_tick_init: initializes the platform-dependant timers that Ministak will use.
- pskeys_init: sets the PSKEYS and the patches that Ministak will load to the controller.
- btstack_init: actually starts initializing Ministak. You'll need to pass it a device configuration, a remote device database and an structure with a set of callbacks Ministak will use to report your applications about any events that may occur.
- rfcomm_init: initializes the rfcomm communications. In the unlikely case you won't use any of the profiles supported in Ministak, you can omit calling this function.
- sdp_init: initializes the SDP module. In the unlikely case you won't use SDP, you can omit calling this function.

You will need to initialize any specific profile you are going to use as well. Each one of them has an initialization method, usually named after the profile and ending with the suffix _init.

What is the device configuration?

When you call btstack_init you'll need to pass it a structure that contains:

- The name of your device
- If using BLE, the short name of your device. You can set it to *null* if not using BLE.
- A Class Of Device.
- Your Bluetooth address.
- Your Ministak Licence Key, if any. You can leave it *null* otherwise.
- The Trim of your Bluetooth controller.
- A Pin code, if you are using it.
- Your device IO capabilities.
- Whether you want to force MITM protection or not.

What are the IO capabilities?

IO capabilities tell the Bluetooth controller whether you're device has a display and a keyboard attached to it. The association mode in the BD/EDR simple pairing process that your controller will use will depend directly on the IO capabilities on your device. If you use the IO_CAPS_NO_INPUT_NO_OUTPUT capability, you'll force the controller to use the *just works* association mode, which doesn't require of any user interaction but lacks of MITM protection. Any other modes will require some user interaction, please check the Bluetooth Core Specification to know more about the simple pairing process.

What is the remote device database?

That is the database Ministak will use to store the Link Keys of any paired device. Ministak is shipped with a simple database that stores the Link Keys on volatile memory. If you want store them on persistent memory or you want any specific behaviour, you will need to implement your own device database. Any valid database must implement the interface defined by the remote_device_db_t structure. Look at the Doxygen documentation or at remove_device_db.h to check out what are all the methods in the interface for.

How do I get any messages from Ministak?

When you call btstack_init, you'll pass it a structure with a set of callbacks, defined on the btastck_events structure. Callbacks are named after the type of events they report about. Of them, you need to implement hci_available, which will get called any time the HCI channel is available for you to communicate with the Bluetooth controller (see *How do I communicate with the Bluetooth controller in my application?*).

Though you can leave it *null*, you'll probably also need to implement the hci_event callback method. hci_event will get called every time the Bluetooth controller sends an HCI command or event to the host. As input argument, you'll get the raw HCI data received from the controller. The first byte in that data is the HCI op. code and the rest contains its arguments, which vary depending on the op. code. Please look at the Bluetooth core spec. to check out the available op. codes and their arguments.

The rest of the callbacks in btstack_events you can choose to implement them or leave them *null* depending on whether you plan handling them or not.

How do I know when Ministak is initialized?

When Minsitak finishes its initialization it will call the hci_available method you registered by means of btastck_init for the first time (see *How do I initialize Ministak?)*

How do I set the PSKeys?

You will be provided with a pskeys.h file that will be written when porting Ministak to your platform and which contains the minimal set of PSKeys Minsitak needs to run, if not agreed otherwise. Any other PSKeys you may need to set, you can add them to the pskeys.h file following the format {ID, SIZE, DATA} where ID is a one word field, size is a two words field with the size of the data in number of words. Each word two bytes.

The structure declared in pskeys.h is passed as an argument to pskeys_init and Minsitak will load them on the controller on the initialization.

How do I set the patches?

You will be provided with a patches.h file that will be written when porting Minsitak to your platform. That will contain the latest patches publicly available for your controller at the time of the porting. However, it is advisable to keep up to date with CSR or your local supplier in order to get your patches file up to date.

The structure declared in patches.h is passed as an argument to pskeys_init and Minsitak will load them on the controller on the initialization.

How do I communicate with the Bluetooth controller in my application?

When using any Bluetooth profile, you should use the API Ministak provides for it. If you need to send specific HCI commands to the controller, check if there is any method in the HCI API that suits your needs. If not, you can use the hci_send_cmd function to send any possible HCI command (look at the available op. codes in hci_cmds.h) along with their arguments. Please look at the Bluetooth core spec. to check out the available op. codes and their arguments.

You should always make sure that the HCI channel is free for your application to use it, since in any interrupt-driven procedure or in a multithreaded environment either Ministak or the controller could be using the HCI channel at the same time you try to write on it.

Whenever you get your hci_available callback called you can be sure the HCI channel is free for you to use it. Otherwise, you need to call hci_can_send_packet_now with the appropriate argument (HCI_COMMAND_DATA_PACKET or HCI_ACL_DATA_PACKET depending on what you are trying to do). A common solution is to make your application behave like a state machine that triggers communications that are performed only in the hci_available callback only if the HCI channel is available.

```
enum STATES {
       BUTTON_PRESSED,
       SPP_DATA_RECEIVED,
       NONE,
};

enum STATES my_state = SPP_DATA_RECEIVED;

void hci_available() {
  if (hci_can_send_packet_now(HCI_COMMAND_DATA_PACKET)) {
    // only hci commands
    switch (my_state) {

       case BUTTON_PRESSED:
         // Do whatever, for instance, turn discoverable on/off
         my_state = NONE;
         break;

     }
   }
   if (hci_can_send_packet_now(HCI_ACL_DATA_PACKET)){
     // only acl data packets
     switch (my_state) {

       case SPP_DATA_RECEIVED:
         /* Do whatever, for instance answer back with spp_send */
         my_state = NONE;
         break;
       }
    }
}

void button_pressed() {
       my_state = BUTTON_PRESSED;
       hci_available();
}

void spp_data_received() {
       my_state = SPP_DATA_RECEIVED;
       hci_avaiable();
}
```

Notice that if the HCI channel is not available when, for example, button_pressed gets called, the next time it is available, hci_available will get called and my_state will have been set properly so the appropriate action will be triggered.

A simpler approach is to check `hci_can_send_packet_now(HCI_COMMAND_DATA_PACKET)` and `hci_can_send_packet_now(HCI_ACL_DATA_PACKET)` on the same if condition. That would avoid splitting your communications with the controller in two different blocks of code and will ease code maintenance. In cases where the communications latency and throughput is not critical that probably is a good solution.

How do I use any profile in Ministak?

You will need to initialize any profile you are going to use after calling rfcomm_init (see *How do I initialize Ministak?*). When you do it, you will need to give Ministak a set of callback functions Ministak will call any time it receives any event on that profile.

Profiles will usually provide their own methods to open and close connections, as well as to send any data over that profile.

How do I send data over the SPP profile?

As any other profile, you'll first need to initialize it (see *How do I use any profile in Ministak?*). After that, you just need to use the spp_send method in the SPP API. You should always use it inside the hci_available method (see *How do I communicate with the Bluetooth controller in my application?*).

How do I receive data over the SPP profile?

As any other profile, you'll first need to initialize it (see *How do I use any profile in Ministak?*). After that, you will get your spp_recv callback called any time Ministak gets any data received over the SPP.

How do I close a connection?

If you just want to close a specific profile, you must use the function the profile provides. It will usually be called after the profile name and ending with the _disconnect suffix.

On the other hand, if you want to completely close a connection with another device, and thus close any open profile, you can use the hci_disconnect command (declared in hci_cmds.h).

How do I pair with a device?

If any device tries to pair with Ministak, you'll get the callback hci_event called with the HCI_EVENT_USER_CONFIRMATION_REQUEST op. code. Then, you'll need to either accept or reject the pairing request by calling hci_send_cmd with the hci_user_confirmation_request_reply (see *How do I get any messages from Ministak?* and *How do I communicate with the Bluetooth controller in my application?*).

In order to start pairing with a Bluetooth device, just open a connection by opening a profile to a given Bluetooth address or by using the hci_create_connection op. code in hci_send_cmd. Your Bluetooth controller will then manage any communications needed for the pairing to succeed.

How do I bond with a device?

If you're using BLE, the Bluetooth controller may need to start encryption when accessing any GATT characteristics that require it. In that case, Ministak will need to store a Long Term Key (TLK) for any given device it started encryption with. The database you provide Ministak in btstack_init (see

What is the remote device database?) will need manage these LTK and implement the methods get_ltk and put_ltk. Put_ltk will get called any time Ministak needs to store a LTK, while get_ltk will get called any time Ministak needs to retrieve a previously stored LTK.

How do I use BLE as a master?

In order to use BLE in Ministak as a master you'll need to implement your own GATT database. The GATT database contains the description of all the GATT services your device provides along with their characteristics and the handles to access them. The GATT database must match its definition in the Bluetooth Core spec. and each record in the database can be implemented by means of the attribute structure defined in gatt.h.

You're encouraged start by looking at the demo-ble.c example you'll be provided with Ministak.

How do I use BLE as a slave?

For you to connect to a BLE master, you'll need to first discover its GATT database and store whatever information and handles you'll use. After discovering services, you can read and write to and read from the GATT characteristics using the le_central_write_request and le_central_read_request methods in le_central_att.h. You'll find there the methods you need to do the service discovery as well.