1520 Dell Ave
Campbell, CA 95008

Recognition Systems, Inc.

FingerKey Network Command Specification

Command subset as it applies to the HandNet-Lite Software Product, phase 1

Version 0.27

FingerKey Network Command Specification

Command subset as it applies to the HandNet-Lite Software Product, phase 1

Forward

The purpose of this document is to state the commands used by the *HandNet-Lite* software product and describe those commands and their associated data structures in details. The *HandNet-Lite* software product uses many but not all of the Network commands available in the FingerKey terminal (henceforth called the FKT).

Network commands (just "commands" from now on) originate from many possible sources. The Network Command Parser inside of the FKT will always see the same data stream regardless of the physical source of data. The following list shows the different command sources:

1. RS232 port
2. RS485 port (½ duplex)
3. Ethernet port (10/100bT) – TCP/IP
4. IrDA port – most likely TinyTP

Normally only one of these sources will be active at a time. It may be possible to have the RS232/422/485 port, the Ethernet port, and the IrDA port all active and receiving commands at the same time but this is not something we actually want to deal with. It is likely that we will enforce a single "Host" source rule in the firmware. In any event, regardless of the source, the data will be formatted the same. This greatly simplifies the firmware. **Reader should review the Command/Response Protocol Structure in Appendix B before proceeding to understand the command details**.

HandNet-Lite Goal

HandNet-Lite is a software package running on a PC that controls fingerprint readers. The main functions of *HandNet-Lite* are to:

- Configure a network of fingerprint readers and set their configuration properties.
- Manage the users known to the network and the individual fingerprint readers through access profiles.

- The distribution of fingerprint biometric templates to specific readers according to access profiles.

See the *HandNet-Lite* Software Specification and *HandNet-Lite* clarification document for more details.

What we're attempting to do here is list those commands in the FKT that are most likely going to be needed by *HandNet-Lite*.

Command List

The Host and the FKT communicate through a Command-Response pair respectively. Commands are named from the Host point of view, i.e., GET means the Host will retrieve something from the FKT; SET something means the Host will send data to the FKT. When a Host command can be and is carried out, the FKT acknowledges with a positive response of the command performed, either in the form of requested data to be returned or just the command being carried-out successfully, piggy-backed with the current system status bytes. Otherwise, for one reason or another, when the Host command cannot be carried out, the FKT issues a NAK response accompanied by a reason (See Appendix A for details). The following commands will be used by Sierra.

1. Status Poll
2. Status Clear
3. Get / Put User Record
4. Delete User Record
5. Clear User Database
6. Get / Set Setup
7. Idle / Resume
8. Get Reader Info
9. Get / Set Time
10. Beeper control
11. LED control
12. Put Card Format
13. Delete Card Format
14. Clear Card Format Database
15. Get Next Card Format
16. FKT (Soft) Reset
17. FKT Diagnostic Command

The following commands *will NOT* (initially) be available for/in *HandNet-Lite, phase 1*:

18. Remote Enroll
19. Remote Verify

Command Details

The following sections detail each of the commands listed above.

Status Poll (0x30)

The Status Poll command is used to determine the current state of the device, whether there are datalogs present, as well as other yet-undetermined-things. This command is based on the HandReader command, but differs in that many of the bits returned by a HandReader have no analog in the FKT. Note that this FingerKey model does NOT have datalogs. We're reserving space in the response for this feature as it will be required in future models.

StatusPoll (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x30*** StatusPoll command
Total Data length	1*		

The response packet from the FKT contains the command type and 3 status bytes in its Data field. Currently, only the SysStat0 byte is defined and their bit definitions are detailed in **Table 2**. SysStat1 and SysStat2 bytes are undefined and are reserved for future expansion. The format of the Data field in response packet (from FKT) is as follows:

StatusPoll (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x30*** StatusPoll command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Table 2: **SysStat0 Bit Definitions**

Label	Bit	Mask	Description
FKT_COLDBOOT	0	01H	Set when FKT is completely initialized to its default/factory configuration via the ColdBoot button being pressed during power up. Cleared by StatusClear command (Host uses this bit to determine to reload the user database, if necessary).
FKT_LOCKOUT_FOR_HOST	1	02h	Set when Idle command is received and the FKT is able to go to Idle mode to lockout local activities for the Host. Cleared when the Resume command is received, or when Idle/autoResume timeout expires.

TAMPER	2	04H	This bit indicates whether a tamper has occurred and will be software-latched so that the Host has an opportunity to read it. Subsequently, the Host can send a StatusClear command to clear this field, and the tamper switch will be sensed again.
DBASE_FULL	3	08H	Set when the user database is full. Cleared when one or more user database slots become available.
DBASE_EMPTY	4	10H	Set when the user database is empty. Cleared when one or more user database slots are filled.
USER_REC_MOD	5	20H	Set when there is at least 1 new/modified user record, deleted user record, or database erased at FKT. Cleared when StatusClear command is received. See Note 1 in GetNextUserRecord section for operation details
DURESS_DETECTED	6	40H	A duress condition occurs at the FKT. The Host is notified with this bit field via the StatusPoll command, and can subsequently clear this field via the StatusClear command.
SysStat0_Bit7	7	80H	TBD

Status Clear (0x31)

The Status Clear command is used to clear a status bit in one of the status bytes. This is used to ensure that the Host and FKT have a full handshake in the status bits notification.

StatusClear (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x31*** StatusClear command
Status Clear Type	1*	1*	**0*** = Clear FKT_RESET bit in SysStat0 byte **1*** = Clear USER_REC_MOD bit in SysStat0 byte, see section Note 1. **2*** = Clear TAMPER bit in SysStat0 byte, see section Note 2. **3*** = Clear DURESS_DETECTED bit in SysStat0 byte. (**4-255**)* = TBD
Total Data length	2*		

The FKT returns a positive response when the command is carried out successfully, with the format of the Data field in the response packet as follows:

StatusClear (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x31*** StatusClear command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

When the StatusClear command cannot be carried out, for example an undefined Status Clear Type is requested, the FKT responds with a NAK accompanied by a reason (See Appendix A for details).

Note 1: USER_REC_MOD bit in SysStat0 will be re-scanned when Status Clear Type (1) is carried out to prevent a premature clearing of this bit when there are still un-retrieved new/modified user records.

Note 2: TAMPER bit in SysStat0 will be re-scanned when Status Clear Type (2) to reflect the latest condition of the tamper switch.

User Records

This is the current FingerPrint user record data structure. It consists of a Header1 data structure and 2 Fp_Template data structures. These data structures are used in the GetUserRecordByID, GetNextUserRecord, and PutUserRecord commands and responses. The data structures are detailed as follows:

typedef unsigned char **Fp_Template**[800];

```
typedef struct
{
    unsigned char id[26];          // ID is key for record search, 25 digits + Null string terminator.
                                   // ID entered from keypad is 15 digits, whereas ID from Wiegand can be
                                   // up to 25 digits max.
    unsigned char header1_flags;   // See Table 1 for bit definitions
    unsigned char threshold;       // 0 d - 255. Threshold level for biometric match, scaled to be similar
                                   // to HandReader threshold.
                                   // 0 d: Use global fingerkey reader threshold (in Setup)
    unsigned char authority;       // Permission level for product menu tree
                                   // 0 d: None,
                                   // 1: Service, 2: Setup, 3: Management, 4: Enrollment, 5: Security
    unsigned char timezone;        // 0 d.  Timezone table index; not used in this model
    BYTE effective[8];             // Date time User Record is effective in Host database
    BYTE expires[8];               // Date time User Record is expired in Host database
} Header1;

typedef struct
{
    Header1 header_1;

    Fp_Template template_1;
    Fp_Template template_2;
} Fp_UserRecord;
```

Table 1: **header1_flags (in struct Header1) bit definitions**

Label	Bit(s)	Mask	Description
User Record Status	0	01H	**1** = User record is new or has been modified **0*** = User record has NOT been changed **FKT**: RW. **Host**: RW.
No Finger Verify	1	02H	**1** = User verification does not require biometric match **0*** = User verification requires biometric match **FKT**: RW. **Host**: RW.
Second Finger As Duress	2	04H	**1** = Use second fingerprint template as duress **0*** = Use second fingerprint template as normal **FKT**: RW. **Host**: RW.
Template Count	4, 3	18H	Fingerprint template counts in this user record **00***, **01**, **10** = zero, 1, 2 fingerprint templates **11** = Undefined **FKT**: RW. **Host**: RW.
Reserved	5	20H	TBD
Reserved	6	40H	TBD
Reserved	7	80H	TBD

Get User Record by ID (0x32)

This command is used to get a user record by ID number. It looks up and possibly returns the user record with the indicated ID (Null-terminated string), depending on "what" is specified. This command should be bracketed by the Idle command and the Resume command to ensure the FKT is idle before sending the ID and "what" fields. Note that the user-record-attribute-mask of the "what" field (See Table 3) is ignored in the get user record by ID command; only the user-record-portion-mask of the "what" field is relevant.

GetUserRecordByID (command packet from Host) is as follows:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x32*** GetUserRecordByID command
ID	26*	1*	25 ID digits + 1 Null string terminator
what	1*	27*	what fields of the user record to be returned. See Table 3.
Total Data length	28*		

If the User ID exists, the FKT returns the portion (specified by "what") of the user record. If the "what" portion of the user record to be returned fits in 1 packet fragment, the format of the Data field in the first (and also last) GetUserRecordByID response packet (from FKT) is as follows:

GetUserRecordByID (first response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x32*** GetUserRecordByID command
User Record Data	n*	1*	The length and type of user record data returned depend on the "what" field in the command received. See Table 3.
Total Data length	(1 + n)*		**Max-Data-field-payload** (**632***) bytes or less

If the "what" portion of the user record to be returned spans multiple packet fragments, the Host should issue a zero-length command (see Appendix B) after receiving the first packet fragment to signal the FKT for the next packet fragment, the format of the Data field for the subsequent and the last GetUserRecordByID response packet fragment is as follows:

GetUserRecordByID (subsequent and last response packet from FKT) Data field format:

Field	Bytes	Offset	Description
User Record Data	n*	0*	Remaining user record data. The length and type of user record data returned depend on the "what" field in the command received. See Table 3.
Total Data length	n*		**Max-Data-field-payload** (**632***) bytes or less

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the User ID does not exist, or the database is empty, the FKT issues a NAK response accompanied by a reason (See Appendix A for details).

Get Next User Record (0x33)

This command is used to get the next user or the next new/modified user record in the database, possibly increment the user database iterator and return the next user record, depending on "what" is specified. Note: The type of (any or new/modified) user record returned depends on which initial ResetIterator sent from the Host previously. The series of this command should be bracketed by the Idle command and the Resume command to ensure the FKT is idle to perform this (possibly) lengthy operation.

GetNextUserRecord (command packet from Host) Data field format is as follows:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x33*** GetNextUserRecord command
Sequence Number	2*	1*	The Sequence number is simply an unique number used for positive acknowledge between the Host/Master FKT and the remote FKT. It may or may not have any relationship with how the user database is tracked internally. See section Note 2, and Tables 4 and 5 for usage in various scenarios. **0xFFFF***: Request FKT to Reset its internal iterator type (per the user-record-attribute-mask in the "what" field that follows). **Other value***: The Host receives the previous user record with no error, and the FKT is to return the next user record.
what	1*	3*	what fields of the user record to be returned. See Table 3.
Total Data length	4*		

If the database is not empty, the FKT returns the portion (specified by "what") of the user record. If the "what" portion of the user record to be returned fits in 1 packet fragment, the format of the Data field in the first (and also last) GetNextUserRecord response packet (from FKT) is as follows:

GetNextUserRecord (first response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x33*** GetNextUserRecord command
User Record Data	n*	1*	Remaining user record data. The length and type of user record data returned depend on the "what" field in the command received. See Table 3.
Total Data length	(1 + n)*		**Max-Data-field-payload** (**632***) bytes or less

If the "what" portion of the user record to be returned spans multiple packet fragments, the Host should issue a zero-length command (see Appendix B) after receiving the first packet fragment to signal the FKT for the next packet fragment, the format of the Data field for the subsequent and the last GetNextUserRecord response packet fragment is as follows:

GetNextUserRecord (subsequent and last response packet from FKT) Data field format:

Field	Bytes	Offset	Description
User Record Data	n*	0*	Remaining user record data. The length and type of user record data returned depend on the "what" field in the command received. See Table 3.
Total Data length	n*		**Max-Data-field-payload** (**632***) bytes or less

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the database is empty or it is end of record retrieval, the FKT issues a NAK response accompanied by a reason (See Appendix A for details).

When the "what" portion of a user record is returned, it is returned with a unique sequence number prefixed to it as follows:

```
struct GetUserRecordHeader1Response
{
    uint16 sequence_num;      // Unique sequence number for positive acknowledgement
    struct Header1 header_1;
}

struct GetUserRecordResponse
{
    uint16 sequence_num;      // Unique sequence number for positive acknowledgement
    struct Fp_UserRecord UserRecord;
}

struct GetUserRecordOneTemplateResponse
{
    uint16 sequence_num;      // Unique sequence number for positive acknowledgement
    struct Fp_Template template;
}

struct GetUserRecordBothTemplatesResponse
{
    uint16 sequence_num;      // Unique sequence number for positive acknowledgement
    struct Fp_Template template1;
    struct Fp_Template template2;
}
```

The "what" field is divided into upper and lower-nibble fields. The lower-nibble field specifies the type of user record, and the upper-nibble field specifies the portion of the user record to be returned. The type and portions of the user record to be returned or to be set are according to the following table:

Table 3: "**what**" **Field Definitions**

Bits	What is Returned (Get) or Set
0..**3**	**User Record Attribute Mask** **0** = Any user record to be returned. **1** = Modified user record ONLY to be returned. **2**..**15** = TBD
4..7	**User Record Portion Mask** **0** = Entire user record (**GetUserRecordResponse**, **SetUserRecord**) **1** = Only the Header_1 (**GetUserRecordHeader1Response**, **SetUserRecordHeader1**) **2**..**15** = TBD

This table may be modified as needed to make the most sense. Perhaps there are only two cases that really need consideration: the entire user record and just the header portion.

Note 1: USER_REC_MOD usage.

When the USER_REC_MOD bit is set in SysStat0 byte, the Host is alerted of this bit by a periodic StatusPoll command to the FKT. For example, to retrieve all modified (Entire) user record(s), the Host will need to initially request the FKT to go into Idle mode to disable any future activities at the local FKT. It then issues the GetNextUser <0xFFFF> <what = 0x01> command to reset the iterator to the first new/modified user record in the user database, and subsequently issues the GetNextUser <0xnnnn> <what=0x01> command to retrieve the next new/modified user record in the database until there is no more new user record, at which time the FKT will respond with a NAK. Finally, the Host will issue the StatusClear to clear the USER_REC_MOD bit in SysStat0 byte, and then the Resume command to tell the FKT to resume local operation.

Note 2: GetNextUserRecord command usage.

The GetNextUserRecord is used to get (modified or just any) user record(s) in a series. The sequence starts by the Host getting the first (modified or any) user record, and followed by getting the next (modified or any) user record(s) until the end of user record is reached in the database. These scenarios are summarized in **Table 4** and **5** respectively.

Table 4: **Get First (modified or any) User Record Scenarios with the GetNextUser Command**

Scenario	Host/Master FKT	Slave FKT
4.**1**	Get First user record ----------> (GetNextUser<**0**x**FFFF**><what>) Response received okay (Go to (**5**.**1**))	Command received okay. It is a reset to get the First (modified or any) user record. <---------- Responds with First user record (<**GetNextUserRecord**><**SN**$_i$><data…>)
4.**2**	Get First user record ----------x (GetNextUser<**0**x**FFFF**><what>) Response not received (Timeout) (Go back to (**4**.**1**) to request again)	Command not received at all or CRC error

4.3	Get First user record ----------> (GetNextUser<**0**x**FFFF**><what>) Response not received (Timeout) or CRC error (Go back to (**4.1**) to request again)	Command received okay. It is a reset to get the First (modified or any) user record. x---------- Responds with First user record (<**GetNextUserRecord**><**SN**$_i$><data…>)
4.4	Get First user record ----------> (GetNextUser<**0**x**FFFF**><what>) Response received okay. Host stops sending GetNextUser command.	Command received okay. But user database is EMPTY. <---------- Responds with **NAK** (<**NAK**><**GetNextUserRecord**> <**DbaseEmpty**>)

Table 5: **Get Next (modified or any) User Record Scenarios with the GetNextUser Command**

Scenario	Host/Master FKT	Slave FKT
5.1	Get Next user record ----------> (GetNextUser< **SN**$_{i,j,k,...}$><what>) Response received okay (Repeat (**5.1**) with **SN**$_{j,k,...}$	Command received okay. Previous response is confirmed being received. <---------- Responds with Next user record (<**GetNextUser**><**SN**$_{j,k,...}$><data…>)
5.2	Get Next user record ----------x (GetNextUser< **SN**$_{i,j,k,...}$><what>) Response not received (Timeout) (Go back to (**5.1**) to request again)	Command not received at all or CRC error
5.3	Get Next user record ----------> (GetNextUser< **SN**$_{i,j,k,...}$><what>) Response not received (Timeout) or CRC error (Go back to (**5.1**) to request again)	Command received okay. Previous response is confirmed being received. x---------- Responds with Next user record (<**GetNextUserRecord**>< **SN**$_{j,k,...}$> <data…>)
5.4	Get Next user record ----------> (GetNextUser< **SN**$_{i,j,k,...}$><what>) Response received okay. Host can decide how to proceed, for example, back to (**4.1**) again to reset to beginning of user record database.	Command received okay. But sequence number is mismatched. <---------- Responds with **NAK** (<**NAK**><**GetNextUserRecord** > <**SeqNumMismatched**>)
5.5	Get Next user record ----------> (GetNextUser< **SN**$_{i,j,k,...}$><what>) Response received okay. Host stops sending GetNextUser command.	Command received okay. But, it is the end of the (modified or any) user record in database. <---------- Responds with **NAK** (<**NAK**><**GetNextUserRecord** > <**EndOfUserRecord**>)

Where **SN**$_i$, **SN**$_j$ and **SN**$_k$ are unique sequence numbers assigned during user record retrieval, and are such that $T_i < T_j < T_k < \ldots$ over the retrieval time.

Put User Record (0x34)

This command is very similar to the Get User Record By ID command in reverse… This command should be bracketed by the Idle command and the Resume command to ensure the FKT is idle before sending the "what" and User Record Data fields; the Idle command also serves as an indicator to the FKT firmware NOT to mark the User_Record_Status bit in the header1_flags (in Header1 data structure) as having been modified, ie header1_flags will be updated exactly as it is being received.

PutUserRecord command sets the user record with the indicated ID (Null-terminated string) depending on "what" is specified. Note that the user-record-attribute-mask of the "what" field (See Table 3) is ignored in the put user record by ID command; only the user-record-portion-mask of the "what" field is relevant.

```
struct PutUserRecordHeader1
{
    struct Header1 header_1;
}

struct PutUserRecord
{
    struct Fp_UserRecord UserRecord;
}
```

If the "what" portion of the user record to be set fits in 1 packet fragment, the format of the Data field in the first (and also last) PutUserRecord command packet (from Host) is as follows:

PutUserRecord (first command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x34*** PutUserRecord command
what	1*	1*	what fields of the user record are to be set by the FKT
User Record Data	n*	2*	The length and type of user record data to be set depend on the "what" field specified in the command received. See Table 3.
Total Data length	(2 + n)*		**Max-Data-field-payload** (**632***) bytes or less

If the "what" portion of the user record to be set spans multiple packet fragments, after the sending the first packet fragment, the Host should wait for a zero-length response (see Appendix B) from the FKT to signal for the next packet fragment, the format of the Data field in the subsequent and the last command packet fragments is as follows:

PutUserRecord (subsequent and last command packet from Host) Data field format:

Field	Bytes	Offset	Description
User Record Data	n*	0*	Remaining user record data. The length and type of user record data returned depend on the "what" field in the command received. See Table 3.
Total Data length	n*		**Max-Data-field-payload** (**632***) bytes or less

After receiving the last packet fragment, if the user record does not exist, it will be added only if the "what" field specifies "Entire User Record"; otherwise, if the user record exists, it will be updated per "what" is specified. The FKT will return a response after the user record is successfully stored, with the format of the Data field in the response packet as follows:

PutUserRecord (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x34*** PutUserRecord command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

If the PutUserRecord command cannot be carried out at all, for example if the user database is FULL, the FKT responds with a NAK accompanied by a reason (See Appendix A for details).

Delete User Record By ID (0x35)

This command deletes an entire user record per user ID.

DeleteUserRecordByID (Command) Data format: Delete the user record with the indicated Null-terminated ID. This command should be bracketed by the Idle command and the Resume command to ensure the FKT is idle before sending the ID field.

DeleteUserRecordByID (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x35*** DeleteUserRecordByID command
ID	26*	1*	25 ID digits + 1 Null string terminator
Total Data length	27*		

DeleteUserRecordByID (Response) Data format: If the user record exists, the command will be carried out, and the FKT responds with the format of the Data field in the response packet as follows:

DeleteUserRecordByID (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x35*** DeleteUserRecordByID command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the user record does NOT exist, or the database is empty, the FKT will return a NAK accompanied by a reason (See Appendix A for details).

Clear User Database (0x36)

This command erases the entire user database in the FKT and commands the Fingerprint sensor to erase its database as well. This command should be bracketed by the Idle command and the Resume command to ensure the FKT is idle. Note that this command currently can take as long as 40 seconds before the FKT responds back to the Host.

ClearUserDatabase (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x36*** ClearUserDatabase command
Total Data length	0		

If the command is carried out, and the FKT responds with the format of the Data field in the response packet as follows:

ClearUserDatabase (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x36*** ClearUserDatabase command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Get Setup (0x37)

This command retrieves the reader Setup information from the FKT. This Setup data structure is currently padded to a 256-bytes data structure and its members are defined as below with default values marked by [d]:

```
typedef struct
{
    uint16   setupDataLen;              // Length of actual bytes used in setup_data structure
    BYTE     service_password[11];      // 10 ASCII digits + 1 Null string terminator ("1"[d])
    BYTE     setup_password [11];       // ("2"[d])
    BYTE     management_password[11];   // ("3"[d])
    BYTE     enrollment_password[11];   // ("4"[d])
    BYTE     security_password[11];     // ("5"[d])
    uint16   threshold;                 // 0-255 (63[d] = EER)
    BYTE     idLength;                  // sets maximum ID length to 25 digits + 1 Null string terminator
    BYTE     secondary_finger_mode;     // 0: Disabled.
                                        // 1[d]: As alternate finger enrollment/verification
                                        // 2: As duress signal.
    BYTE     numberOfTries;             // 0, 1, 2, 3[d], 4, 5, number of times an ID can be rejected before
                                        // being locked out.
    uint16   autoResumeTimeout          // 60 .. 300[d] .. 65535, FKT auto-resume timeout (in seconds)
                                        // if timeout value in Idle command is 0 or, the Host
                                        // fails to send the Resume command.
```

```
Char    readyStr[21];          // The Null-terminated ASCII string to be
                               // displayed on the front panel.
BYTE    beeperEnable;          // 0: Disabled, 1 d: Enabled
BYTE    extLed;                // 0 d: LED controlled internally, 1: LED controlled externally
BYTE    extBell;               // 0 d: Bell controlled internally,  1: Bell controlled externally
BYTE    readerAddr;            // 0-31, valid reader address in binary, (32 d =>Undefined/New
                               // reader, so that it would not respond to any commands initially)
BYTE    commType;              // Communication channel type
                               // 0: None (Stand Alone), 1: RS232, 2: RS485 d, 3: Ethernet
BYTE    baudRate;              // RS232/RS485 baud rate select
                               // 0: 4.8k, 1: 9.6k d, 2: 19.2.k, 3: 28.8k, 4: 38.4k, 5: 57.6k bps
BYTE    ipAddr[4];             // IP address for Ethernet network: 4 binary bytes,
                               // example, (0.0.0.0) d
BYTE    subnetMask[4];         // Subnet mask for Ethernet network: 4 binary bytes,
                               // example, (255.255.255.0) d
uint16  facilityCode;          // 0 .. 255 d.. 65535, Facility code
BYTE    output_mode;           // 0: Disabled, 1 d: To Wiegand (data stream) device, 2-255:TBD

// Next are format numbs for card IO; 0 means suppress;
// negative values have special meanings; real card formats have
// numbers in range 1-32767.  If more than one input format,
// all must be Wiegand, or all Magstripe, or all Barcode.
int16   card_synth_fmt;        // 1 d, From pad, SC, &C.
int16   card_out_fmt;          // -1 d = Pass thru.  Otherwise, synth.
int16   card_in_fmts[5];       // Up to 5 formats.  75 d, 117 d, 122 d, 256 d, 1 d

// Global options that can apply to most Wiegand formats:
BYTE    IDoverflo;             // 0 d  = Suppress output.
                               // 1    = Substitute all 1-bits
                               // 2    = Subsitute zero
BYTE    IDunknown;             // 0 d = Suppress output
                               // 1    = Substitute with ID_unk value
                               // 2    = Increment by ID_unk value
                               // 3    = Toggle all parity bits
BYTE    bioreject;             // 0 d = Suppress output
                               // 1    = Substitute with bio_fail value
                               // 2    = Increment by bio_fail value
                               // 3    = Toggle all parity bits
BYTE    duress_act;            // 0    = Suppress output
                               // 1    = Substitute with sig_duress value
                               // 2 d = Increment by sig_duress value
                               // 3    = Toggle all parity bits
int16   ID_unk;                // -32767 .. 0 d .. 32767, ID_unk value for case 1 or 2 above
int16   bio_fail;              // -32767 .. 0 d.. 32767, bio_fail value for case 1 or 2 above
int16   sig_duress;            // -32767 .. 0 d.. 32767, sig_duress value for case 1 or 2 above
BYTE    gatewayAddr[4];        // Gateway address for Ethernet network: 4 binary bytes,
                               // example, (0.0.0.0) d
uint16  site_ID;               // Site ID for keypad output
uint16  company;               // Company code for keypad output
uint32  expiry;                // Expiry for keypad output
uint16  issue_code;            // Issue code for keypad output
BYTE    reserved[116];         // Reserved for FKT Setup expansion
```

} **SETUP_DATA**;

Note: Empty characters in the string field should be cleared. For example, if the service_password string is "4", then service_password[0] = 0x34, and service_password[1] .. service_password[10] should contain 0. This applies to password and ready strings.

GetSetup (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x37*** GetSetup command
Total Data length	1*		

The FKT responds by sending back its SETUP_DATA information to the Host as follows:

GetSetup (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x37*** GetSetup command
Setup Data	256 bytes of **SETUP_DATA**	1*	See **SETUP_DATA** structure definition
Total Data length	257*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters. Initially, the FKT will support sending the entire SETUP_DATA structure to the Host.

Set Setup (0x38)

Store a new/modified reader Setup in the FKT.

SetSetup (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x38*** SetSetup command
Setup Data	256 bytes of **SETUP_DATA**	1	See **SETUP_DATA** structure definition
Total Data length	257*		

SetSetup (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x38*** SetSetup command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters. Initially, the FKT will support setting the entire SETUP_DATA structure from the Host.

Note if any network parameters, such as the reader address, comm. type, baud rate, IP address and subnet mask are changed in the Setup data, the reader should be reset.

Idle (0x39)

Put the FKT in *Idle Mode*. When the FKT is in Idle Mode it does not respond to any keyboard inputs. This command is used prior to loading or clearing the database so that the database is not modified by two different sources at once (a network Host source and a local user). If the FKT is busy or is already idled for another Host source at the time of receipt of the Idle command, the Host will receive a NAK response. Otherwise, the FKT will lock out local keypad access for the duration specified (or upon receipt of the Resume command), before resuming local access. If the timeout duration is 0, the autoResume timeout value in the reader SETUP_DATA is used as the default.

Idle (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x39*** Idle command
Idle Timeout	2*	1*	2-bytes (uint16) Idle period in seconds.
Total Data length	3*		

If the FKT grants Idle mode (FKT_LOCKOUT_FOR_HOST bit in SysStat0 byte will be set), the FKT responds with the format of the Data field in the response packet as follows:

Idle (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x39*** Idle command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the FKT is currently busy, it returns a NAK accompanied by a reason (See Appendix A for details).

Resume (0x3A)

Request FKT to exit Idle mode and re-enable local keypad and display access.

Resume (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3A*** Resume command
Total Data length	1*		

The FKT exits Idle mode and resumes its local operation if the Resume command is from the same Host source as that of the previously-received Idle command. The FKT_LOCKOUT_FOR_HOST bit in SysStat0 byte will be reset, and the FKT responds with the format of the Data field in the response packet as follows:

Resume (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3A*** Resume command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

If the Host source of the Resume command is not the same as that of the previously-received Idle command, the FKT will send a NAK response with FKTIsBusy as the reason.

Get Reader Info (0x3B)

Return the FKT ReaderInfo information. The FKT ReaderInfo Structure is padded to a 256-bytes data structure and is defined as follows with default values marked by [d]:

```
typedef struct
{
    uint16      readerInfoLen           // Length of actual bytes used in Reader Info structure
    BYTE        model_index;            // 0 [d] = FingerKey-B
                                        // 1   = FingerKey-A
                                        // 2   = FingerKey-P
                                        // 3-255: TBD
    BYTE        memoryConfig;           // 0 [d] = 512K Flash, 512K SRAM, 1-255: TBD
    BYTE        promDate[21];           // FKT Null-terminated version string.
                                        // Max 20-chars "MM.mmx date" string, ex. "10.01a 9/26/03"
    BYTE        modelName[21];          // Marketing 20-chars + Null terminator string name
                                        // "RSI FingerKey"*
    uint32      serialNumber;           // 4-bytes Binary
    uint16      userCapacity;           // Maximum number of users, 50 [d]
    unit16      usersEnrolled;          // Number of users currently enrolled, 0 [d]
    BYTE        sensorType;             // Index indicating type of fingerprint sensor used in the FKT
                                        // 0 [d]: Type1, 1-255: TBD
    BYTE        maxTemplateCount;       // Maximum number of templates per user record, 2 [d]
    uint16      templateSize;           // Size of each template, 800 [d] bytes
    BYTE        sensorMemoryConfig;     // Index indicating flash memory size of fingerprint sensor
                                        // 0 [d]: 4 Megabytes, 1-255: TBD
    BYTE        sensorFirmwareVersion[21];      // Fingerprint sensor firmware version string (MM.mmmm)
    BYTE        MACAddress[6];          // MAC binary Address of Ethernet board
    BYTE        boardRevision;          // identifier of board hardware revision, 1 [d]
    uint16      UserDatabaseVersion;    // Current database version in FKT
    BYTE        ethernetEnabled;        // Ethernet option is purchased and thus enabled for use.
                                        // 0 [d] = Not purchased/disabled.  1 = Purchased/Enabled.
```

```
    BYTE        BootLoaderVersion[21];  // FKT Boot Loader version string
    BYTE        reserved[145];          // Reserved for FKT reader info expansion
} FKT_READER_INFO;
```

GetReaderInfo (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3B*** GetReaderInfo command
Total Data length	1*		

The FKT returns its ReaderInfo information with the format of the Data field in the response packet as follows:

GetReaderInfo (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3B***GetReaderInfo command
FKT_Reader_Info Data	256 bytes of **FKT_READER_INFO**	1*	See **FKT_READER_INFO** data structure definition
Total Data length	257*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Get Time (0x3C)

Get the current time registered in the FKT.

GetTime (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3C*** GetTime command
Total Data length	1*		

The FKT returns the time information with the format of the Data field in the response packet as follows:

GetTime (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3C*** GetTime command
Seconds	1*	1*	Seconds (**0** – **59**)*
Minutes	1*	2*	Minutes (**0** – **59**)*
Hours	1*	3*	Hours (**0** – **23**)*
Date	1*	4*	Date (**0** – **31**)*
Month	1*	5*	Month (**1** – **12**)*
Year	1*	6*	Year (**2000** base, **0** – **99**)*
Total Data Length	7*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Set Time (0x3D)

Set a new/modified current time in the FKT.

SetTime (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3D*** SetTime command
Seconds	1*	1*	Seconds (**0** – **59**)*
Minutes	1*	2*	Minutes (**0** – **59**)*
Hours	1*	3*	Hours (**0** – **23**)*
Date	1*	4*	Date (**0** – **31**)*
Month	1*	5*	Month (**1** – **12**)*
Year	1*	6*	Year (**2000** base, **0** – **99**)*
Total Data Length	7*		

If the command is carried successfully, the FKT responds with the format of the Data field in the response packet as follows:

SetTime (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x3D*** SetTime command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the SetTime command cannot be carried out, for example one of the time/date parameters is out of range, the FKT issues a NAK response accompanied by a reason (See Appendix A for details).

Note: The real time clock feature is not available in the FingerKey-B product. Therefore, the time and date will NOT be displayed on the LCD, but the firmware internally still supports these time commands.

Remote Enroll (0x3E)

Start the remote enroll process. Not implement in *HandNet-Lite*, phase 1.

Remote Verify (0x3F)

Start the remote verify process. Not implement in *HandNet-Lite*, phase 1.

Beeper control (0x40)

Turn the beeper On for a duration, Off, or sound the beeper n times. When the beeper is turned On, each beep lasts for the duration (index) specified (duration_index x 50 mSec), and is turned Off for the same duration before the next beep (if there are more than 1 beeps).

> **Note**: The priority of the beeper control follows the mode set by the "flags" field in the **SETUP_DATA** structure.

BeeperControl (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x40*** BeeperControl command
Duration (index) of each beep	1*	1*	(**1-7**)* **50 mSec/unit**. **0*** = Off
Number of beeps	1*	2*	(**1-15**)* beeps. **0*** = Off
Total Data Length	3*		

If the command is carried out successfully, the FKT responds with the format of the Data field in the response packet as follows:

BeeperControl (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x40*** BeeperControl command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the BeeperControl command cannot be carried out, for example, an invalid beeper duration index or invalid number of beeps is requested, the FKT issues a NAK accompanied by a reason (See Appendix A for details).

LED control (0x41)

Turn the selected LED On for a duration, Off, or Flash for a duration.

> **Note**: The priority of the LED control follows the mode set by the "flags" field in the **SETUP_DATA** structure.

LEDControl (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x41*** LEDControl command

LED Operation Type	1*	1*	**0*** : Red and Green LEDs Off **1*** : Red LED On **2*** : Green LED On **3*** : Flash Red LED (**4-255**)*: Undefined
Flash On/Off Cycle Time	1*	2*	(**0**..**255**)*. **50 mSec**/**unit**. Each LED On and Off period is = (n_units x 50) mSec
LED Operation Duration	1*	3*	(**0**..**255**)*. Duration in seconds to perform the specified LED operation before automatically turned off by the FKT (in case the Host forgets to terminate the LED operation).
Total Data Length	4*		

If the command is carried out successfully, the FKT responds with the format of the Data field in the response packet as follows:

LEDControl (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x41*** LEDControl command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the LEDControl command cannot be carried out, for example an undefined LED operation type is requested, the FKT issues a NAK response accompanied by a reason (See Appendix A for details).

Put Card Format (0x42)

Put Card Format command stores an existing or new card format into the FKT card format database. The FKT card format data structures are as follows:

```
//  Format-number Conventions--
//  0:              Not a format; marks end of list of formats
//  negative:       Not a format; signifies special usage
//  0-9999:         Intrinsic format, available when code installed
//  10000-19999: Downloaded formats
//  20000-29999  Special cases requiring special engineering
//  30000-32767: Temporary, throw-away, not intended to ship
//  30030:          (One such that produces a constant result)
//  It is further recommended, but NOT required, to use formats
//  having the low order 4 digits in range 1-999 for W, 1001-1999
//  for M, and 2001-2999 for B.  The code shall not utilize this
//  distinction for distinguishing amongst B, M, and W.

//  Format descriptor encompassing Barcode, Magstripe, and Wiegand

#define WGDGRID 84                          //84 bits is big enough for 25-dig ID; of
```

```
                                                    //course, may still not be big enough
#define RSI_MAX_WIEGAND_FORMAT_LEN 84

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef BYTE RSI_FK_WIEGAND_FLAGS;
typedef BYTE RSI_FK_WIEGAND_BIT_COUNT;

typedef struct
{
    BYTE  formatType;                       // Format type, 'W' for Wiegand
    WORD formatNum;                         // Format Number
    unsigned char formatName[21];           // Null-terminated card format name string
    RSI_FK_WIEGAND_FLAGS flags;             // Bit 0: Backwards (retro)
                                            // Bit 2: BCD
                                            // Bit 3: Enforce exact number of bits
                                            // Bit 4: Permutation exists, i.e. use permutation table
    RSI_FK_WIEGAND_BIT_COUNT bitCount;   // Wiegand Format Length in Bits
    BYTE grid[WGDGRID];                     // Bit-by-bit description of Wiegand bit stream
    BYTE mapping[WGDGRID];                  // Permutation table to map Wiegand bit stream
} RSI_FK_WIEGAND_FORMAT;
```

grid[ ] Byte Explanation:

For each byte in grid[], the low nibble values are defined as follows:

0 Constant 0 bit
1 Constant 1 bit
2 ID bit
3 Site code bit
4 Don't care bit (ignored on input, 0 if synthesized)
5 Don't care bit (ignored on input, 1 if synthesized)
6 First-pass parity bit
7 Second-pass parity bit
8 Third-pass parity bit
9 Fourth-pass parity bit
10 Site_ID bit
11 Company bit
12 Expiry bit
13 Issue_Code bit
14-15 Reserved for future enhancements

For the bytes in the **grid**[] that are designated as pass parity bit (i.e. lower nibble value = **6**, **7**, **8**, **9**), bits **4**, **5**, **6** and **7** of the high order nibble indicate the first, second, third, and fourth pass parity bit respectively, where **0** will indicate even parity, and **1** will indicate odd parity. And for other bytes in the **grid**[] that are not designated as a pass parity bit (i.e. lower nibble value ≠ **6**, **7**, **8**, **9**), bits **4**, **5**, **6** and 7 of the high order nibble indicate whether or not this Wiegand bit will be included in the first, second, third, and fourth pass parity calculation respectively (**0** = excluded in parity calculation for that pass, **1** = included in parity calculation for that pass). Having more than one grid byte indicated for a given parity bit pass is improper and will not be allowed. If there are fewer than four parity bits, the high-nibble bit corresponding to an omitted pass in the grid byte (that is not designated as pass parity bit) should be set to zero.

Mapping[ ] Byte Explanation:

The mapping[] array is used only if bit 4 of flags (RSI_FK_WIEGAND_FLAGS) is set, in which case it represents a permutation applied on input BEFORE the material in grid[] is utilized, and on output, its inverse is applied AFTER grid[] is utilized, and before the bits are transmitted. Only the first BK elements of the mapping[] array are used, each of these BK values must be distinct, and each value must be LT BK. The value in a particular position of mapping[] array represents the 0-based position of the bit which is to wind up at that position after completion of the permutation (this on input). Note that to formulate the correct entries for mapping[], you may first number the elements 0, 1, ..., BK-1 and then perform upon the first BK elements the exact same permutation as is to be performed upon the original (i.e., input) objects (i.e., bits). If bit 0 of flags is set, the permutation implied by it will be performed before that of mapping[] on input, and after that of mapping[] on output.

The format of the Data field in the PutCardFormat command packet (from Host) is as follows:

PutCardFormat command Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x42*** PutCardFormat command
Card Format Data	194*	1*	Content of **RSI_FK_WIEGAND_FORMAT**
Total Data length	195*		

If the card format to be stored exists in the FKT, it will be overridden; if it does not exist and there is room in the FKT card format database, it will be stored as new. The FKT will respond with the format of the Data field in the response packet as follows:

PutCardFormat (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x42*** PutCardFormat command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

If the PutCardFormat command cannot be carried out at all, for example if the card format database is FULL or the card format descriptor content is not proper, the FKT responds with a NAK accompanied by a reason (See Appendix A for details).

Delete Card Format (0x43)

This command deletes the card format specified by the card format number.

DeleteCardFormat (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x42*** DeleteCardFormat command

Card Format Number	2*	1*	Card Format (number) to be deleted
Total Data length	3*		

DeleteCardFormat (Response) Data format: If the specified card format exists, the command will be carried out, and the FKT responds with the format of the Data field in the response packet as follows:

DeleteCardFormat (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x43*** DeleteCardFormat command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the card format does NOT exist, or for other reasons that it cannot be deleted, the FKT will return a NAK accompanied by a reason (See Appendix A for details).

Clear Card Format Database (0x44)

This command clears all (user-defined) card formats in the FKT. Built-in card formats are NOT cleared.

ClearCardFormatDataBase (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x44*** ClearCardFormatDataBase command
Total Data length	1*		

ClearCardFormatDataBase (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x44*** ClearCardFormatDataBase command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

- The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Get Next Card Format (0x45)

This command is used to get the next card format (both built-in and user-defined) in the FKT card format database. Its operation is very similar to that of the GetNextUserRecord command. The Host would normally initiate this command with an initial card format number = 0xFFFF to instruct the FKT to start retrieving card

format from the beginning of the card format database. If a card format exists in the database, the content of the card format descriptor will be returned to the Host. The Host would follow by subsequent GetNextCardFormat command(s) with the previously-received card format number from the FKT as an indication of it receiving the last card format successfully, and as a request for the next card format. If the end of the card format database is reached, the FKT will return a NAK response with the EndOfCardFormat as the reason.

GetNextCardFormat (command packet from Host) Data field format is as follows:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x45*** GetNextCardFormat command
Card Format Number	2*	1*	The card format number of the last successfully-received card format, or restart from the beginning of the card format database. **0xFFFF***: Request FKT to start retrieving card format from the beginning of the card format database. **Other card format value***: The last successfully-received card format, and the FKT is to return the next available card format.
Total Data length	3*		

If the card format database is not empty, the FKT returns the content of the (next) card format descriptor. The format of the Data field in the GetNextCardFormat response packet (from FKT) is as follows:

GetNextCardFormat (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x45*** GetNextCardFormat command
Card Format Data	194*	1*	Content of **RSI_FK_WIEGAND_FORMAT**
Total Data length	195*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Otherwise, if the card format database is empty or it is end of card format retrieval, or the previously sent/received card format number is mismatched, the FKT issues a NAK response accompanied by a reason (See Appendix A for details).

FKT Soft Reset (0x46)

The FKT Soft Reset command causes the FKT to perform a soft reset of the system. The Host normally sends this command to cause one or more system setup parameters to take effect, or to re-synchronize with the FKT. The Host should also send the Idle command (and receives a successful Idle response) before sending the FKTSoftReset command to ensure that it is okay to reset the FKT. The FKT delays about 100 msec before resetting the system to allow for the transmission of the FKTSoftReset response to the Host.

FKTSoftReset (command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x46*** FKTSoftReset command
Total Data length	1*		

The FKT responds to the FKTSoftReset command as follows:

FKTSoftReset (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x46*** FKTSoftReset command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

***** The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

FKT Diagnostic Command (0x7E)

The FKT Diagnostic Command is a special command used strictly for internal control.

FKTDiagCmd (first command packet from Host) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x7E*** FKTDiagCmd command
FKTDiagCmd data bytes	n*	1*	FKTDiagCmd data bytes
Total Data length	(1 + n)*		**Max-Data-field-payload** (**632***) bytes or less

If the FKTDiagCmd data bytes spans multiple packet fragments, after the sending the first packet fragment, the Host should wait for a zero-length response (see Appendix B) from the FKT to signal for the next packet fragment, the format of the Data field in the subsequent and the last command packet fragments is as follows:

FKTDiagCmd (subsequent and last command packet from Host) Data field format:

Field	Bytes	Offset	Description
FKTDiagCmd data bytes	n*	0*	Remaining FKTDiagCmd data bytes
Total Data length	n*		**Max-Data-field-payload** (**632***) bytes or less

After receiving the last packet fragment, the FKT responds, if the command is performed successfully and NO other data is to be returned, with the format of the Data field as follows:

FKTDiagCmd (response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x7E*** FKTDiagCmd command
SysStat0	1*	1*	See Table 2
SysStat1	1*	2*	Reserved = 0*
SysStat2	1*	3*	Reserved = 0*
Total Data length	4*		

If the FKTDiagCmd command is performed successfully and there is data to be returned, the FKT responds with the format of the Data field as follows:

FKTDiagCmd (first response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	**0x7E*** FKTDiagCmd command
Diagnostic data	n*	1*	The length and type of diagnostic data returned varies.
Total Data length	(1 + n)*		**Max-Data-field-payload** (**632***) bytes or less

If the diagnostic data to be returned spans multiple packet fragments, the Host should issue a zero-length command (see Appendix B) after receiving the first packet fragment to signal the FKT for the next packet fragment, the format of the Data field for the subsequent and the last FKTDiagCmd response packet fragment is as follows:

FKTDiagCmd (subsequent and last response packet from FKT) Data field format:

Field	Bytes	Offset	Description
Diagnostic data	n*	0*	Remaining diagnostic data. The length and type of diagnostic data returned varies.
Total Data length	n*		**Max-Data-field-payload** (**632***) bytes or less

If the FKTDiagCmd command fails, the FKT returns a NAK response with the generic FKTDiagCmdFailed reason in the Data field as follows:

Field	Bytes	Offset	Description
NAK	1*	0*	= **0x15*** NAK response.
Failed Cmd_Type	1*	1*	**0x7E*** FKTDiagCmd command
Reason	1*	2*	Generic **FKTDiagCmdFailed** error code
Total Data Length	3*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

Appendix A

NAK response

The FKT may issue a NAK response to a command instead of a normal response, as an indication of an error or a lack of data to be returned. The NAK response has the following form:

<**Fluff**><**Response Sync**><**Address**><**TotalPackets**><**PacketNum**><**Length**><**Data**><**CRC**><**Fluff**>

where:
<**Fluff**>, <**Response Sync**>, <**Address**> and <**CRC**> fields as described in Appendix B.
<**TotalPackets**> = **0x31** (**1** packet fragment)
<**PacketNum**> = **0x31** (packet fragment # **1**)
<**Length**> = **0x30**, **0x30**, **0x30**, **0x33** (3 encoded as Ascii-Hex chars)
<**Data**> field format consists of:

Field	Bytes	Offset	Description
NAK	1*	0*	= **0x15***. Provides a way for the Host/Master to check quickly if the command has been performed. If a command is not performed, the failed command type and reason (See Table 6) are also returned in the NAK response.
Failed Cmd_Type	1*	1*	(**0x30 – 0x7E**)*
Reason	1*	2*	See Table 6 and internal control document
Total Data Length	3*		

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt. There are *length* bytes but *2*length* ASCII-HEX characters

Table 6 "**Reason**" **definitions in a NAK response**

This is a preliminary list of reasons that the FKT returns in the NAK response for a given Host command.

enum **NAK_REASON**
{
NoNak = 0,
StatusClearInvalid ,
WhatFieldInvalid,
UserRecordBytesInvalid,
EndOfUserRecord,
SeqNumMismatched,
UserIDInvalid,
UserIDNotFound,
SensorReadTemplateErr,
SensorWriteTemplateErr,
DBaseIsFull,
SetupBytesInvalid,
FKTIsBusy,
NoIdleRecvPreviously,
TimeDateParamInvalid,
BeeperParamInvalid,
LedParamInvalid,
FKTDiagCmdFailed,

UnexpectedResponseCode,
EthernetNotEnabled,
CardFormatInvalid,
CardFormatDBaseFull,
CardFormatDeleteFailed,
CardFormatNumMismatched,
EndOfCardFormat
};

Reason	Associated Command Type (s)	Description
NoNak	None	When the command is carried out successfully, the FKT normally responds with the current system status bytes. So, NoNak is never returned when the Host command is successfully performed.
StatusClearInvalid	StatusClear	Status requested to be cleared is not defined
WhatFieldInvalid	GetUserRecordByID GetNextUserRecord PutUserRecord	Undefined "what" field requested
UserRecordBytesInvalid	PutUserRecord	Number of user record data bytes received does not match the size of the data structure specified by the "what" field.
EndOfUserRecord	GetNextUserRecord	End of (modified/any) user record retrieval in user database
SeqNumMismatched	GetNextUserRecord	Confirmation sequence number not matched
UserIDInvalid	GetUserRecordByID PutUserRecord DeleteUserRecordByID	Bad or invalid User ID string
UserIDNotFound	GetUserRecordByID PutUserRecord DeleteUserRecordByID	User ID does not exist in database
SensorReadTemplateErr	GetUserRecordByID GetNextUserRecord	Error occurred while retrieving template(s) from sensor
SensorWriteTemplateErr	PutUserRecord	Error occurred while writing template(s) to sensor
DBaseIsFull	PutUserRecord	Database has reached the maximum FKT user capacity. Cannot add more user record(s)
SetupBytesInvalid	SetSetup	Number of setup data bytes received does not match size of setup_data structure
FKTIsBusy	Idle Resume GetUserRecordByID DeleteUserRecordByID GetNextUserRecord ClearUserDatabase	FKT is currently busy and cannot carry the requested command.

NoIdleRecvPreviously	GetUserRecordByID GetNextUserRecord PutUserRecord DeleteUserRecordByID	Idle was not received before receipt of one of these commands.
TimeDateParamInvalid	SetTime	One or more time/date parameters is out of range
BeeperParamInvalid	BeeperControl	Beep-duration/num beeps are invalid
NbeepsInvalid	BeeperControl	Number-of-beeps is out of range
LedParamInvalid	LEDControl	Undefined LED operation requested
UnexpectedResponseCode	Any Host Command/Operation	An unexpected error/code occurred when a Host command/operation is carried out by a subsystem module.
EthernetNotEnabled	SetSetup	This error code is returned when the commType in the setup data structure is set to Ethernet, and the Ethernet option has not been purchased yet.
FKTDiagCmdFailed	FKTDiagCmd	Generic failure code for the FKT diagnostic command
CardFormatInvalid	PutCardFormat	Card format descriptor content is invalid.
CardFormatDBaseFull	PutCardFormat	FKT card format database is full, thus cannot add (new) card format.
CardFormatDeleteFailed	DeleteCardFormat	FKT cannot delete specified card format.
CardFormatNumMismatched	GetNextCardFormat	Mismatch of previously sent/received card format number.
EndOfCardFormat	GetNextCardFormat	End of card format retrieval in the FKT card format database.
nn-255	Reserved	TBD

Appendix B

This appendix describes the structure of the commands and responses to and from the FKT.

Command/Response Protocol Structure

The commands to the FKT and the responses from the FKT have similar formats. The main difference between the two data streams is the value in the sync field.

Commands from the Host to the FKT take the following form:

<**Fluff**><**Command Sync**><**Address**><**TotalPackets**><**PacketNum**><**Length**><**Data**><**CRC**><**Fluff**>

Responses from the FKT to the Host take the following form:

<**Fluff**><**Response Sync**><**Address**><**TotalPackets**><**PacketNum**><**Length**><**Data**><**CRC**><**Fluff**>

Regardless of the network media, RS485/RS232 or Ethernet (and IrDA), these same command and response packets are intended to work transparently. Currently, the Ethernet protocol stack shipped with the FKT does not support fragmentation and reassembly of large application packets. Although the embedded stack supports an Ethernet Maximum Segment Size (**MSS**) of **1460** bytes (1500 bytes – 40 bytes TCP/IP header), some Hosts may use a MSS of 1300+ bytes. For now, we will choose to use a TCP/IP data payload of **1280** bytes to transport the RSI FKT Host commands. The actual maximum number of payload bytes in the Data field of the RSI FKT Host Command protocol is then as follows:

Command/Response Packet pre-header bytes:
Fluff(1)+**Cmd/Resp Sync**(3)+**Address**(1)+**TotalPackets**(1)+**PacketNum**(1)+**Length**(4 Ascii-Hex chars) = 11 bytes

Command/Response Packet post-header bytes:
CRC(4 Ascii-Hex chars)+**Fluff**(1) = 5 bytes

Overhead = **16** bytes

Max-Data-field-payload = 1280 – 16 (overhead) = **1264** (Ascii-Hex) chars (or **632** binary bytes)

Since the amount of data transfer in fingerprint template management can be more than 1280 bytes, these original Data blocks can span multiple command/response packet fragments. Thus, the **TotalPackets** and **PacketNum** fields are introduced to take care of breaking down these large Data blocks into packet fragments and reassembling them for the upper applications. The following paragraphs describe each field of the command and response packets for Data blocks that can be transferred in a single packet or must be transferred in multiple packet fragments.

<**Fluff**> bytes are 0xFF. These bytes are not to be trusted. They may or may not be present. They may or may not be 0xFF upon receipt. Their sole purpose is to put plenty of "stop" bits on the line in the case of an RS485 link when the line is "turned around". These bytes are not needed by the receiver state machine in the FKT but **must** be generated by the Host transmitter.

<**Command Sync**> are the three bytes <**0**x**0C 0**x**0A 0**x**3E**>. These bytes are chosen so that devices running the old protocol will not be triggered into action from a new protocol command and the FKT's receiver state machine will not be triggered from a command nor a response on the old protocol. Even though 0x3E is a valid Hand Reader address, it is an *unlikely* address. Most Hand Readers are addressed in the range 0x00 to 0x1F.

<**Response Sync**> are the three bytes <**0**x**0D 0**x**0A 0**x**3E**>. These bytes are chosen so that devices running the old protocol will not be triggered into action from a new protocol command and the FKT's receiver state

machine will not be triggered from a command nor a response on the old protocol. Even though 0x3E is a valid Hand Reader address, it is an *unlikely* address. Most Hand Readers are addressed in the range 0x00 to 0x1F.

<**Address**> is the value **0**x**30** to **0**x**4F** – exactly 32 possible values.

<**TotalPackets**> has the value **0**x**30** to **0**x**33** for **0** to **3** packet fragments. These values are derived depending on the size of the original Data block being transferred. If the original Data block (binary) bytes size is less than/equal to the Max-Data-field-payload, then the value of **TotalPackets** is just **0x31** for **1** packet (fragment); otherwise it can be calculated as follows:

Example:
Currently, the largest message is PutUserRecord, its original Data block (binary) byte size is:
PutUserRecord Cmd(1)+UserID(26)+what(1)+FpUserRecord(1646) = 1674 (binary) bytes
TotalPackets = (1674 +(632-1))/632 = **3** (or **0x33**) packet fragments.

Note: When a command/response packet having **TotalPackets** = **0x30** (**0** packet fragment), **PacketNum** = **0x31**..**0x33** (packet fragment # **1** to **3**) and **Length** = **0x30**, **0x30**, **0x30**, **0x30** (**0** length Data field bytes), the packet is interpreted as the signal from the Host/FKT ready for the next packet fragment (indicated by **PacketNum**).

<**PacketNum**> has the value **0**x**31** to **0**x**33** for packet # **1** to **3**. Normally, the value of PacketNum is from **0x31** to **0x33** for packet (fragment) # **1** to **3**, depending on the size of the original Data block being transferred. If the original Data block (binary) bytes size is less than/equal to the Max-Data-field-payload, the value of **PacketNum** is then just **0x31** for packet fragment # **1**. Otherwise it can be **0x31** to **0x33** for packet fragment # **1** to **3**.

Note: When a command/response packet having **PacketNum** = **0x31**..**0x33** (packet fragment # **1** to **3**) **TotalPackets** = **0x30** (**0** packet fragment) and **Length** = **0x30**, **0x30**, **0x30**, **0x30** (**0** length Data field bytes), the packet is interpreted as the signal from the Host/FKT ready for the next packet fragment (indicated by **PacketNum**).

<**Length**> consists of four ASCII-HEX characters representing a 16-bits binary value (high byte first) of data bytes in the packet (fragment); it can be <**0x30**, **0x30**, **0x30**, **0x30**> to <**0x30**, **0x32**, **0x44**, **0x32**> for **0** to **632** (Max-Data-field-payload) Data field bytes. Note that this Length field is ASCII-HEX encoded, as are all data bytes in the Data field that follows. The Length refers to the bytes of data, not the number of ASCII-HEX (Data field) characters. The number of ASCII-HEX (Data field) characters transmitted is equal to twice the data length because each Data field byte is transmitted as two ASCII-HEX characters, most significant nibble first.

Note: When a command/response packet having **Length** = **0x30**, **0x30**, **0x30**, **0x30** (**0** length Data field bytes), **TotalPackets** = **0x30** (**0** packet fragment) and **PacketNum** = **0x31**..**0x33** (packet fragment # **1** to **3**) the packet is interpreted as the signal from the Host/FKT ready for the next packet fragment (indicated by **PacketNum**).

<**Data**> consists of <Length x 2> ASCII-HEX characters representing <Length> bytes (possibly zero bytes) of binary data. If the original Data block (binary) bytes size is less than/equal to the Max-Data-field-payload, the Data field for packet fragment # **1** (and also the last) will contain the following:

Field	Bytes	Offset	Description
Cmd_Type	1*	0*	(**0**x**30** to **0**x**7E**)* -- exactly 79 possible values. These command type values correspond to the values next to the command description.

data	n*	1*	n data bytes
Total Data Length	(1 + n)*		**Max-Data-field-payload** (**632***) bytes or less

If the original Data block (binary) bytes size is greater than the Max-Data-field-payload, there will be more than 1 packet fragments, and the Data field for the subsequent and the last packet fragments will just contain data bytes as follows:

Field	Bytes	Offset	Description
data	n*	0*	n data bytes
Total Data Length	n*		**Max-Data-field-payload** (**632***) bytes or less

* The values and offsets of the fields described in the Data field format table(s) above are in their un-encoded form for the ease of reading. They will be (Ascii-Hex) encoded for transmission and decoded upon receipt.receipt. There are *length* bytes but *2*length* ASCII-HEX characters.

In particular, if the packet fragment is not the first nor the last packet fragment, its Data field should have Max-Data-field-payload bytes, whereas the Data field in the first and the last packet fragment can have Max-Data-field-payload bytes or less.

<**CRC**> consists of four ASCII-HEX characters representing the 16-bits CRC value (high byte first) calculated over all of the data in the packet starting with <**Sync**> field and ending at the last data byte. The CRC polynomial used is CRC-CCITT. The high byte of the CRC is stored first. The CRC is stored in ASCII-HEX format. The CRC is calculated over the binary data *not* the ASCII-HEX form of the data. The receiver should always convert the ASCII-HEX data to binary before storing in the receive buffer and before adding to the CRC.

Zero-Length Command/Response Packets

As described in the above section, when **TotalPackets** = **0x30**, **PacketNum** = **0x31**..**0x33**, and **Length** = **0x30**, **0x30**, **0x30**, **0x30**, these command/response packets are designated as the signal from the Host/FKT for the next packet fragment (indicated by **PacketNum**).

Signal from the Host to the FKT for next packet fragment:

<Fluff><Command Sync><Address=FKT address><TotalPackets=0x30><PacketNum=0x31..0x33><Length=0x30 0x30 0x30 0x30><CRC><Fluff>

Signal from the FKT to the Host for next packet fragment:

<Fluff><Response Sync><Address=0xFF><TotalPackets=0x30><PacketNum=0x31..0x33><Length=0x30 0x30 0x30 0x30><CRC><Fluff>

ASCII-HEX format

Length, Data, and CRC information is stored in ASCII-HEX format. The length and CRC fields are 16-bit fields whereas the Data field is considered to be an array of 8-bit fields.

Each source byte is converted to two ASCII characters before transmission. Each character represents 4 bits of the value. The first of the two bytes is the high nibble and the second is the low nibble.

Example 1: 0x47 is converted to the two ASCII characters 0x34 0x37 or '4' '7'.

Example 2: 0x1234 is converted to the four ASCII characters 0x31 0x32 0x33 0x34

Example 3: 0xAF is converted to the two ASCII characters 0x41 0x46 or 'A' 'F'.

The intent of this format is two-fold: first we don't want to transmit any characters that could possibly interfere with the old protocol. Avoiding all control (non-printable) characters does this. Second, since we're sending everything in printable ASCII characters we'd like to be able to read and understand the message without having to write a program to interpret the protocol. A dumb terminal program will be sufficient to read transmissions.

Endian-ness

We choose High Byte Endian format because it is convenient for this processor/compiler combination as well as being "right-reading" on a dumb terminal.

Appendix C

The FingerKey (Host) Network Command protocol in its current state has been demonstrated to work well on the RS232/485 media source. However, it is subject to change(s) to have this protocol working on the Ethernet and on the IrDA media sources as future implementation progresses.