# BFM100 Communication Protocol
# Version 2.1

Preliminary

**Disclaimer:**
BioIentic reserves the right to make changes, without further notice, to any product, including circuits and/or software described or contained in this document in order to improve design and/or performance.

# 1. Overview

The communication protocol used by the BFM100 contains 3 bytes header, data with variable size and termination field containing check sum. This packet structure is the same for communication in the both directions (from HOST to the fingerprint module BFM100 and from BFM100 to the HOST). The structure of the communication packet is shown on Fig. 1.

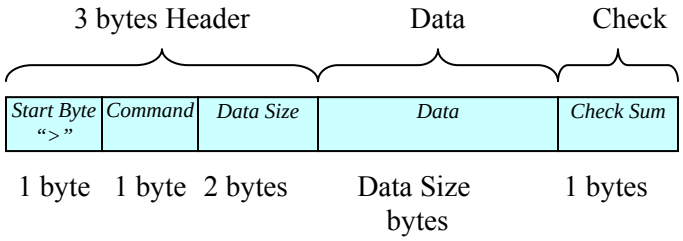



Fig. 1

- Start byte is 1 byte. It defines the beginning of a packet. The value is 0x3E (ASCII ">") .
- "Command" is a 1 byte field. Refer to the Command Table below.

- "Data Size" specifies the size (number of bytes) of the "Data" field. "Data Size" is 2 bytes long. The complete packet can be maximum 1Kbyte long so "Data Size" can be maximum 1019 bytes.
- "Checksum" is 1 byte. It is a sum of all bytes in the "Header" and "Data". Only the least significant byte from the sum is taken into account.

There are two types of packets. Request packets from HOST to the fingerprint module (will call them simply ***request)*** and response packets from module to the HOST (will call them simply ***response***). Each ***request*** is usually followed immediately by ***response***. In same cases the ***response*** is given after fingerprint has been detected and process by the module.

Note1: The transmission of the data byte order is "Little Endian", the lowest byte is transmitted first and the lowest word is transmitted first.

Note2: The empty fields of the communication protocol are denotes like this ""

Note3: Some of the packet fields can take set of predefined values. These values are separated by /

Note4: If packet field in this document is left blank it means that its value varies depending of other packet data.

Note5: All *response* packets contain the error code as first byte in their data field. The error codes (1 byte) are unique and can be seen in Table 2. Detailed description of all possible errors is included below each *response* packet.

Note6: The HOST should not send batch of packets it should send the next packet after he has received the response of the previous one. No packets buffering is implemented in BFM100. An exception of this rule is valid for several system operation commands which set the general mode of operation of the fingerprint module. These commands are Enroll Finger (single), Enroll Finger (multiple), Verify Finger (1:1), Verify Finger (1:N), Verify Finger (1:Nset), etc. All these have two repose packets and the HOST can change the operational mode after he has received the first response packet. Sending extra commands during image dumping is not recommended.

# 2. Command packets

All available commands are shown in Table 1. Detailed command description follows. The unique error codes are defined in Table 2.

| Command category | Command ID | Code | Description |
|---|---|---|---|
| **Support** | Invalid Checksum | 0x01 | This packet is sent from module to the HOST each time the start byte ">" is detected by the module but there is check sum error. |
| | Finger Detected | 0x03 | Each time a finger is detected the module sends this packet. |
| **System Management** | Reset Module | 0x05 | Resets the Fingerprint module |
| | Get Firmware Version | 0x06 | As a response the module returns the version of the firmware |
| | Write Parameters | 0x10 | Writes module parameters which control the fingerprint recognition process. |
| | Read Parameters | 0x11 | Reads the module parameters |
| | Set Time | 0x15 | Set the RTC inside the module |
| | Get Time | 0x16 | Get the date-time stamp. |
| **System Operation** | Operation Mode | 0x20 | Switching between "Autonomous" and "Controlled" operational mode. |
| | Enroll Finger (single) | 0x25 | Enrolls finger based on single finger impression. Finger ID is provided by the HOST or is generated by the module. |
| | Enroll Finger (multiple) | 0x26 | Enrolls finger based on several impression of a same finger. |
| | Verify Finger (1:1) | 0x30 | Verifies if the next finger matches with given template. |

| | | | |
|---|---|---|---|
| | Verify Finger (1:N) | 0x31 | Verifies if the next finger matches with any template in the database. |
| | Verify Finger (1:N set) | 0x32 | Verifies if the next finger matches with a finger from template set specified in the data field of this command. |
| | Read Status | 0x40 | Reads the status structure of the Fingerprint module. |
| **Database Management** | Read Template | 0x50 | Reads given template from database. |
| | Write Template | 0x51 | Writes the template provided by the HOST into the module FLASH memory. |
| | Read templateID List | 0x55 | Returns the list of all templateIDs stored in the module. |
| | Delete Template | 0x60 | Deletes permanently given template from database. This command is irreversible! |
| | Delete Whole Database | 0x61 | Deletes permanently the whole database. This command is irreversible. |
| | Master Template | 0x65 | Assigns/clears master property to a given fingerprint template. |
| **Dump images** | Dump Grayscale Image | 0x70 | The module dumps last grayscale fingerprint image captured from the sensor. |
| | Dump Binary Image | 0x75 | The module dumps last fingerprint image after it has been binarized. |
| **User Interface** | Set GPO | 0x80 | Sets the general purpose outputs of the module |
| | Beep | 0x83 | Generates several sounds using buzzer built in the fingerprint module. |
| | Get GPI | 0x85 | Gets the general purpose inputs of the module |

| | | | |
|---|---|---|---|
| | Buttons State | 0x86 | If the user presses a button of the fingerprint module a notification packet is sent. |

Table 1. Commands Table

| Error ID | Code | Description |
|---|---|---|
| No Error | 0x00 | Successful completion of the requested operation. This code is used to indicate the match with the given template or with any template from the database. |
| Invalid Value | 0x05 | The HOST has issued wrong value in its *request* packet. |
| Low Quality | 0x10 | The fingerprint image has a quality too low to be proceeded. |
| Inconsistent Fingerprints | 0x15 | The consistency of the finger impressions is too low to perform a multiple enrolment. |
| Mode Set | 0x20 | The mode of the fingerprint module is set correctly. This error code is returned by the Enroll/Match commands. |
| Template not Found | 0x50 | The requested template is not found in the database. |
| Database is Empty | 0x55 | The database is empty so verification (1:N) can not be set. |
| Template ID already exists | 0x51 | The template ID used in "*Write Template*" command already exists. |
| FLASH Write Error | 0x80 | Error during writing in the FLASH memory of the module. |
| FLASH Read Error | 0x81 | Error during reading from the FLASH memory of the module. |

| | | |
|---|---|---|
| FLASH is Full | 0x85 | FLASH memory is full and the requested command can not be completed. |
| Last Image Packet | 0xA0 | Used to indicate the last packet in the image transmission. |
| Image Check Sum Error | 0xA1 | Check sum error in the last data packet (for image transmission) was detected by the HOST. The wrong packet will be resent by the module. |
| There is no Valid Image | 0xA2 | There is no Valid Image to be sent to the HOST. |
| Wrong Confirmation Packet | 0xA3 | The *confirmation* packet sent by the HOST is invalid. |
| Timeout for Confirmation. | 0xA4 | Data packet (during the image transmission) is sent but no confirmation from the HOST arrived within certain amount of time. |
| Do Not Match | 0xFF | Used to indicate that the finger doesn't match with the provided template or with any template in the database. |

Table 2. Errors Table

## 2.1 Support packets

### *Invalid Checksum (0x1)*

This packet is sent from module to the HOST each time the start byte ">" is detected by the module but there is check sum error. No response to this packet expected by the module.

Request:

| | | | | |
|---|---|---|---|---|
| "> " | *0x01* | 0x0000 | "" | *0x3F* |

### *Finger detected (0x03)*

Each time a finger is detected on sensor the module sends this packet. Module does not expect response to this packet.

Request:

| “>” | *0x03* | 0x0001 | Error Code=0x00 | *0x42* |
|---|---|---|---|---|

## 2.2 System Management

### *Reset Module (0x5)*

Resets the Fingerprint module. This command exists as *request* only.

Request:

| “>” | *0x05* | 0x0000 | “” | *0x43* |
|---|---|---|---|---|

### *Get Firmware Version (0x6)*

The HOST issues this command to retrieve the version of the firmware. There are 2 versions of this command, *request* (from HOST to the module) and *response* from module to the HOST. In the response the version is put in the *data* field of the command.

Request:

| “>” | *0x06* | 0x0000 | “” | *0x44* |
|---|---|---|---|---|

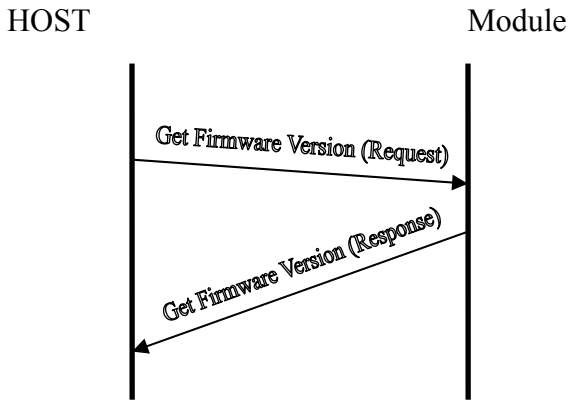
Response:

| “>” | *0x06* | 0x0003 | | |
|---|---|---|---|---|

| Error Code | Firmware Version |
|---|---|
| 1 byte | 2 byte |

- *Error Code=0x00 – No error*

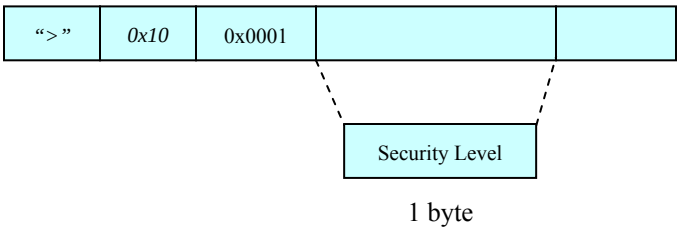
*Timeline*



## *Write Parameters (0x10)*

Write module parameters which control the fingerprint recognition process. There are 2 versions of this command, *request* (from HOST to the module) and *response* from module to the HOST. The *request* command contains the parameters to be stored in the module. Currently only the security level is defined as parameter. It can be from 0 (worst security) up to 255 (the best security). The parameters are written in FLASH memory of the module so their values remain after restart.

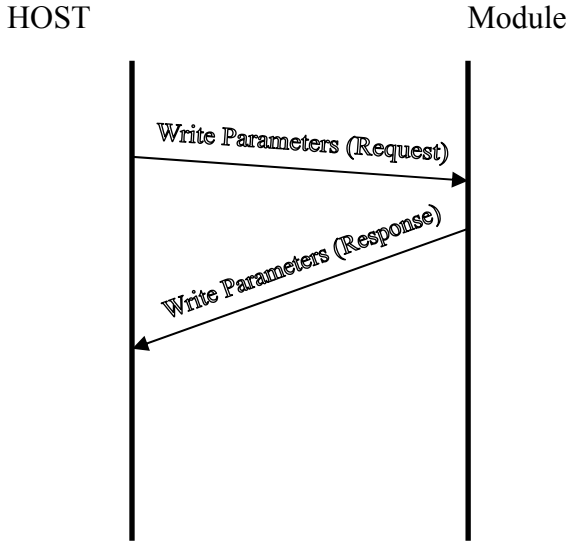
Request:




Response:


- *Error Code = 0x00* – No error
- *Error Code = 0x05*– The HOST has specified wrong parameter value
- *Error Code = 0x80*–FLASH memory write error

*Timeline*

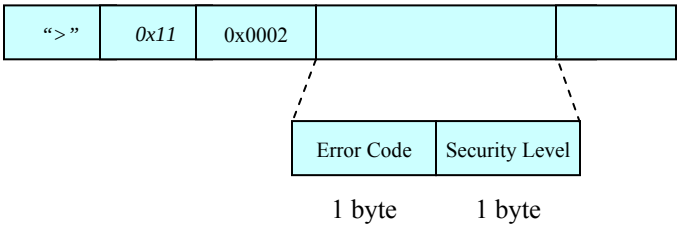


## *Read Parameters (0x11)*

Read the module parameters which control the fingerprint recognition process. There are 2 versions of this command, *request* (from HOST to the module) and *response* from module to the HOST. The *response* command contains the parameters from the module. Currently only a security level is defined as parameter. It can be from 0 (worst security) up to 255 (the best security).

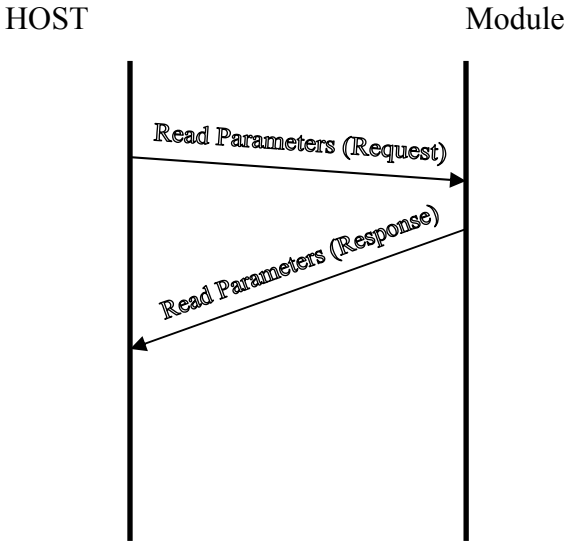
Request:


Response:




- *Error Code = 0x00* – No error

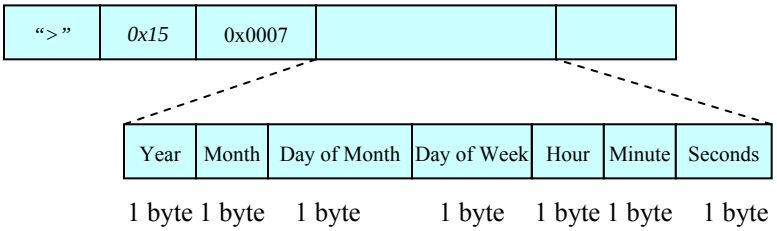
*Timeline*



## *Set Time (0x15)*

Set the Real Time Clock of the module. The date-time is send in the data field of *request* packet defined as 7 bytes long vector with one byte for year, month, day, hour, minute and seconds. Not that all 7 fields should be in binary coded decimal form (BCD). A response packet shows a successful set of the date-time.

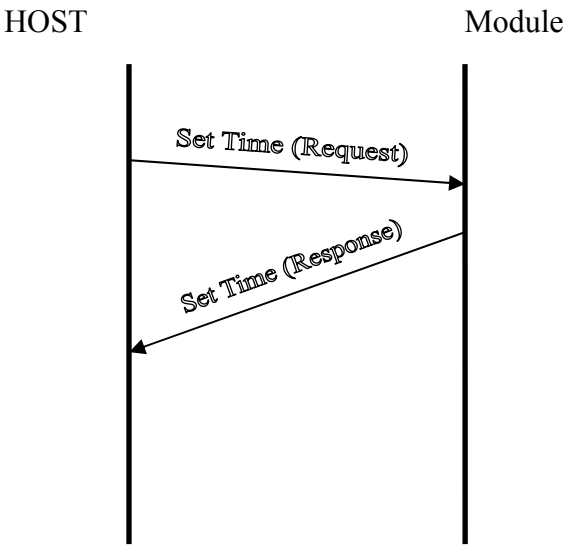
Request:




Response:


- *Error Code = 0x00* – No error
- *Error Code = 0x05*– The HOST has specified invalid time stamp

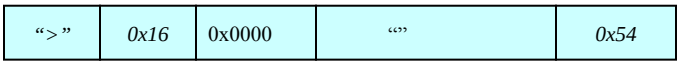
*Timeline*



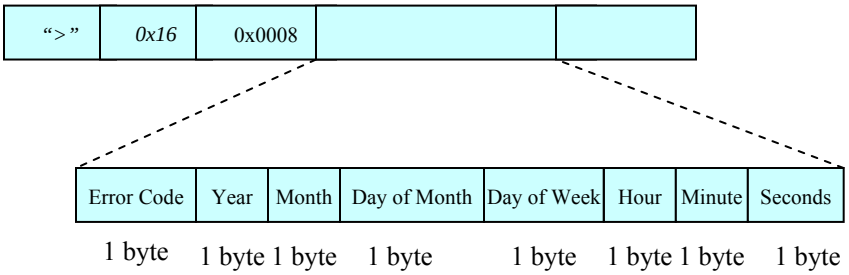
## *Get Time (0x16)*

Retrieve the time from the Real Time Clock of the module. The date-time is send to the HOST in the data field of *response* packet and is defined as 7 bytes long vector with one byte for year, month, day of month, day of week, hour, minute and seconds. All time fields are in binary coded decimal form (BCD).

Request:

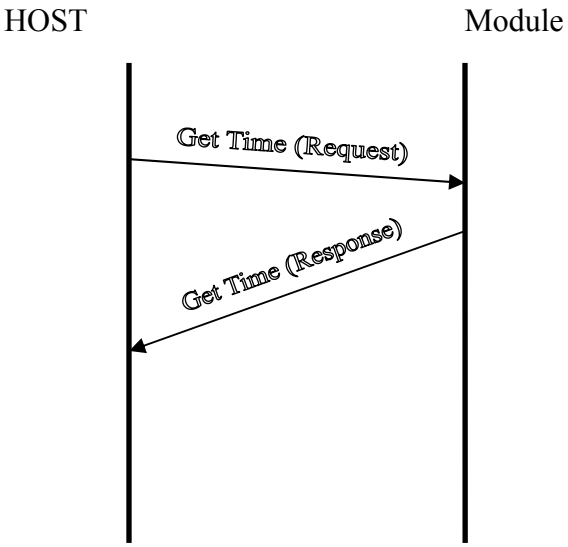



Response:




- *Error Code = 0x00* – No error

*Timeline*



## 2.3 System Operation

### *Operation Mode (0x20)*

The BFM100 supports "Autonomous" and "Controlled" operational modes. In "Autonomous" operation (*Operation = 0x00*) the button supported by the module can do some basic module control (Check BFM100 user manual for more information). In "Controlled" operation (*Operation = 0x01*) the buttons don't influence the operation of the module but can still be read using ***GetGPI*** and the event packets ***Button State*** are still sent. In both "Autonomous" and "Controlled" operation the control of the module can be done using commands.

Request:

| | | | | |
|---|---|---|---|---|
| ">" | *0x20* | *0x0001* | Operation = 0x00/0x01 | *0x5E* |

Response:

| | | | | |
|---|---|---|---|---|
| ">" | *0x20* | *0x0001* | *Error Code=0x00* | *0x5F* |

- *Error Code = 0x00* – Operation mode set.

### *Enroll Finger (single) (0x25)*

Sets the module in enrollment mode based on single finger impression. Next finger will be processed and its template will be saved in the FLASH database inside the module. There are *request* and *response* packets of this type. TemplateID (2 bytes long) is provided by the HOST or is generated automatically by the module (in this case the data field in the *request* packet should be blank). Both an error code and the templateID (from the *request* packet or the one which was automatically generated by the module) are sent to the HOST by the *response* packet. Note that templateIDs above 65500 are reserved for system purpose and should not be sent by the HOST. If a TemplateID is issued by the HOST the attempt for enrolment after the first one will generate *Response1* with *Error Code* = 0x60.

Request:

| | | | | |
|---|---|---|---|---|
| ">" | *0x25* | 0x0002/0x0000 | templateID/"" | */0x63* |

Response1:

| “>” | *0x25* | 0x0001 | 0x20 | 0x84 |
|---|---|---|---|---|

- *Error Code = 0x05*– Invalid Value for *templateID*
- *Error Code = 0x20* – Verification mode is set correctly.
- *Error Code* = 0x60 - TemplateID already exists in the database.

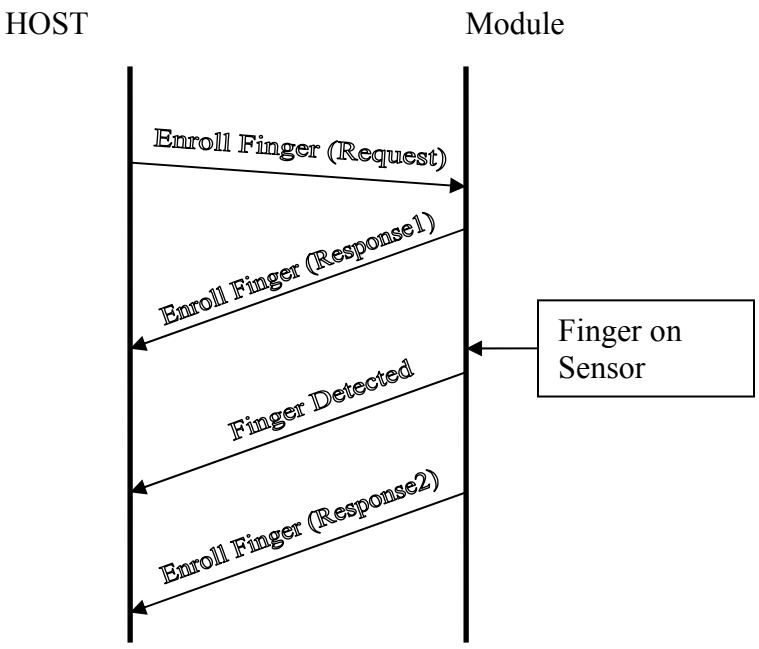
Response2:

| “>” | *0x25* | 0x0001/0x0003 | | |
|---|---|---|---|---|

| Error Code | “”/templateID |
|---|---|
| 1 byte | 2 byte |

- *Error Code = 0x00* – No error
- *Error Code = 0x10*– Low quality of the fingerprint
- *Error Code = 0x80*–FLASH memory write error
- *Error Code = 0x85*–FLASH memory is full

*Timeline*



## *Enroll Finger (multiple) (0x26)*

Sets the module in enrollment mode based on several impression of a same finger and save it as template in the FLASH database inside the module. The number of impressions is detected automatically by the module. There are *request* and *response* packets of this type. TemplateID (2 bytes long) is provided by the HOST or is generated automatically by the module (in this case the data field in the *request* packet should be blank). Both an error code and the templateID (from the *request* packet or the one which was automatically generated by the module) are sent to the HOST by the *response* packet.
Note that templateIDs above 65500 are reserved for system purpose and should not be sent by the HOST. If a TemplateID is issued by the HOST the attempt for enrolment after the first one will generate *Response1* with *Error Code* = 0x60.

Request:

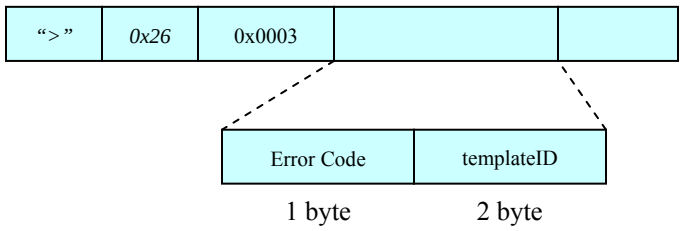
| "> " | *0x26* | 0x0002/0x0000 | templateID/"" | */0x64* |
|---|---|---|---|---|

Response1:

| "> " | *0x26* | 0x0001 | Error Code | |
|---|---|---|---|---|

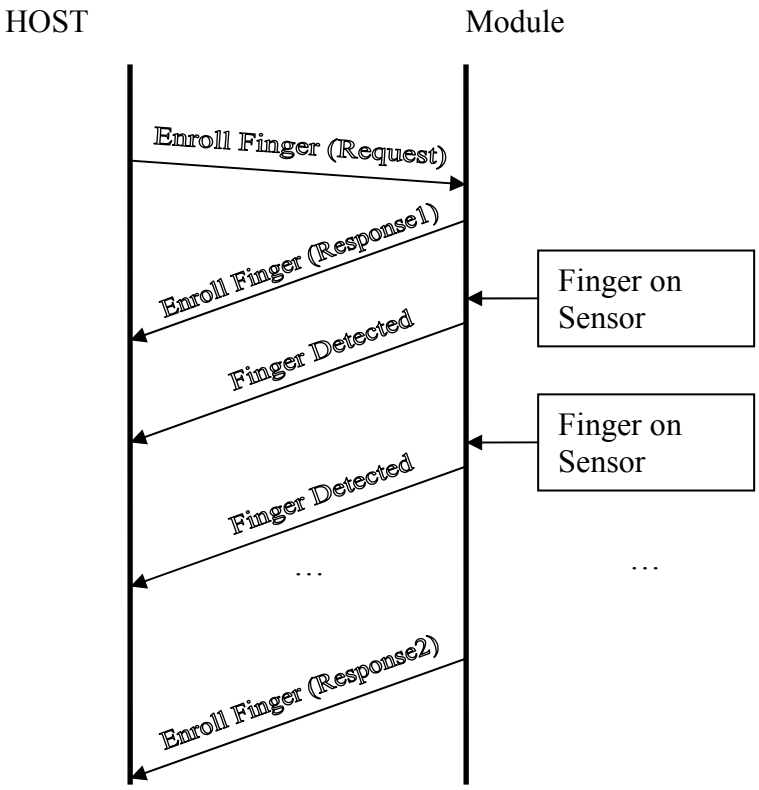
- *Error Code = 0x05*– Invalid Value for *templateID*
- *Error Code = 0x20* – Verification mode is set correctly.
- *Error Code* = 0x60 - TemplateID already exists in the database.

Response2:




- *Error Code = 0x00* – No error
- *Error Code = 0x10*– Low quality of the fingerprints
- *Error Code = 0x15*– Inconsistent fingerprints for multiple enrolment
- *Error Code = 0x80*–FLASH memory write error
- *Error Code = 0x85*–FLASH memory is full

*Timeline*




## *Verify Finger (1:1) (0x30)*

Sets the module in a verification mode with given template. There are *request* and *response* packets of this type. The templateID is sent (2 bytes long) by the HOST in the *request* packet. TempateID specifies a template from database and the next finger is compared with this template. Both an error code and the templateID (from the *request* packet) are sent to the HOST by the *response* packet. Note that templateIDs above 65500 are reserved for system purpose and should not be sent by the HOST.

Request:

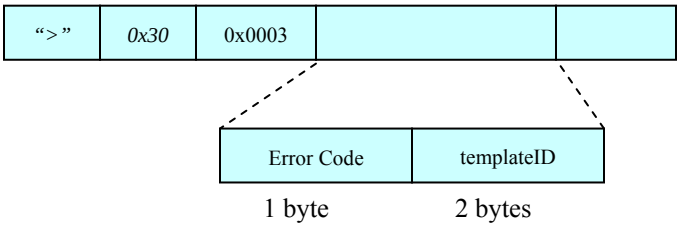
| “>” | *0x30* | 0x0002 | templateID | |
|---|---|---|---|---|

Response1:

| “>” | *0x30* | 0x0001 | 0x20 | 0x8F |
|---|---|---|---|---|

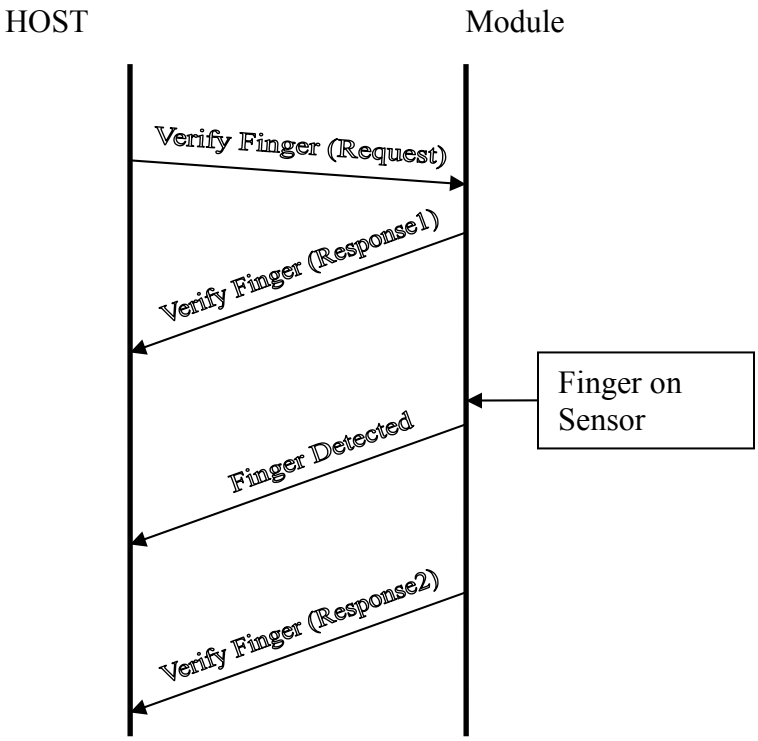
- *Error Code = 0x05*– Invalid Value for *templateID*
- *Error Code = 0x20* – Verification mode is set correctly.
- *Error Code = 0x50*–Template not found

Response2:




- *Error Code = 0x00* – The finger matches with the template
- *Error Code = 0xFF*– The finger doesn’t match with the template

*Timeline*



## *Verify Finger (1:N) (0x31)*

Sets the module in a verification mode (1:N) using all templates from database. In this mode the whole database is checked. There are *request* and *response* packets of this type. If a matching template is found (the first match from the beginning of the database is considered), its' templateID is sent to the HOST by the *response* packet. Otherwise only the error code is issued in the *response* packet.

Request:

| "> " | *0x31* | 0x0000 | "" | 0x6F |
|---|---|---|---|---|

Response1:

| "> " | *0x31* | 0x0001 | 0x20 | 0x90 |
|---|---|---|---|---|

- *Error Code = 0x20* – Verification mode is set correctly.
- *Error Code* = 0x55 - Database is Empty so (1:N) verification can not be set

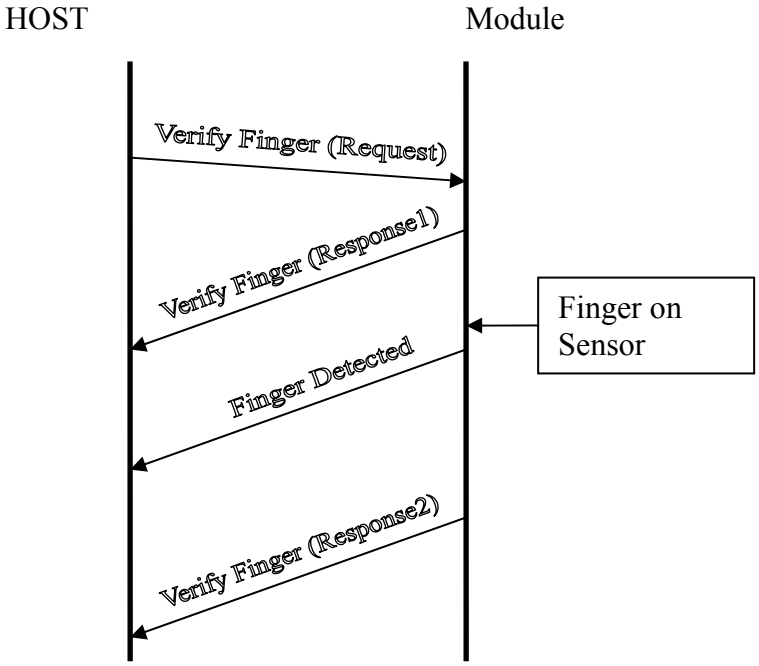
Response2:

| "> " | *0x31* | 0x0003/0x0001 | | |
|---|---|---|---|---|

| Error Code | templateID/"" |
|---|---|
| 1 byte | 2 bytes |

- *Error Code = 0x00* – The finger matches with a template from database
- *Error Code = 0xFF*– The finger doesn't match any template from the database

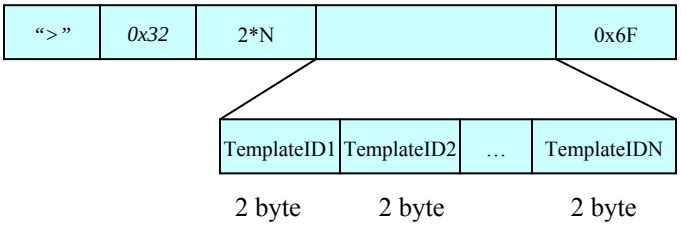
*Timeline*



## *Verify Finger (1:Nset) (0x32)*

Sets the module in a verification mode based on a set of templates which is specified in the data field of the *request* packet. There are *request* and *response* packets of this type. In case of matching (the first match from the beginning of the set is considered) with a template its templateID is sent to the HOST by the *response* packet. Otherwise only the error code is issued in the *response* packet. Note that the templateIDs from the set which are not found in the database are ignored. The templateIDs above 65500 are reserved for system purpose and should not be sent by the HOST.

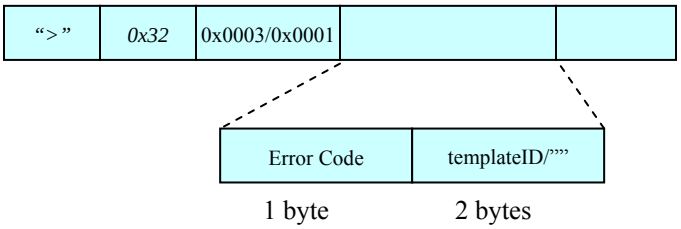
Request:




Response1:

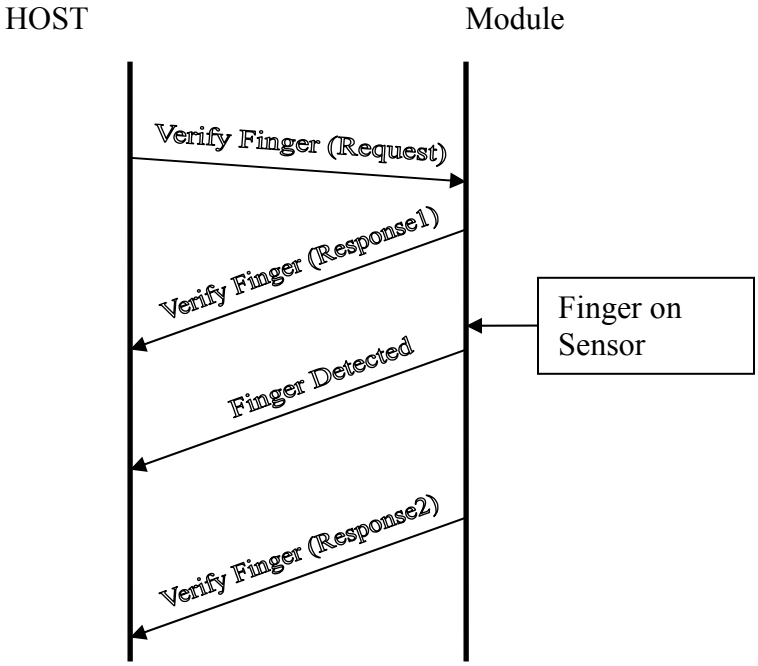

- *Error Code = 0x05*– All *templateIDs* are invalid.
- *Error Code = 0x20* – Verification mode is set correctly.
- *Error Code = 0x50*–There is no a template from the list available in the database.

Response2:



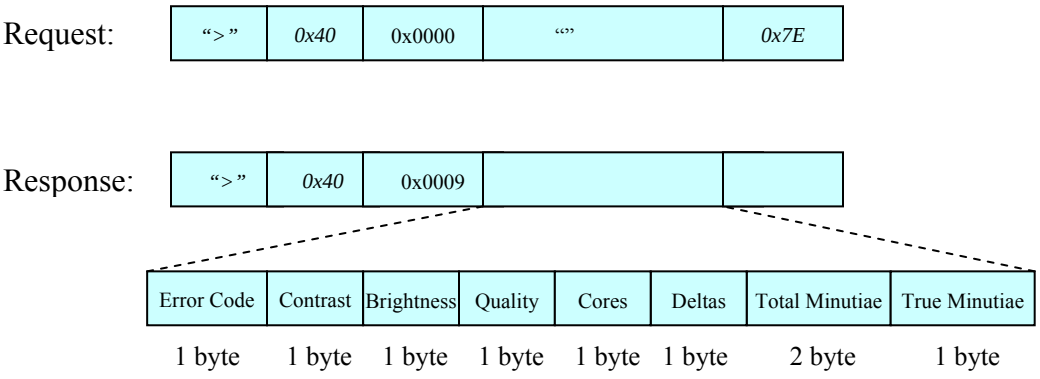

- *Error Code = 0x00* – The finger matches with a template from database
- *Error Code = 0xFF*– The finger doesn't match any template from the provided set

Timeline



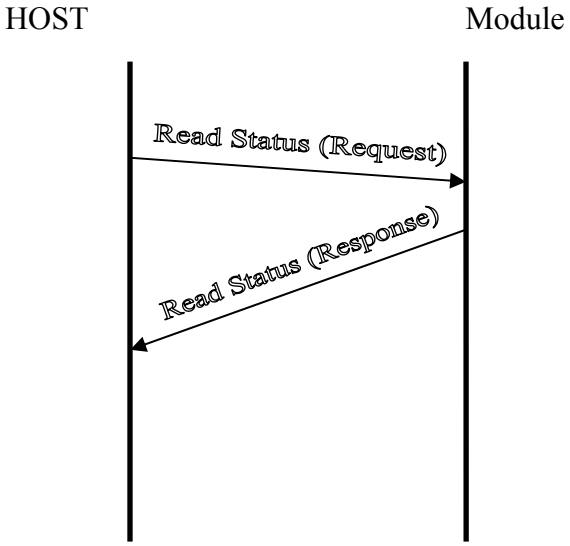

### *Read Status (0x40)*

Read the status structure of the Fingerprint module. There are *request* and *response* packets of this type. The *response* command contains the fingerprint status structure for the previous fingerprint processing. The status contains - contrast of the original grayscale image, brightness of the original grayscale image, quality of the fingerprint image, number of core singularities, number of delta singularities, total number of minutiae and finally the true minutiae number.




- *Error Code = 0x00* – No error

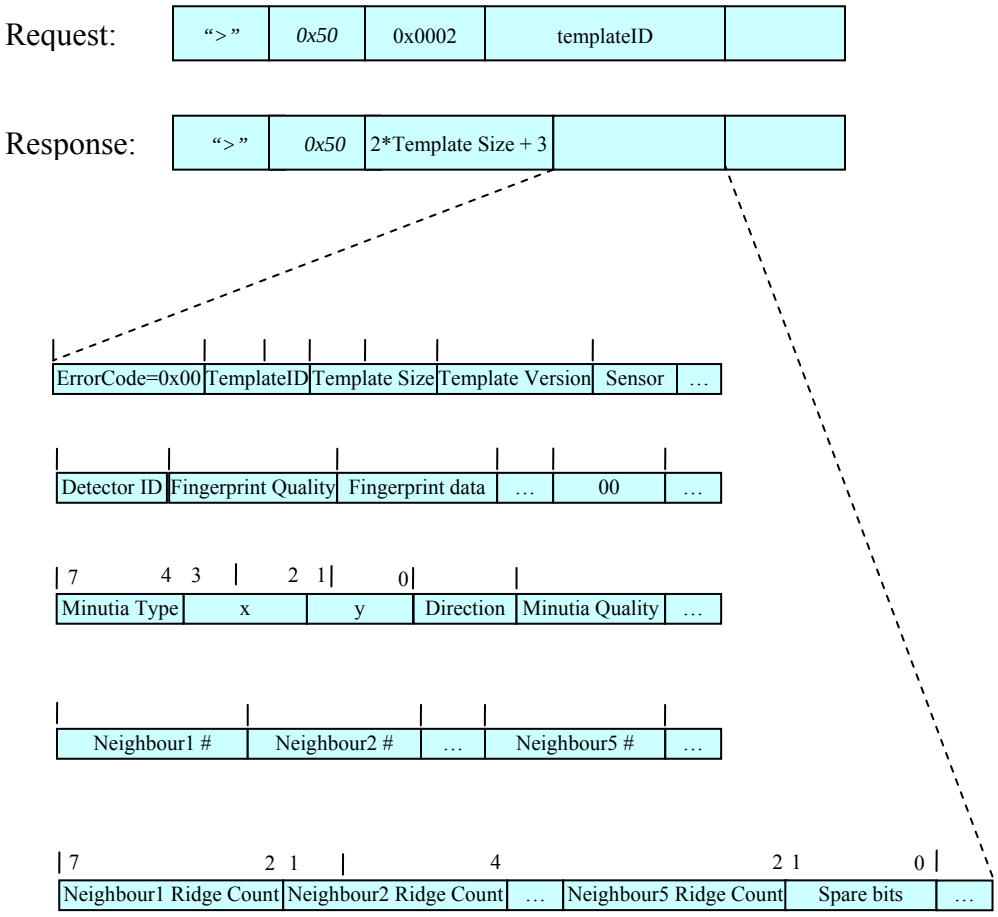
*Timeline*



## 2.4 Database Management

### *Read Template (0x50)*

Read of template from the module database. The templateID is specified in the *request* packet and the actual template is returned in the *response* packet. The structure of a template record is as it is shown below. For simplicity the bytes boundaries are shown with small lines above the data field. Digits above the data field indicate bit number in the byte. All fields in the template record are described below. In case of error a single 1 bytes error code is returned in the data field of the *response* packet. Note that templateIDs above 65500 are reserved for system purpose and should not be sent by the HOST.




- *Error Code = 0x00* – No error

Response:

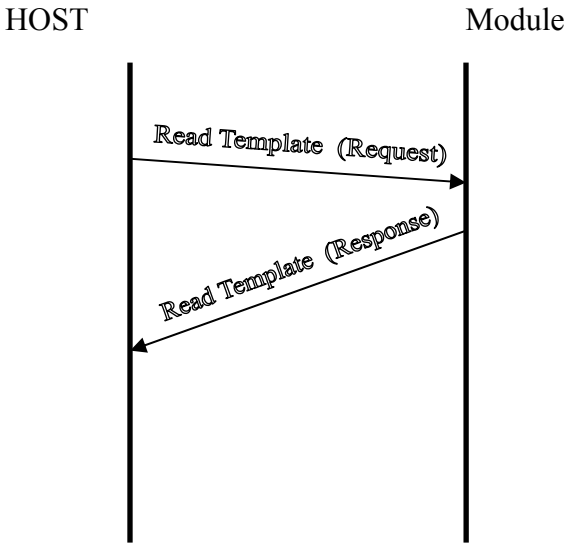
| ">" | *0x50* | 0x0001 | Error Code <> 0x00 | |
|---|---|---|---|---|

- *Error Code = 0x50*–Template not found
- *Error Code = 0x05*– Invalid Value for *templateID*
- *Error Code = 0x81*–FLASH memory read error

*Template record fields:*

- **TemplateID (2 bytes)**
  The ID of the template sent by the HOST in the *request* packet.
- **Record Size (2 bytes)**
  The Template Record starts with *Record Size*, which specifies its size in words.
- **Template Version (1 byte)**
  Version of the template. We define 2 records version
  1 – Record as it is defined above but without neighbor indexes.
  2 – Record with neighbor indexes as it is shown above.
- **Sensor ID (1 byte)**
  This byte specifies the type of the sensor used capturing of the fingerprint image. *Sensor ID*=1 for the Fujitsu MBF200.
- **Detector ID (1 byte)**
  Specifies the minutiae detection and template creation algorithm used for creation of the minutiae data in this record.
- **Fingerprint Quality (1 byte)**
  Global quality of minutiae template.
  0 – no information in the minutiae record
  1 – minimum minutiae quality
  255 – maximum minutiae quality
- **Fingerprint data (N words)**
  User defined data like name of the person which finger is enrolled, index of the finger on the hand, etc. This field is N words long and is terminated by zero byte.
  If you don't use it you must put single zero word. N should be between 1 and 100
- **Minutia Type (4 bits for each minutia)**
  Type of the minutiae. Only the following combinations are defined now
  0000 – Bifurcation
  0001 – Ridge ending
- **x (10 bits for each minutia)**
  x coordinate of the minutia in pixels staring at the top of the image.
- **y (10 bits for each minutia)**
  y coordinate of the minutia in pixels staring at the left side of the image.
- **Direction (1 byte for each minutia)**
  Direction of the minutia.
  0 – 0 [rad]
  255 – 2π rad]
- **Minutia Quality (1 byte for each minutia)**
  Quality of minutia.
  0 – no information in this minutia
  1 – minimum minutia quality
  255 – maximum minutia quality

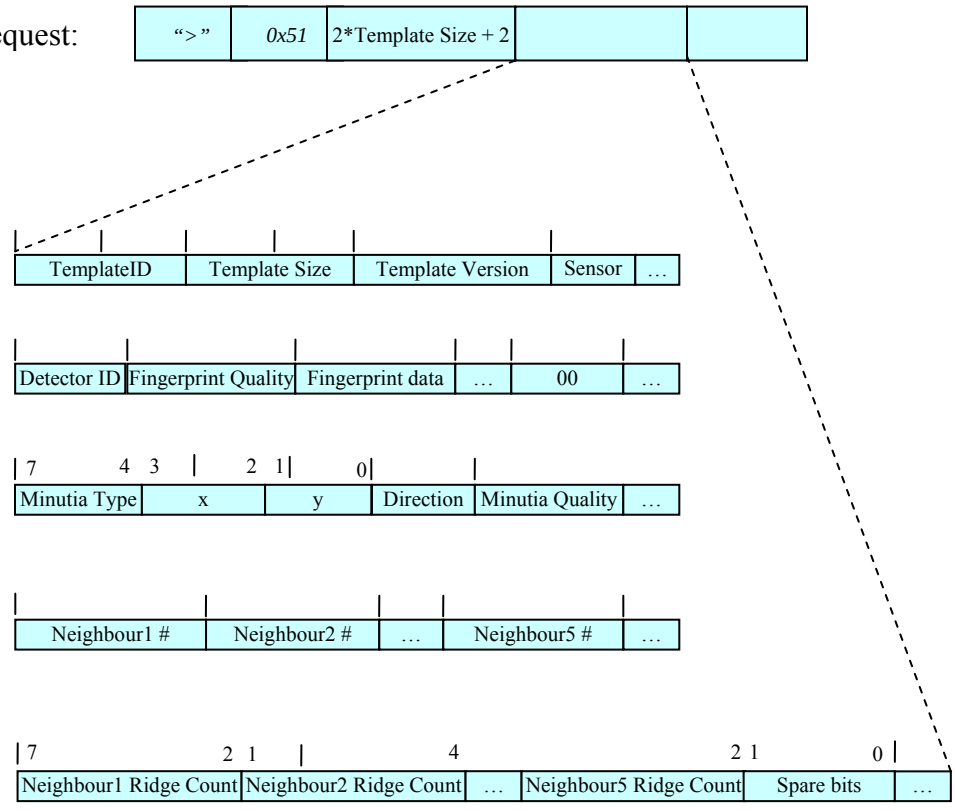
*Timeline*




## *Write Template (0x51)*

Write template to the FLASH database in the module. The template record is provided in the *request* packet. The structure of a template record can be seen in the description of "**Read Template**" packet. The command result is returned in a *response* packet.

Request:



Response:

| ">" | *0x51* | 0x01 | Error Code | |
|---|---|---|---|---|

- *Error Code = 0x00*- No error
- *Error Code = 0x05*– Wrong template or reserved TemplateID.
- *Error Code = 0x60* - Template ID already exists
- *Error Code = 0x80*–FLASH memory write error

## Read TemplateID List *(0x55)*

Returns (in the *response* packet) the list of all templateIDs stored in the FLASH database. Each templateID is 2 bytes long.

Request:

| ">" | *0x55* | 0x0001 | List Type | |
|---|---|---|---|---|

- *List Type = 0x00*- All fingerprint templates are returned
- *List Type = 0x01*- Only master fingerprint templates are returned

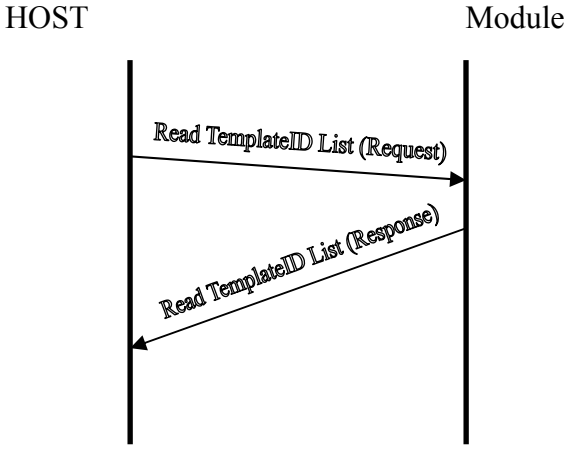
Response:

| ">" | *0x55* | 2*N+1 | | |
|---|---|---|---|---|

| Error Code | TemplateID 1 | TemplateID 2 | … | TemplateID N |
|---|---|---|---|---|
| 1 byte | 2 bytes | 2 bytes | | 2 bytes |

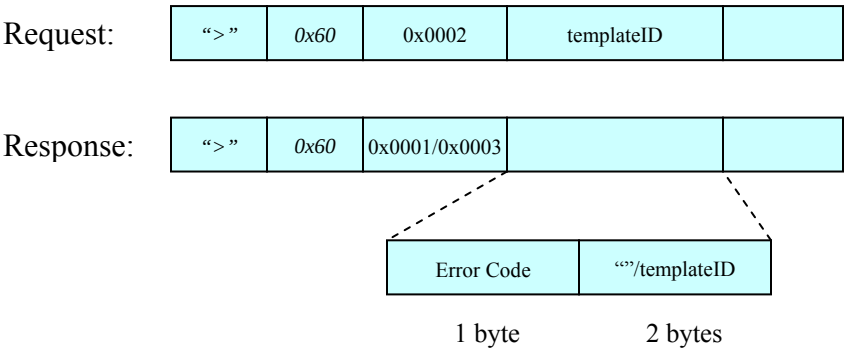
- *Error Code = 0x00*- No error

*Timeline*



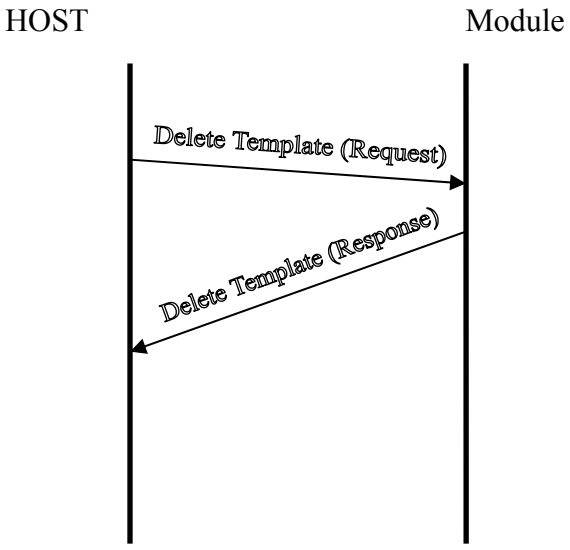
## *Delete Template (0x60)*

Deletes permanently given template from database. This is an irreversible process. The templateID (2 bytes long) is provided by the HOST in the data field in the *request* packet. The successfully deleted templateID is sent to the HOST by the *response* packet. Values above 0xFFF0 are reserved to indicate error during enrolment so should not be used by the HOST in the *request* packet. Note that templateIDs above 65500 are reserved for system purpose and should not be sent by the HOST.




- *Error Code = 0x00*- No error
- *Error Code = 0x05*– Invalid Value for *templateID*
- *Error Code = 0x80*–FLASH memory write error
- *Error Code = 0x50*–Template not found

*Timeline*



## *Delete Whole Database (0x61)*

Delete permanently the whole database. This is an irreversible process. On successfully deletion the module sends to the HOST a *response* packet with 0x00 in the data field. If data field is nonzero then error has occurred.

Request:

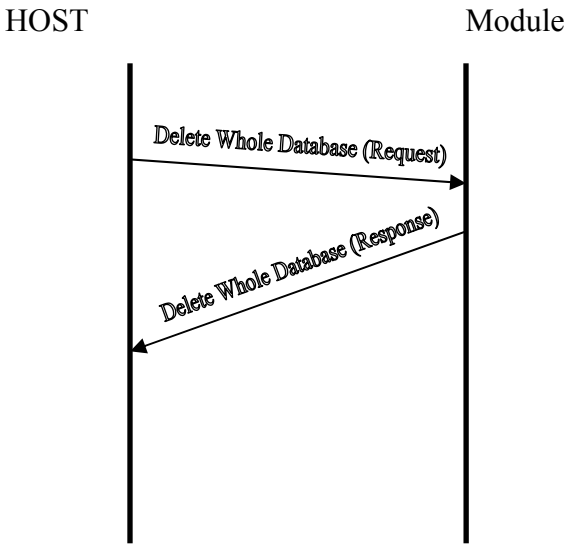
| | | | | |
|---|---|---|---|---|
| ">" | *0x61* | 0x0000 | "" | 0x9F |

Response:

| | | | | |
|---|---|---|---|---|
| ">" | *0x61* | 0x0001 | Error Code | |

- *Error Code = 0x00*- No error
- *Error Code = 0x80*–FLASH memory write error

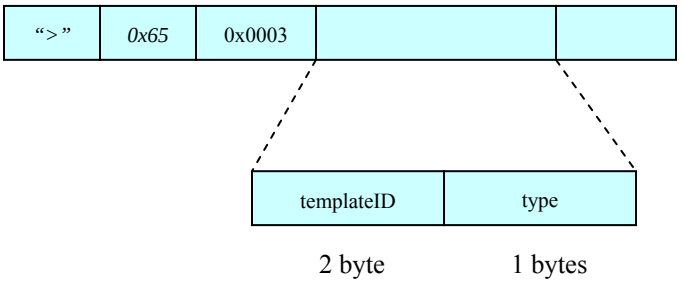
*Timeline*




## *Master Template (0x65)*

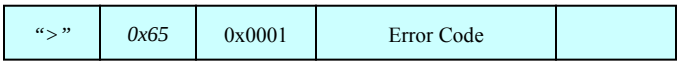
Assigns//clears master property to a given fingerprint template (templateID). Master templates are used to authorized some special procedures like fingerprint enrolment. The concept of master fingerprints is very useful in autonomous (host-less operation of the module). There can be more than one master template in the database. On successfully execution the module sends to the HOST a *response* packet with 0x00 in the data field. If data field is nonzero then error has occurred.

Request:



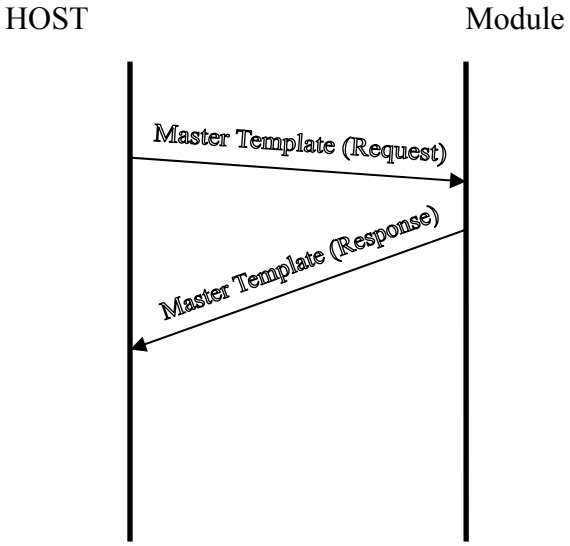

- *type*=0 The fingerprint template is set to be normal fingerprint template. It can not be used to grand access to enrolment procedure anymore.
- *type=1*- the specified template is set to be master. It can be used to grand access to enrolment in autonomous operation.

Response:




- *Error Code = 0x00*- No error
- *Error Code = 0x05*– Invalid Value for *templateID*
- *Error Code = 0x80*–FLASH memory write error
- *Error Code = 0x50*–Template not found

*Timeline*



## 2.5 Dump images

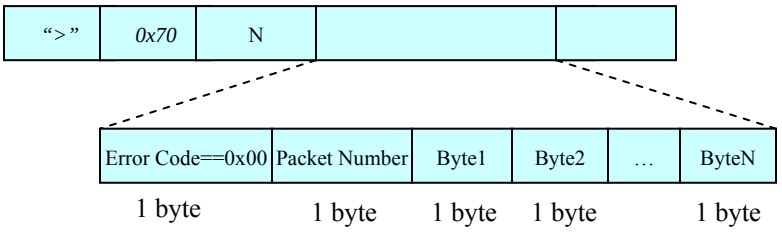
### *Dump Grayscale Image (0x70)*

The module dumps the grayscale fingerprint image captured (or will be captured) from the sensor. The image is sent to the HOST using several about 1KBytes long *response* packets. After each *response* packet a *conformation* packet (from the HOST to the module) is expected before next *response* packet. If the confirmation packet indicates that the previous *response* packet was not successfully received by the HOST the module resends it. If the *response* packet (with data) is sent but no *confirmation* from the HOST arrived within certain amount of time the module terminates the image transmission and sends *response* with timeout error. In the last *response* packet the module sets appropriate error code which indicates the end of image transmission. During fingerprint processing the grayscale image is processed too. A parameter in the *request* packet defines which grayscale type (GStype) is requested.

Request:


- *GStype=0* – The enhanced grayscale image corresponding to previous finger impression is returned. (Only this mode is implemented in this version of the protocol)
- *GStype=1*- Next raw grayscale image (just from the sensor) will be returned as soon as there is a finger for processing on the sensor. (Currently not supported)
- *GStype=2*- Next enhanced image will be returned as soon as there is a finger for processing on the sensor. (Currently not supported)

Response:




- *Error Code = 0x00*- No error
- *Error Code =0xA0* Last packet for image transmission

Response:

| “>” | *0x70* | N | Error Code <> 0x00 | |
|---|---|---|---|---|

- *Error Code* = 0x05– Invalid *GStype* value
- *Error Code = 0xA2* - There is no valid image to send.

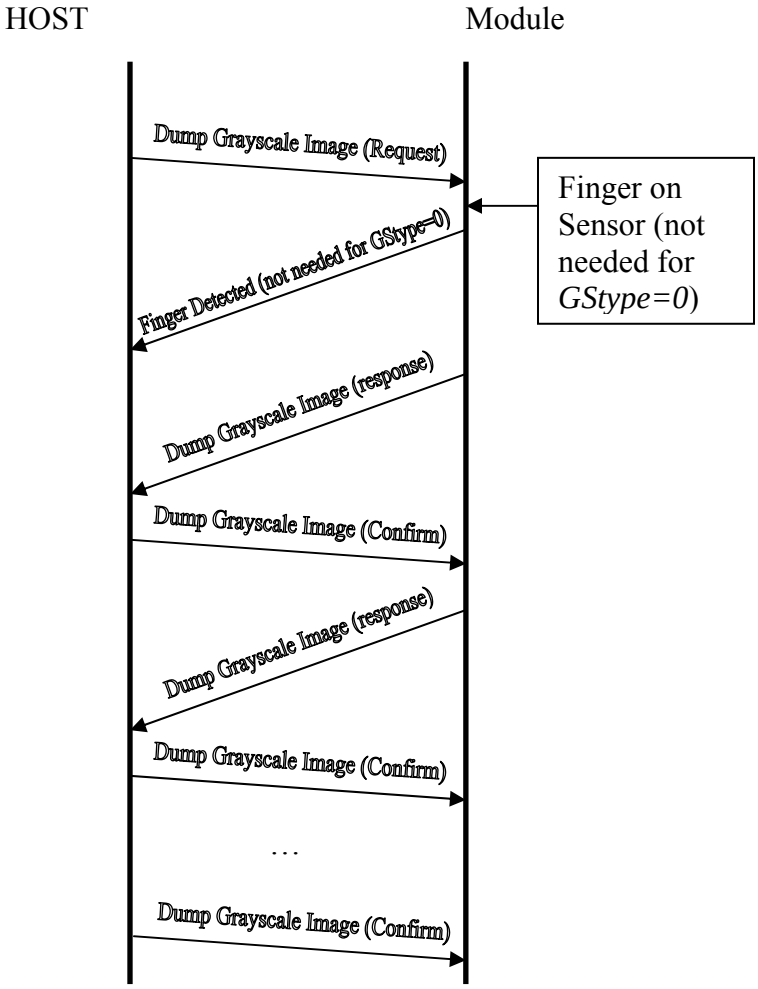
Confirm:

| “>” | *0x70* | 0x02 | | |
|---|---|---|---|---|

| Error Code | Confirm for packet number |
|---|---|
| 1 byte | 1 byte |

- *Error Code = 0xA1*- check sum error in the image data packet detected by the HOST.

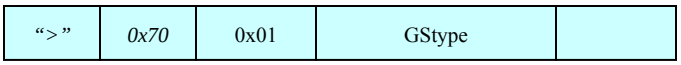
*Timeline*



## *Dump Binary Image (0x75)*

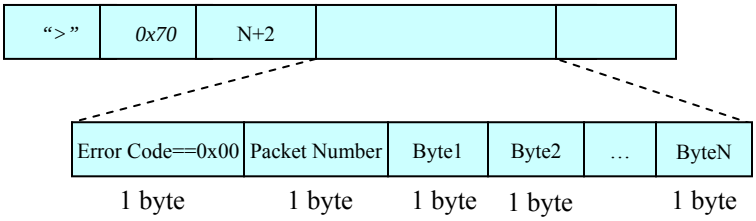
The module dumps the binary fingerprint image from the module. Black pixel is represented using byte with value 0x00 and white with 0xFF. The image is sent to the HOST by several *response* packets each about 1KBytes long. After each *response* packet a *conformation* packet (from the HOST to the module) is expected before next *response* packet. If the confirmation packet indicates that the previous *response* packet was not successfully received by the HOST the module resends it. If the *response* packet (with data) is sent but no *confirmation* from the HOST arrived within certain amount of time the module terminates the image transmission and sends *response* with timeout error. In the last *response* packet the module sets appropriate error code which indicates the end of image transmission. During fingerprint processing the grayscale image is processed too. A parameter in the *request* packet defines which grayscale type (GStype) is requested.

Request:




- *GStype=0* – The final binary image corresponding to previous finger impression is returned. (Only this mode is implemented in this version of the protocol)
- *GStype=1*- Next (after new fingerprint on sensor is available) binary image just after binarization. (Currently not supported)
- *GStype=2*- Next (after new fingerprint on sensor is available) final binary image. (Currently not supported)

Response:




- *Error Code = 0x00*- No error
- *Error Code =0xA0* Last packet for image transmission

Response:

| “>” | *0x70* | 0x01 | Error Code <> 0x00 | |
|---|---|---|---|---|

- *Error Code = 0xA2*- There is no valid image to send

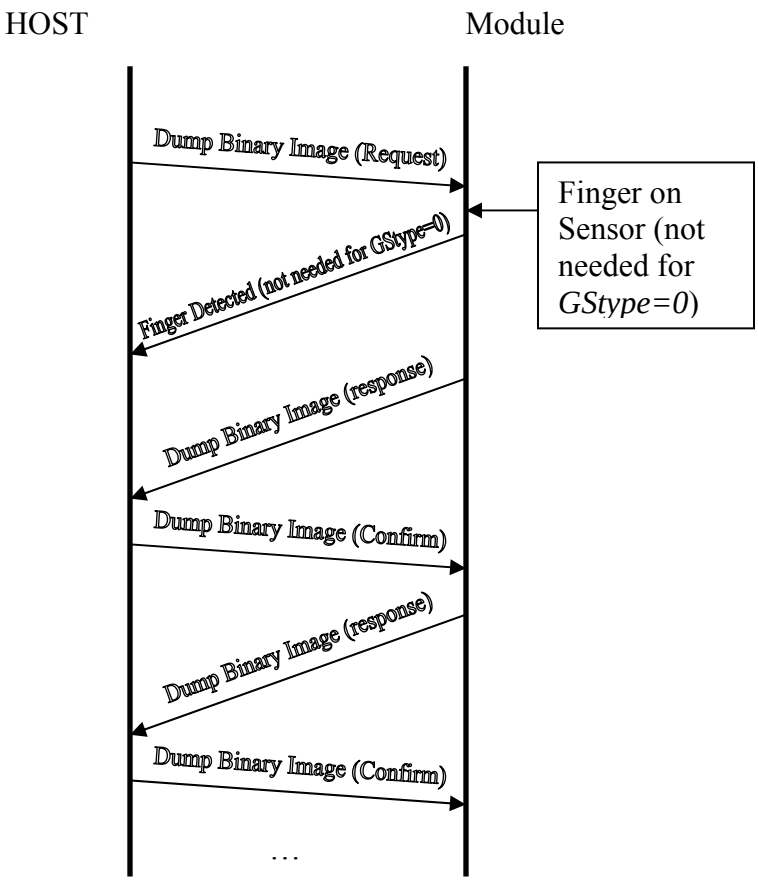
Confirm:

| “>” | *0x70* | 0x02 | | |
|---|---|---|---|---|

| Error Code | Error code is for packet number |
|---|---|
| 1 byte | 1 byte |

- *Error Code -0x00* – No error
- *Error Code = 0xA1*- check sum error in the image data packet detected by the HOST.

*Timeline*

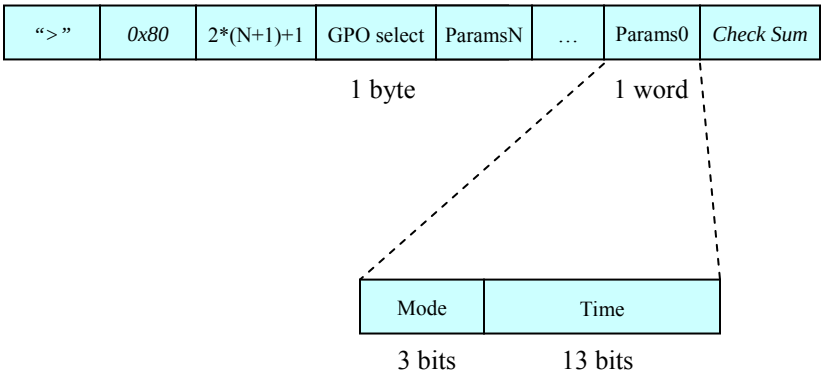


## 2.6 User Interface

### *Set GPO (0x80)*

The fingerprint module contains 3 LEDs (red, green, yellow), 1 buzzer output (BUZ) and 3 general purpose outputs (GPO0 andGPO1). The state of these outputs can be set using the data field in the *request* packet. For example, to set GPO1 in logical high use 1 in the appropriate bit of the data field of this command. A reserved bit is ignored by the module.

Request:




- *GPO select* – Each of the 6 least significant bits of this field corresponds to one of the GPO. Bit0 to Green, Bit1 to Red, bit2 to Yellow, bit3 to Buz, bit4 to GPO0 and bit5 to GPO1. If a bit is 1 there should be a parameter field for the selected general purpose output. There are no parameter fields for other GPOs. The number of the parameters should be equal to the number of ones in *GPO select.* The parameter fields are ordered similarly as the order in *GPO select* field

- *Time* – Time in *ms* for the 'single shot' mode. This parameter is not used for other modes. Max 8.191 s can be specified.
- *Mode*- Defines the mode of the specified GPO.

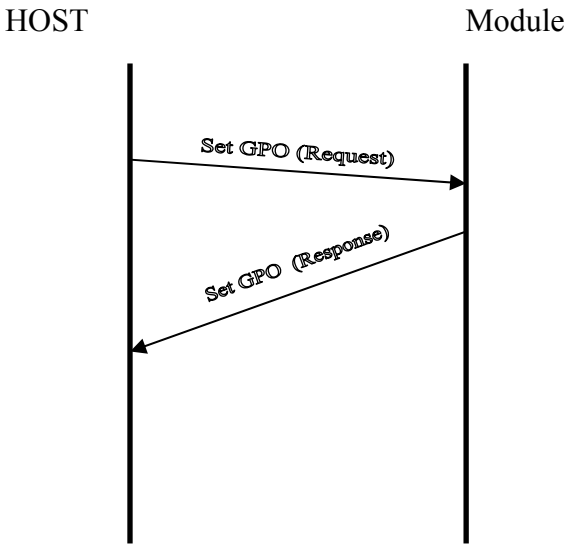
| *Value[bits]* | *Description* |
|---|---|
| *000* | *Off* |
| *001* | *Slow blinking* |
| *010* | *Fast blinking* |
| *011* | *On* |
| *100* | *Change* |
| *101* | *Single shot (lits for the time as it is defined in Time field)* |

Response:

| | | | | |
|---|---|---|---|---|
| “>” | *0x80* | 0x0001 | Error Code | |

- *Error Code=0x00 – No error*
- *Error Code = 0x05*– Invalid Value for Size, *GPO* or *Mode*

Timeline



## *Beep (0x83)*

One can control the buzzer using the '*Set GPO*' command but to simplify the sound generation an extra *'Beep'* command is implemented. It generates several sounds using the buzzer which is built in the fingerprint module.

Request:

| | | | | |
|---|---|---|---|---|
| ">" | *0x83* | 0x0001 | Sound | |

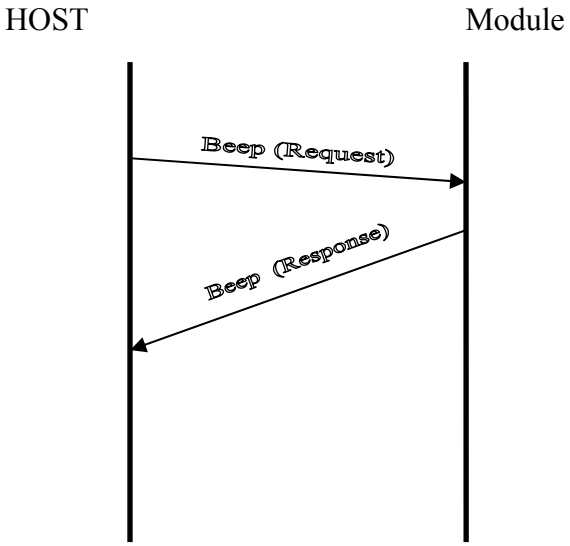
- *Sound=0x00 – "OK"* sound, two beeps 128ms apart.
- *Sound=0xFF – "CANCEL*" sound, 512ms single beep.

Response:

| | | | | |
|---|---|---|---|---|
| ">" | *0x83* | 0x0001 | Error Code | |

- *Error Code=0x00* – No error
- *Error Code=0x05*- Invalid sound

Timeline



## *Get GPI (0x85)*

The fingerprint module has two general purpose inputs (GPI0 and GPI1) and three buttons (Identify, Enroll and Delete). The state of these inputs can be read in the data field of the *response* packet as it is illustrated below. The data in the "Reserved" field is set to 0 by the module.

Request:

| "> " | *0x85* | 0x0000 | "" | 0xC3 |
|---|---|---|---|---|

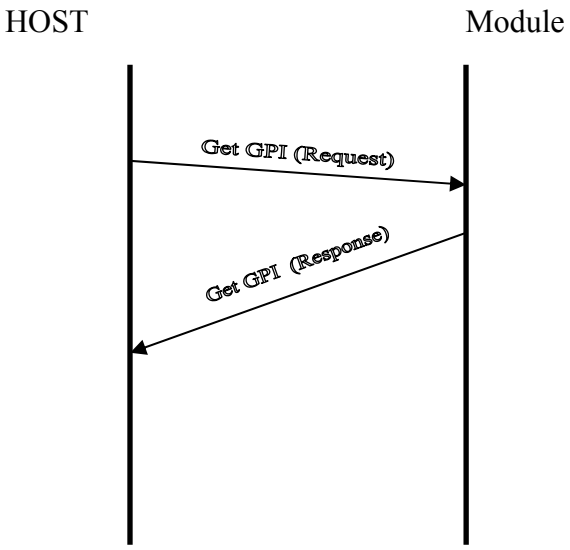
Response:

| "> " | *0x85* | 0x0002 | | |
|---|---|---|---|---|

| Error Code | Reserved | GPI1 | GPI0 | Delete State | Enroll State | Identify State |
|---|---|---|---|---|---|---|
| 1 byte | 2 bits | 1 bit | 1 bit | 1 bit | 1 bit | 1 bit |

- *Error Code=0x00 – No error*

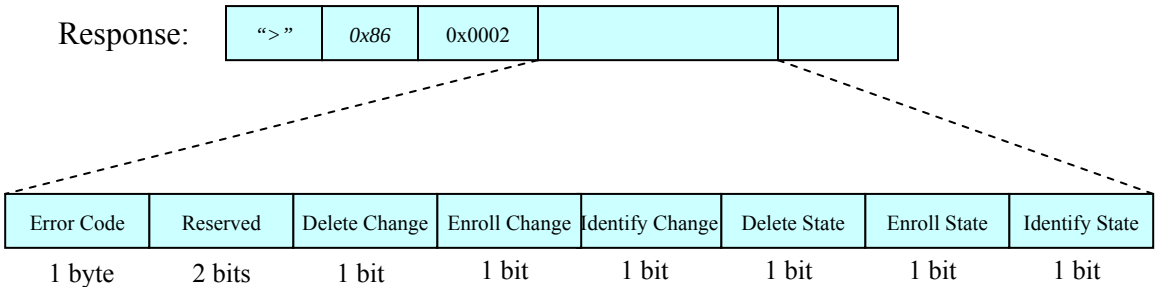
Timeline



## *Buttons State (0x86)*

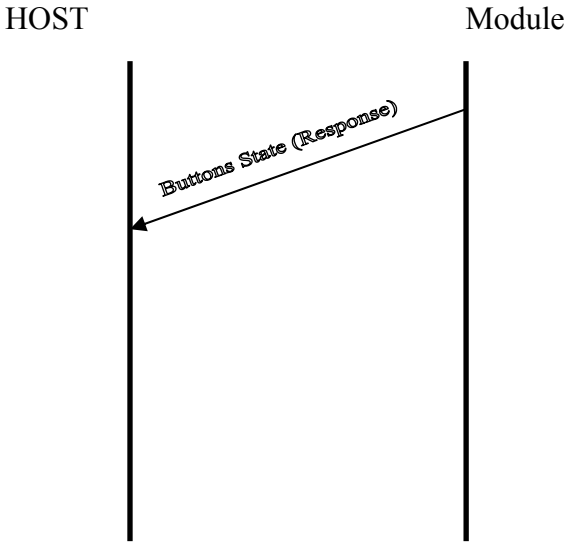
There are 3 buttons implemented in the fingerprint module (Identify, Enroll and Delete). In case one or more buttons is pressed the notification packet is sent to the HOST. In the data field of this command there are both change and state information included. The HOST should not respond to this packet. The data in the "Reserved" field is set to 0 by the module.

Response:




- *Error Code=0x00 – No error*

Timeline



# 3.0 Contact Info

## Headquarters:

**BioIdentic, Inc.**

**Web: http://www.bioidentic.com**

**Anna Ahmatova Str., bl. 519, ap. 4, Sofia, Bulgaria**

**Phone**: +359898652775

**Emails: info@bioidentic.com**
**support@bioidentic.com**
**sales@bioidentic.com**