Lumidigm vCOM Command Reference

PLT-02301, Rev. A.2
Software Version 6.00
August 2016

hidglobal.com

Copyright

Trademarks

Revision History

Date	Description	Version
08/15/16	Software Version 6.00	A.2
07/24/15	V371 release.	A.1
12/16/14	First release as part of HID Global (1.6)	A.0

Contacts

For additional offices around the world, see **www.hidglobal.com** corporate offices.

North America & Corporate	**Lumidigm**
611 Center Ridge Drive Austin, TX 78753 USA Phone: 866-607-7339 Fax: 949 732 2120	For Lumidigm specific issues: Website: http://www.hidglobal.com/lumidigm-technical-support Email: Lumidigm@hidglobal.com Phone: 505 272 7057
HID Global Customer Support: support.hidglobal.com	

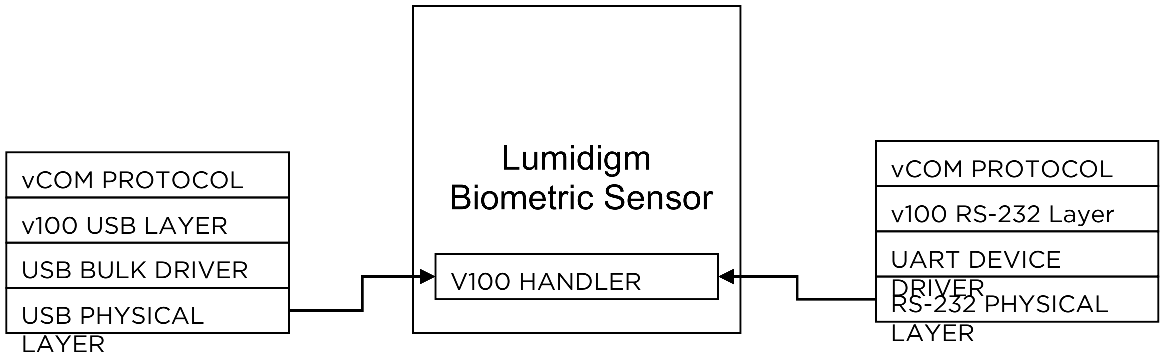
Contents

1 Overview

This document provides message transaction and command set descriptions for communication with the Lumidigm, Inc. Biometric Fingerprint Sensor using the vCOM protocol.

1.1 System Diagram

The vCOM protocol may be used over either a serial communications or USB communications channel. vCOM message transactions from either channel are parsed, validated, implemented, and a reply vCOM message is returned. The vCOM protocol is a layer above the device driver layer and is suitable for porting to additional physical communication channels or devices.



1.1.1 USB Communications

2.0

Bulk Transfer (64k Max Packet)

1.1.2 Additional

U32:	4 byte little Endian
U16:	2 byte little Endian
U8:	1 byte

1.2 vCOM Transaction Protocol Message Packet Description

Each message transaction with the Lumidigm Biometric Fingerprint Sensor consists of a Command Packet from the host followed by a Reply Packet provided by the sensor. All messaging is transaction based, implying all commands will elicit a response from the sensor. Both Command and Reply Packets in a transaction shall be of the same generic format with command-specific information embedded in the "opaque data" field.

Message Format

SOH	CMD	ARG	SIZE	OPAQUE DATA	Reserved

Where:

Message Field Definition

Field	Size in Bytes	Default Value	Description
SOH	2	0x560D	Start of Heading
CMD	4	-	Enumerated V100 Command
ARG	2	-	Command specific argument. Usually defined by an enumerated type, cast into 2 bytes.
SIZE	4	-	Size of OPAQUE DATA, if any
OPAQUE DATA	SIZE	-	Data payload specific to Command
Reserved	2	0x0000	Reserved for future use (*)

The opaque data message is optional and may be zero bytes in length. The minimum size command is therefore 14 bytes consisting of the SOH, CMD, ARG, SIZE, Reserved.

1.2.1 Example

A template capture command is assembled as follows:

(Little Endian LSB/MSB hex)

SOH		CMD				ARG		SIZE				Reserved
0D	56	01	00	00	00	01	00	00	00	00	00	00 00 (*)

The Reply Packet with a dummy 2 byte template for example would be:

SOH		CMD				ARG		SIZE				OPAQUE DATA		Reserved
0D	56	01	00	00	00	01	00	02	00	00	00	01	02	00 00 (*)

(*) The last two reserved bytes are ignored, thus any value is acceptable here.

1.3 v100 USB Layer description

The v100 USB Layer is concerned only with transmitting and receiving opaque data, not interpreting any data it sends or receives. Opaque data packets sent and received that are larger than 64KB are split up into 64KB packets, followed by a packet that is equal in size to the remainder of the Size/64KB (or Original Size/64KB).

At a high level, a client transaction goes as follows:

```
// Write the header, containing the size to send.

usbcb.ulCommand = 0x0;
usbcb.ulCount   = nTxSize;
usbcb.ulData    = 0x0;
```

1.3.1 *** Send header

```
WriteBytes(USBWriteHandle, &usbcb, sizeof(usbcb), MSEC_TIMEOUT))
```

1.3.2 *** Send data

```
WriteBytes(USBWriteHandle, myPacket, usbcb.ulCount, MSEC_TIMEOUT))
```

1.3.3 *** Read response header

```
ReadBytes(USBReadHandle, &usbcb, sizeof(usbcb), MSEC_TIMEOUT))
```

1.3.4 *** Read response data

```
ReadBytes(USBReadHandle, pResponse, usbcb.ulCount, MSEC_TIMEOUT))
```

In the above case, the "WriteBytes" and "ReadBytes" functions are responsible for splitting up large data packets into 64KB blocks.

The recommended timeout (MSEC_TIMEOUT) should be 7000 milliseconds.

See the provided example source code for Windows implementation details.

1.4 USB Bulk Driver Description

See the provided example source code for the Windows example.

1.5 v100 RS-232 Layer Description

The v100 RS-232 layer is concerned only with transmitting and receiving opaque data, not interpreting any data it sends or receives.

The v100 RS-232 layer uses a standard XModem protocol, with 128 byte packets. Pseudocode for the protocol looks something like this:

```
TransmitCommand(uchar* pPacket, uint nPacketSize)
{
// Flush anything in input buffer
flushReceiveInput();
// Send 2 NAK’s (0x15) to alert host we are about to send
SendTwoNAKs();
// Perform standard XModem transmit, 128 byte, 16 bit CRC
XModemTransmit(myPacket,nTxSize);
// Wait for 2 NAKs
WaitForTwoNAKs();
// Go into XModem Receive
XModemReceive(pResponseBuffer,nMaxRxSize);
}
```

Note: See the Example Source code for implementation details.

1.6 UART Device Driver

The V100 UART Device supports speeds from 9600 BAUD to 240.4 kbit/sec.

1.6.1 RS-232 Communication Port Settings

Baud Rate	Selectable [9.6, 19.2, 38.4, 57.6, 115.2, 230.4] kbit/sec
Data Bits	8
Parity	none
Stop Bits	1
Flow Control	None
Endianness	Little

Note: See the sample code for sample Windows implementation.

1.7 Command Reference

The individual commands which may be sent to the biometric reader are described below. Some of these commands may be disabled and will return error codes if they are not supported by the current firmware revision or impractical to implement due to other configuration specific options. Additionally commands may be "Locked" and only "Unlocked" by license keys.

There are three basic categories of commands:

- **Commands**

  Low-level commands which provide greater granularity and flexibility for customizing or redefining device operation.

- **Database Commands**

  Consists of both Macro and Atomic commands for manipulation and management of the on-board template database

- **Diagnostic/Maintenance Commands**

  Commend Set for upgrading firmware, running built-in diagnostics testing and loopback testing

The availability of the various commands depends upon the capability supported in the firmware revision as well as the capability locked/unlocked by the license key. Many commands are designated as "Open" and are always available, while other commands are designated as "Lock" indicating either the firmware does not support the feature or the license key restricts access. The "CMD_GET_CONFIG" may be issued for a global capability response.

1.8 Error Handling

Each Command Packet will elicit a Reply Packet. A successful transaction will elicit a Reply Packet of the same type *CMD,* where an unsuccessful transaction will yield a reply of type CMD_ERROR. Checking the *CMD* field first of the response packet ensures that you are interpreting the right type of Reply Packet.

1.9 Data Types

The following data types may be exchanged with the reader (See Appendix A for Definitions):

1.9.1.1 _V100_INTERFACE_CONFIGURATION_TYPE

This is a READ only structure stored in FLASH which provides all necessary device configuration and capability information

1.9.1.2 _V100_INTERFACE_COMMAND_TYPE

This structure contains all USER configurable parameters. It is volatile in nature and may be READ or WRITTEN as required.

1.9.1.3 _V100_INTERFACE_STATUS_TYPE

This READ only structure provides global error report for the device.

1.10 Templates

The user can set the template mode using the CMD_SET_OPTION call with OPTION_SET_TEMPLATE_MODE. The template mode describes the input/output template format for the commands. The template formats currently supported are as follows:

_V100_TEMPLATE_MODE	**Template Type**
TEMPLATE_ANSI_378	ANSI/INCITS 378-2004(ANSI378) ANSI 378+ (for sensors that have the Minex III certified extractor, see note below)
TEMPLATE_ISO_NORMAL	ISO/IEC 19794-2:2005(ISO 19794-2) ISO 19794:2011 (for sensors that have the Minex III certified extractor, see note below)

The default template mode will be TEMPLATE_ANSI_378. The format of the input/output template for the following commands corresponds to the template mode set during CMD_SET_OPTION call.

Note: ANSI 378+ and ISO 19794:2011 templates are returned by the V30x-40-S sensor, the V31x sensor using the Lumidigm Device Service 6.00, and V30x sensors upgraded to FW 29428 or higher. These templates will be somewhat larger, but still well below the max template size of 2048 bytes as defined by the VCOM API. These templates can be used to improve biometric performance of the Minex III certified matcher when CMD_MATCH, CMD_ID_VERIFY, and when using the new proprietary 1:N database engine with the V31x sensor.

Command
CMD_GET_TEMPLATE
CMD_MATCH
CMD_MATCH_EX
CMD_SET_TEMPLATE
CMD_TRUNCATE_378
CMD_ID_GET_USER_RECORD
CMD_ID_IDENTIFY_378
CMD_ID_SET_USER_RECORD
CMD_ID_VERIFY_378
CMD_ID_VERIFY_MANY

1.11 Images

The image formats available are:

1. Monochrome 8-BPP composite image using CMD_GET_COMPOSITE_IMAGE or CMD_GET_IMAGE
2. Composite image in Finger Image Record (FIR) format using CMD_GET_FIR_IMAGE.

2 Base Command Set Summary

The command sets can be summarized in four groups: general commands, diagnostic commands, 1:1 commands, and 1:N commands.

2.1 General Commands

General Commands	Code	Description
CMD_ARM_TRIGGER	0x47	Starts presence detection and image stack acquisition thread
CMD_CANCEL_OPERATION	0x64	Cancels capture-related commands
CMD_CONFIG_COMPORT	0x56	Used to change the baud rate, data bits, and flow control settings of the serial communication channel
CMD_ERROR	0xE0	Error packets are only valid as a Reply Packet. They are returned when an error occurs, in response to any Command Packet sent.
CMD_FILE_DELETE	0xB5	Deletes a file
CMD_GET_ACQ_STATUS	0x48	Returns “Busy” status of current acquisition thread
CMD_GET_CMD	0x4C	Returns current settings of USER controllable features (_V100_INTERFACE_COMMAND_TYPE)
CMD_GET_COMPOSITE_IMAGE	0x43	Return composite image
CMD_GET_CONFIG	0x49	Returns device configuration structure which includes definitions of all supported services (_V100_INTERFACE_CONFIGURATION_TYPE)
CMD_GET_FIR_IMAGE	0x65	Returns composite image in FIR format
CMD_GET_GPIO	0x63	Gets GPIO mask
CMD_GET_IMAGE	0x41	Return image
CMD_GET_OP_STATUS	0x8B	Retrieves status of macro operation
CMD_GET_SERIAL	0x55	Get device serial number
CMD_GET_STATUS	0x4A	Returns device status structure which contains all device error codes, conditions, and health monitoring data (_V100_INTERFACE_STATUS_TYPE)
CMD_GET_TAG	0x8C	Gets tag data
CMD_GET_TEMPLATE	0x45	Returns the template in the current Template buffer. This was the last template processed
CMD_MATCH	0x04	Two templates may be provided to the unit and a similarity score is returned
CMD_MATCH_EX	0x60	One or two templates are provided returning a minutia and/or spoof matching score
CMD_PROCESS	0x57	Processes existing data in stack
CMD_SAVE_LAST_CAPTURE	0xA1	Saves last capture
CMD_SET_CMD	0x4D	Sets current setting of USER controllable features (_V100_INTERFACE_COMMAND_TYPE)
CMD_SET_COMPOSITE_IMAGE	0x44	Set Composite Image Buffer
CMD_SET_GPIO	0x62	Sets GPIO mask

General Commands	Code	Description
CMD_SET_IMAGE	0x42	Send Image
CMD_SET_LED (V30x only)	0x54	Permits manual On/Off control of the user feedback LEDs
CMD_SET_OPTION	0x58	Sets system options
CMD_SET_TAG	0x87	Sets a tag
CMD_SET_TEMPLATE	0x46	Downloads template to current template buffer
CMD_TRUNCATE_378	0x59	Truncates 378 templates
CMD_UPDATE_FIRMWARE	0xA2	Updates firmware
CMD_VID_STREAM	0x06	Sets video stream mode

2.2 Diagnostic Commands

Diagnostic Commands	Code	Description
CMD_RESET	0xC2	Issues a system reset (reboot) command to the device

2.3 1:1 Commands

1:1 Commands	Code	Description
CMD_ADD_USER	0x89	Add user record to DB
CMD_DELETE_USER	0x84	Delete a user from DB
CMD_ENROLL_USER	0x82	Enroll a user
CMD_FORMAT_DB	0x86	Format user database
CMD_GET_DB_METRICS	0x85	Get database metrics
CMD_GET_USER	0x88	Get user record
CMD_GET_VERIFICATION_RULES	0x81	Get policy/criteria for enrollment
CMD_SET_VERIFICATION_RULES	0x80	Set policy/criteria for enrollment
CMD_VERIFY_USER	0x83	Verify a user

2.4 1:N Commands

1:N Commands	Code	Description
CMD_ID_CREATE_DB	0x70	Creates new DB
CMD_ID_DELETE_DB	0x78	Deletes a DB
CMD_ID_DELETE_USER_RECORD	0x7C	Deletes a User or User Record from DB
CMD_ID_ENROLL_USER_RECORD	0x76	Enrolls a User Record
CMD_ID_GET_DB_METRICS	0x7A	Gets metrics of working DB
CMD_ID_GET_PARAMETERS	0x7E	Gets ID parameters
CMD_ID_GET_RESULT	0x7B	Retrieves the result of last successful identify executed
CMD_ID_GET_SYSTEM_METRICS	0x7D	Gets information of DB groups on the system
CMD_ID_GET_USER_RECORD	0x74	Retrieves a user record from a DB
CMD_ID_GET_USER_RECORD_HEADER	0xA3	Retrieves a user record header from working DB
CMD_ID_IDENTIFY	0x79	Identifies a User
CMD_ID_IDENTIFY_378	0x77	Identifies a User from a template
CMD_ID_SET_PARAMETERS	0x7F	Sets ID parameters
CMD_ID_SET_USER_RECORD	0x75	Adds existing User Record to DB
CMD_ID_SET_WORING_DB	0x71	Sets working DB
CMD_ID_VERIFY_378	0xA0	Verifies a User or User Record from a template
CMD_ID_VERIFY_MANY	0xA4	Captures a finger print and verifies it against a set of input templates.
CMD_ID_VERIFY_USER_RECORD	0x73	Verifies a User or User Record

3 General Command Descriptions

3.1 CMD_ARM_TRIGGER

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ARM_TRIGGER	A*	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ARM_TRIGGER	None	*Z	-

REMARKS: Sets the Trigger mode. Valid modes are as follows:

`TRIGGER_ON`: Arms the device trigger, and starts the processing chain as described in the `_V100_INTERFACE_COMMAND_TYPE` structure.

`TRIGGER_OFF`: If presence detection is running, this disarms the trigger and returns the device to an idle state. It returns GEN_OK or GEN_ERROR_APP_BUSY if the system is busy. In either case user must poll for completion using CMD_GET_OP_STATUS for macro commands and CMD_GET_ACQ_STATUS for atomic commands. Please see CMD_GET_OP_STATUS/ CMD_GET_ACQ_STATUS for expected values.

`TRIGGER_FINGER_DETECT`: Arms the device trigger and starts finger detection mode. This mode is supported for M30x, M31x and V31x sensors, and V30x sensors with firmware greater than 9538 only. User can poll for status using CMD_GET_ACQ_STATUS for finger presence or not.

`CANCEL_VERIFICATION`: Cancels CMD_ENROLL_USER, CMD_VERIFY_USER if in progress.

Arming the device trigger can be seen as the “execution” step in the workflow of your application. The workflow of the system is described in detail in the `CMD_SET_CMD` description.

Footnote	C/R	Category	Data Type	Size (bytes)	Description
A	C	ARG	_V100_TRIGGER_MODE	sizeof(…)	Trigger State to Set
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.2 CMD_CANCEL_OPERATION

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_CANCEL_OPERATION	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_CANCEL_OPERATION	None	*Z	-

REMARKS:
Cancel capture-related commands. You can cancel CMD_ARM_TRIGGER, CMD_ID_IDENTIFY, CMD_ID_ENROLL_USER_RECORD, CMD_ID_VERIFY_USER_RECORD and CMD_ID_VERIFY_MANY using this command.

It returns GEN_OK or GEN_ERROR_APP_BUSY if the system is busy. In either case user must poll for completion using CMD_GET_OP_STATUS for macro commands and CMD_GET_ACQ_STATUS for atomic commands. Please see CMD_GET_OP_STATUS/ CMD_GET_ACQ_STATUS for expected values.

RETURNS:
See Error code definitions for error codes related to this command.

Footnote	C/R	Category	Data Type	Size bytes)	Description
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.3 CMD_CONFIG_COMPORT

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_CONFIG_COMPORT	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_CONFIG_COMPORT	None	*Z	-

REMARKS:

Configures V100 UART Baud Rate. Baud rate defaults at 9600 baud. Baud rate tested up to 230,400 BPS. Resets to 9600 baud after device reboots. Baud rate change takes effect after response packet is sent.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C	Opaque1	unsigned int	4	size of Opaque2(should be 4)
	C	Opaque2	unsigned int	4	Baud Rate to Set
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.4 CMD_ERROR

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
-	-	-	-	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ERROR	*A	-	-

Error Packets are only valid as a Reply Packet. They are returned when an error condition occurs, in response to any Command Packet sent. Cast the returned ARG field into a `_V100_GENERAL_ERROR_CODES` enumeration, and handle error appropriately.
Current supported error codes are shown in the table in Appendix B.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C/R	ARG	unsigned short	2	_V100_GENERAL_ERROR_CODES

3.5 CMD_FILE_DELETE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_FILE_DELETE	None	*Z	*A,*B,*C
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_FILE_DELETE	None	*Z	-

REMARKS:
Deletes the specified file

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C	Opaque	unsigned int	4	Size of FileName
B	C	Opaque	char*	*A Bytes	FileName
C	C	Opaque	_V100_FILE_ATTR	Sizeof(_V100_FILE_ATTR)	File attribute
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.6 CMD_GET_ACQ_STATUS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_ACQ_STATUS	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_ACQ_STATUS	None	*Z	*A

REMARKS:

Allows the client to poll for acquisition and processing status. Note that during Acquisition and Presence detection, `CMD_GET_ACQ_STATUS` can be counted on to return immediately. During Processing, `CMD_GET_ACQ_STATUS` will return at certain intervals during the processing chain, sometimes up to a second after issuing the command. Make sure to set your transport layer time-outs appropriately.

Valid `_V100_ACQ_STATUS_TYPE` are as follows:

`ACQ_BUSY`: Application is acquiring image data, or is in presence detection mode.
`ACQ_PROCESSING`: Application has acquired image data, and is now processing data.
`ACQ_TIMEOUT`: A timeout has occurred. System is back to an idle state.
`ACQ_DONE`: The acquisition/processing chain has completed.
`ACQ_NO_FINGER_PRESENT`: No Finger present (during Finger Detection)
`ACQ_MOVE_FINGER_UP`: User should move finger in direction indicated.
`ACQ_MOVE_FINGER_DOWN`: User should move finger in direction indicated.
`ACQ_MOVE_FINGER_LEFT`: User should move finger in direction indicated.
`ACQ_MOVE_FINGER_RIGHT`: User should move finger in direction indicated.
`ACQ_FINGER_POSITION_OK`: User should stop moving finger for acquisition.
`ACQ_CANCELLED_BY_USER`: Command Cancelled by user.
ACQ_LATENT_DETECTED: Latent detected.
`ACQ_FINGER_PRESENT`: Finger present (during Finger Detection)
ACQ_NOOP: No Operation occurring.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	R	Opaque1	_V100_ACQ_STATUS_TYPE	sizeof(…)	Processing Status
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.7 CMD_GET_CMD

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_CMD	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_CMD	None	*Z	*A

REMARKS:
Gets the current `_V100_INTERFACE_COMMAND_TYPE` structure from the device. See `CMD_SET_CMD` for information about the `_V100_INTERFACE_COMMAND_TYPE` structure.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	C	Opaque1	unsigned int	4	Size of structure
	C	Opaque2	_V100_INTERFACE_COMMAND_TYPE	sizeof(…)	structure data
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.8 CMD_GET_COMPOSITE_IMAGE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_COMPOSITE_IMAGE	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_COMPOSITE_IMAGE	None	*Z	***A**

REMARKS:
Gets the current composite image, and associated spoof value. The dimensions and image format of the image returned can be found by issuing a call to `CMD_GET_CONFIG`, and apply as follows:

Width: Composite_Image_Size_X

Height: Composite_Image_Size_Y

Format: 8-BPP monochrome.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
B	R	Opaque 1	int	4	Spoof Value
	R	Opaque 2	unsigned int	4	Image Size
	R	Opaque 3	unsigned char	Opaque 2	Image bytes
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.9 CMD_GET_CONFIG

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_CONFIG	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_CONFIG	None	*Z	*B

REMARKS:

Returns the `_V100_INTERFACE_CONFIGURATION_TYPE` structure. This is a read-only structure which describes various metrics of system capabilities, including available extractors, matchers, template types, image formats, image sizes, and so forth. See Appendix A for a complete description of the `_V100_INTERFACE_CONFIGURATION_TYPE` structure.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	C	Opaque1	unsigned int	4	sizeof _V100_INTERFACE_CONFIGURATION_TYPE
B	R	Opaque1	_V100_INTERFACE_CONFIGURATION_TYPE	sizeof	_V100_INTERFACE_CONFIGURATION_TYPE
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.10 CMD_GET_FIR_IMAGE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_FIR_IMAGE	None	*Z	*A, *B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_FIR_IMAGE	None	*Z	*C, *D

REMARKS:

Gets the current composite image in FIR format. The size of the FIR image returned will be as follows:

For LUMI_FIR_ANSI: nFIRImageSize = ANSI_381_HDR_SIZE + composite image size
For LUMI_FIR_ISO: nFIRImageSize = ISO_19794_4_HDR_SIZE + composite image size

The dimensions of the composite image can be found by issuing a call to `CMD_GET_CONFIG`, and apply as follows:

Width: Composite_Image_Size_X
Height: Composite_Image_Size_Y
Format: 8-BPP monochrome.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C	Opaque1	_V100_FIR_RECORD_TYPE	4	FIR type
B	C	Opaque2	_V100_FINGER_PALM_POSITION	4	Finger type
C	R	Opaque 1	Unsigned int	4	Size of *D(Bytes)
D	R	Opaque 2	unsigned char*	*C	FIR Image
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.11 CMD_GET_GPIO

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_GPIO	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_GPIO	None	*Z	*A

REMARKS:

Gets GPIO mask.

RETURNS:

See Error code definitions for error codes related to this command.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	R	Opaque 1	unsigned char	1	GPIO mask returned
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.12 CMD_GET_IMAGE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_IMAGE	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_IMAGE	None	*Z	*B

REMARKS:

Gets current image in `_V100_IMAGE_TYPE` buffer. Valid types of images to get for general use are `IMAGE_COMPOSITE, IMAGE_VID_STREAM and IMAGE_WSQ` (M30x, M31x and V31x sensors, and V30x sensors with firmware > 9538). The dimensions and image format of the image returned can be found by issuing a call to `CMD_GET_CONFIG`, and apply as follows:

IMAGE_COMPOSITE

Width: Composite_Image_Size_X

Height: Composite_Image_Size_Y

Format: 8-BPP monochrome.

IMAGE_VID_STREAM

Width: Native_Image_Size_X

Height: Native_Image_Size_Y

Format: Bayer-pattern BGGR

Recommended for high transport speed use only.

IMAGE_WSQ

The actual size of the WSQ image returned will be Image Size. Client can set the wsq compression ratio using CMD_SET_OPTION call with OPTION_SET_WSQ_COMPRESSION_RATIO. Default Compression ratio used is 11:1(bit rate of 0.7273).

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	C	Opaque 1	_V100_IMAGE_TYPE	sizeof()	Type of Image to Get
B	R	Opaque 1	_V100_IMAGE_TYPE	sizeof()	Image type returned
	R	Opaque 2	unsigned int	4	Image Size
	R	Opaque 3	unsigned char	Opaque 2	Image bytes
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.13 CMD_GET_OP_STATUS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_OP_STATUS	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_OP_STATUS	None	*Z	*A

REMARKS:

Gets status related to the following commands:

VERIFICATION:

Command	**Mode**	**Parameter**
CMD_ENROLL_USER	IN_PROGRESS	Describes which finger is currently being enrolled.
	ERROR	Error code related to enrollment
	COMPLETE	None.
CMD_VERIFY_USER	IN_PROGRESS	None.
	ERROR	Error code related to verification
	COMPLETE	Match/No Match
CMD_FORMAT_DB	IN_PROGRESS	Number of Records deleted so far.
	ERROR	Error code related to formatting DB
	COMPLETE	Number of Records deleted.

IDENTIFICATION:

Command	**Mode**	**Parameter**
CMD_ID_CREATE_DB	IN_PROGRESS	% of completion of DB Creation
	ERROR	Error code related to DB Creation
	COMPLETE	None.
CMD_ID_SET_WORKING_DB	IN_PROGRESS	% complete
	ERROR	Error code related to setting working DB
	COMPLETE	None.
CMD_ID_ENROLL_USER_RECORD	IN_PROGRESS	Describes which finger is currently being enrolled.
	ERROR	Error code related to enrollment
	COMPLETE	None.
CMD_ID_IDENTIFY	IN_PROGRESS	% completion of identification
	ERROR	Error code related to identification
CMD_ID_IDENTIFY_378	IN_PROGRESS	% completion of identification
	ERROR	Error code related to identification
	COMPLETE	_V100_OP_ERROR code: STATUS_ID_USER_FOUND - User Found STATUS_ID_USER_NOT_FOUND - User not Found

Command	Mode	Parameter
CMD_ID_VERIFY_USER_RECORD	IN_PROGRESS	% completion of verification
	ERROR	Error code related to verification
	COMPLETE	_V100_OP_ERROR code: STATUS_ID_MATCH - Match STATUS_ID_NO_MATCH - No Match
CMD_ID_VERIFY_378	IN_PROGRESS	% completion of verification
	ERROR	Error code related to verification
	COMPLETE	_V100_OP_ERROR code: STATUS_ID_MATCH - Match STATUS_ID_NO_MATCH - No Match
CMD_ID_VERIFY_MANY	IN_PROGRESS	% completion of verification
	ERROR	Error code related to verification
	COMPLETE	_V100_OP_ERROR code: STATUS_ID_MATCH - Match STATUS_ID_NO_MATCH - No Match
CMD_ID_DELETE_DB	IN_PROGRESS	% complete
	ERROR	Error code related to deleting DB
	COMPLETE	None.
CMD_ID_CREATE_DB	ERROR_ID_PARAMETER	Passing parameters in using the MX00_DB_INIT_STRUCT which are out of the constraints
CMD_ID_CREATE_DB	ERROR_ID_DB_TOO_LARGE	DB too large to create. Number of templates calculated exceeds maximum limit.
CMD_ID_CREATE_DB	ERROR_ID_DB_EXISTS	DB Already exists.
CMD_ID_CREATE_DB	ERROR_ID_NOT_ENOUGH_SPACE	Not enough space on device to create the DB
CMD_ID_SET_WORKING_DB	ERROR_DB_DOES_NOT_EXIST	DB passed in does not exist or the required DB files are missing
CMD_ID_ENROLL_USER_RECORD	ERROR_DB_DOES_NOT_EXIST	nGroupID passed in using _MX00_ID_USER_RECORD does not exist or the required DB files are missing
CMD_ID_ENROLL_USER_RECORD	ERROR_ID_DB_NOT_LOADED	nGroupID passed in using _MX00_ID_USER_RECORD is not loaded into memory. This error code can be returned with FLAG_FAIL_ENROLL_ON_DUPLCATE is set and nGroupID passed in is a identification capable DB
CMD_ID_ENROLL_USER_RECORD	ERROR_ENROLLMENT_QUALIFICATION	Captured prints for enrollment didn't match.

Command	Error Code	Description
CMD_ID_ENROLL_USER_RECORD	ERROR_ID_USER_EXISTS	User-finger passed using _MX00_ID_USER_RECORD already exists in Database
CMD _ID_ENROLL_USER_RECORD	ERROR_ID_DUPLICATE	Captured prints matched with other user-finger in database
CMD_ID_ENROLL_USER_RECORD	ERROR_ID_DB_FULL	Database is full and cannot accept further user-fingers
CMD_ID_ENROLL_USER_RECORD	ERROR_ID_USER_FINGERS_FULL	User already enrolled all fingers in the database. Number of fingers each user can enroll is specified during CMD_ID_CREATE_DB call using _MX00_DB_INIT_STRUCT. Call CMD_ID_DB_METRICS to get information on DB metrics.
CMD_ID_ENROLL_USER_RECORD	ERROR_ID_USERS_FULL	Database is full with users and cannot accept new users. Number of users you may enroll is specified during CMD_ID_CREATE_DB call using _MX00_DB_INIT_STRUCT. Call CMD_ID_GET_DB_METRICS to get information on DB metrics.
CMD_ID_ENROLL_USER_RECORD	ERROR_CAPTURE_TIMEOUT	Timeout occurred during capture
CMD_ID_ENROLL_USER_RECORD	ERROR_CAPTURE_LATENT	Device detected latent
CMD_ID_ENROLL_USER_RECORD	ERROR_CAPTURE_CANCELLED	User canceled capture using CMD_CANCEL_OPERATION call
CMD_ID_ENROLL_USER_RECORD	ERROR_SPOOF_DETECTED	Device detected spoof. This error code can be returned if FLAG_FAIL_ENROLL_ON_SPOOF is set and the device supports spoof.
CMD_ID_IDENTIFY	ERROR_ID_DB_NOT_LOADED	No DB loaded into memory. Call CMD_ID_SET_WORKING_DB to load a database into memory
CMD_ID_IDENTIFY	ERROR_ID_OPERATION_NOT_SUPPORTED	Currently loaded DB is not capable of identification
CMD_ID_IDENTIFY	ERROR_CAPTURE_TIMEOUT	Timeout occurred during capture
CMD_ID_IDENTIFY	ERROR_CAPTURE_LATENT	Device detected latent
CMD_ID_IDENTIFY	ERROR_CAPTURE_CANCELLED	User canceled capture using CMD_CANCEL_OPERATION call
CMD_ID_IDENTIFY	ERROR_CAPTURE_INTERNAL	Internal error occurred during capture

Command	Error Code	Description
CMD_ID_IDENTIFY	ERROR_SPOOF_DETECTED	Device detected spoof. This error code can be returned if FLAG_FAIL_IDENTIFY_ON_SPOOF is set and the device supports spoof.
CMD_ID_IDENTIFY	ERROR_UID_DOES_NOT_EXIST	User ID does not exist. This error code can also appear if the loaded DB is empty.
CMD_ID_IDENTIFY_378	ERROR_ID_DB_NOT_LOADED	No DB loaded into memory. Call CMD_ID_SET_WORKING_DB to load a database into memory
CMD_ID_IDENTIFY_378	ERROR_ID_OPERATION_NOT_SUPPORTED	Currently loaded DB is not capable of identification
CMD_ID_IDENTIFY_378	ERROR_ID_PARAMETER	Template provided is invalid
CMD_ID_IDENTIFY_378	ERROR_UID_DOES_NOT_EXIST	User ID does not exist. This error code can also appear if the loaded DB is empty.
CMD_ID_VERIFY_USER_RECORD	ERROR_DB_DOES_NOT_EXIST	nGroupID passed in using _MX00_ID_USER_RECORD does not exist or the required DB files are missing
CMD_ID_VERIFY_USER_RECORD	ERROR_ID_USER_NOT_FOUND	User not found in the database
CMD_ID_VERIFY_USER_RECORD	ERROR_CAPTURE_TIMEOUT	Timeout occurred during capture
CMD_ID_VERIFY_USER_RECORD	ERROR_CAPTURE_LATENT	Device detected latent
CMD_ID_VERIFY_USER_RECORD	ERROR_CAPTURE_CANCELLED	User canceled capture using CMD_CANCEL_OPERATION call
CMD_ID_VERIFY_USER_RECORD	ERROR_CAPTURE_INTERNAL	Internal error occurred during capture
CMD_ID_VERIFY_USER_RECORD	ERROR_SPOOF_DETECTED	Device detected spoof. This error code can be returned if FLAG_FAIL_VERIFY_ON_SPOOF is set and the device supports spoof.
CMD_ID_VERIFY_378	ERROR_DB_DOES_NOT_EXIST	nGroupID passed in using _MX00_ID_USER_RECORD does not exist or the required DB files are missing
CMD_ID_VERIFY_378	ERROR_ID_USER_NOT_FOUND	User not found in the Database
CMD_ID_VERIFY_378	ERROR_ID_PARAMETER	Template provided is invalid
CMD_ID_VERIFY_MANY	ERROR_CAPTURE_TIMEOUT	Timeout occurred during capture
CMD_ID_VERIFY_MANY	ERROR_CAPTURE_LATENT	Device detected latent
CMD_ID_VERIFY_MANY	ERROR_CAPTURE_CANCELLED	User canceled capture using CMD_CANCEL_OPERATION call

Command	Error Code	Description
CMD_ID_VERIFY_MANY	ERROR_CAPTURE_INTERNAL	Internal error occurred during capture
CMD_ID_VERIFY_MANY	ERROR_SPOOF_DETECTED	Device detected spoof. This error code can be returned if FLAG_FAIL_VERIFY_ON_SPOOF is set and the device supports spoof.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	R	Opaque	_V100_OP_STATUS	Sizeof(…)	Op status structure.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.14 CMD_GET_SERIAL

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_SERIAL	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_CONFIG	None	*Z	*B
REMARKS: Return V100_ERROR_CODE Refer to Error code documentation for detailed description of possible return values. GEN_OK indicates operation was successful. This method of obtaining the serial number is preferred to the config method, but they are equivalent.				

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C	Opaque1	unsigned int	4	Serial Number of device
B	R	Opaque1	unsigned int	4	Serial Number of device
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.15 CMD_GET_STATUS

Command Packet					
SOF	**CMD**		**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_STATUS		None	*Z	-
Reply Packet					
SOF	**CMD**		**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_STATUS		None	*Z	See A)
REMARKS: Retrieves the `_V100_INTERFACE_STATUS_TYPE` status structure, which reports system diagnostics information.					
Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	R	Opaque1	unsigned int	4	size in bytes, of _V100_INTERFACE_STATUS_TYPE
	R	Opaque2	_V100_INTERFACE_STATUS_TYPE	sizeof _V100_INTERFACE_STATUS_TYPE	Status structure describing current system health.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.16 CMD_GET_TAG

Command Packet					
SOF	**CMD**		**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_TAG		-	*Z	-
Reply Packet					
SOF	**CMD**		**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_TAG		None	*Z	*A,*B
REMARKS: Retrieves persisted data issued by command CMD_SET_TAG. An error packet will instead be returned if record does not exist.					
Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	ARG	Unsigned short	2	Num bytes in *B
*B	C	Opaque	char	*A Bytes	Persisted data
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.17 CMD_GET_TEMPLATE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_TEMPLATE	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_TEMPLATE	None	*Z	See A

REMARKS:
Gets the latest extracted template from the device from probe template buffer. The format of the template returned corresponds to the template mode set during `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	R	Opaque 1	unsigned int	4	Size of Template
	R	Opaque 2	unsigned char	Opaque 1	Template Bytes
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.18 CMD_MATCH

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_MATCH	None	*Z	*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_MATCH	GEN_OK	*Z	*C

REMARKS:
Matches two templates, returns a score.
The format of the input templates must correspond to the template mode set during `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE..`
The ARG field specifies the format of the opaque templates and requires both templates to be of the same format.

Footnote	C/R	Category	Data Type	Size bytes	Description
B	C	Opaque 1	unsigned int	4	Template 1 Size
	C	Opaque 2	unsigned char	Opaque 1	Template 1
	C	Opaque 3	unsigned int	4	Template 2 Size
	C	Opaque 4	unsigned char	Opaque 2	Template 2
C	R	Opaque 1	unsigned int	4	Matching Score
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.19 CMD_MATCH_EX

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_MATCH_EX	*A	*Z	*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_MATCH_EX	GEN_OK	*Z	*C

REMARKS:
Matches one or two templates, returns a minutia matching score, and/or a spoof score.
The format of the input templates must correspond to the template mode set during `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`.
The ARG field specifies number of templates passed in (Either 1 or 2)

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C	ARG	short	2	# of Templates passed in
B	C	Opaque 1	unsigned int	4	Template 1 Size
	C	Opaque 2	unsigned char	Opaque 1	Template 1
	C	Opaque 3	unsigned int	4	Template 2 Size
	C	Opaque 4	unsigned char	Opaque 2	Template 2
C	R	Opaque 1	int	4	Minutia Matching Score
	R	Opaque 2	int	4	Spoof Score
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.20 CMD_PROCESS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_PROCESS	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_PROCESS	-	*Z	-

REMARKS:
Processes the current processing chain. Any previously collected data that are currently in active memory buffers will be processed according to the settings in the `_V100_INTERFACE_COMMAND_TYPE` structure. Should only be used if custom composite image is uploaded to device.

Footnote	C/R	Category	Data Type	Size bytes	Description
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.21 CMD_SAVE_LAST_CAPTURE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SAVE_LAST_CAPTURE	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SAVE_LAST_CAPTURE	None	*Z	-

REMARKS:
Saves the last transaction on the Micro-SD card. Not yet implemented.

Footnote	C/R	Category	Data Type	Size bytes	Description
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.22 CMD_SET_CMD

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_CMD	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_CMD	None	-	-

REMARKS:

Sets the `_V100_INTERFACE_COMMAND_TYPE` structure. Will return an Error packet if command type not valid for license.

Valid options in the `_V100_INTERFACE_COMMAND_TYPE` structure to set include:

Structure Field	**Description**
TimeOut	Presence Detection Timeout
Override_Trigger	Override presence detection
Override_HeartBeat_Display	Override Heartbeat
Select_PreProc	1: Create Image 0: Don't Create Image
Select_Matcher	1: Turn Matcher On, 0: Turn off
Select_Extractor	1: Turn Extractor On, 0: Turn off
Select_Spoof_Mode	1: Turn Spoof Detection On , 0: Turn off

The CMD_SET_CMD command controls the processing flow of the sensor. The recommended program flow for image acquisition and processing should usually be as follows:

1. Send a CMD_SET_CMD to set up the processing chain. For instance, if you were only interested in obtaining a Composite Image, using presence detection for improved quality, you may set the *Override_Trigger* flag to 0, the *Select_PreProc* flag to 1, and the *Select_Extractor* flag to 0.
2. Send a CMD_ARM_TRIGGER command, which then executes the processing chain which was last set. In the above case, Presence detection would fire, followed by Image acquisition, and Composite Image creation.
3. Poll for system state using CMD_ACQ_STATUS, waiting for a ACQ_DONE, or ACQ_TIMEOUT.
4. Send a CMD_GET_COMPOSITE_IMAGE to retrieve composite image and spoof information (if applicable).

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	C	Opaque1	unsigned int	4	Size of structure
	C	Opaque2	_V100_INTERFACE_COMMAND_TYPE	sizeof(…)	structure data
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.23 CMD_SET_COMPOSITE_IMAGE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_COMPOSITE_IMAGE	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_COMPOSITE_IMAGE	None	*Z	

REMARKS:
Sets the current composite image. The dimensions and image format of the image set **must** be the same as the image size returned by `CMD_GET_CONFIG`, and apply as follows:
Width: Composite_Image_Size_X
Height: Composite_Image_Size_Y
Format: 8-BPP monochrome.
This command is not supported on V30x sensors with firmware greater than 9538.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	R	Opaque 1	unsigned int	4	Image Size
	R	Opaque 2	unsigned char	Opaque 2	Image bytes
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.24 CMD_SET_GPIO

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_GPIO	-	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_GPIO	None	*Z	-

REMARKS:
Sets GPIO mask
RETURNS:
See Error code definitions for error codes related to this command.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque 1	Unsigned char	1	GPIO mask
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.25 CMD_SET_ IMAGE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_IMAGE	*B	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_IMAGE	None	*Z	

REMARKS:
Sets an image onto device. Supports all image types defined in _V100_IMAGE_TYPE.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	C	Opaque 1	unsigned int	4	Image Size
	C	Opaque 2	unsigned char	Opaque 2	Image bytes
B	C	ARG	_V100_IMAGE_TYPE	sizeof()	Image Type
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.26 CMD_SET_LED (V-Series only)

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_LED	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_LED	None	*Z	-

REMARKS:
Allows granular control of the Feedback LEDs (Red/Green).
Currently supports:

_V100_LED_CONTROL	**Description**
ALL_OFF	Turns all LEDs off
GREEN_ON	Turns Green LED on
GREEN_OFF	Turns Green LED off
RED_ON	Turns Red LED on
RED_OFF	Turns Red LED off

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	C	Opaque1	unsigned int	4	size of _V100_LED_CONTROL enum.
	C	Opaque2	_V100_LED_CONTROL	sizeof	_V100_LED_CONTROL enum
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.27 CMD_SET_OPTION

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_OPTION	*A	*Z	*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_OPTION	None	*Z	-

REMARKS:

Sets System Options. Designed to have ability to add options in the future without breaking existing structures/backwards compatibility.

Supported options include:

_V100_OPTION_TYPE	**Size**	**Option Data**	**Description**
OPTION_PD_LEVEL	4	_V100_PRESENCE_DETECTION_TYPE	Sets PD Sensitivity Level
OPTION_SET_TEMPLATE_MODE (M3xx and V31x sensors, and V30x sensors with firmware > 9538)	4	_V100_TEMPLATE_MODE	Sets template mode
OPTION_SET_WSQ_COMPRESSION_LEVEL(M30x, M31x and V31x sensors, and V30x sensors with firmware > 9538)	4	Unsigned int	Sets WSQ compression level. Values can range from >1 to<=MAX_WSQ_COMPRESSION_RATIO. Default compression ratio is 11:1
OPTION_SET_LATENT_DETECTION_MODE	4	_V100_LATENT_DETECTION_MODE	Sets latent detection mode to ON/OFF
OPTION_SET_FORCE_FINGER_LIFT_MODE (M30x sensors, and V30x sensors with firmware > 9538)	4	_V100_FORCE_FINGER_LIFT_MODE	Sets force finger lift to ON/OFF

Template types supported using OPTION_SET_TEMPLATE_MODE are as follows:

_V100_TEMPLATE_MODE	**Template type**
TEMPLATE_ANSI_378	ANSI/INCITS 378-2004(ANSI378) ANSI 378+ (for sensors that have Minex III extractor, see note in section 1.10)
TEMPLATE_ISO_NORMAL	ISO/IEC 19794-2:2005(ISO 19794-2) ISO 19794:2011 (for sensors that have Minex III extractor, see note in section 1.10)

The format of the input/output template for the following commands corresponds to template mode set using CMD_SET_OPTION call. The default template mode will be TEMPLATE_ANSI_378.

Commands
CMD_GET_TEMPLATE
CMD_MATCH
CMD_MATCH_EX
CMD_SET_TEMPLATE
CMD_TRUNCATE_378
CMD_ID_GET_USER_RECORD

CMD_ID_IDENTIFY_378 CMD_ID_SET_USER_RECORD CMD_ID_VERIFY_378 *CMD_ID_VERIFY_MANY*					
Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
A	C	ARG	unsigned short	2	_V100_OPTION_TYPE
B	C	Opaque1	unsigned int	4	Size Option Data
	C	Opaque2	unsigned char[]	Opaque1	Option Data
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.28 CMD_SET_TAG

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_TAG	-	*Z	*A,*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_TAG	None	*Z	-
REMARKS: Allows the client to persist some application specific data on the unit's flash. Useful applications of this include identifiers which allow an application to verify that "this" unit was purchased from a certain supplier. The maximum number of data that can be stored is 256 bytes. If a record exists, it is overwritten.				

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	ARG	Unsigned short	2	Num bytes in *B
*B	C	Opaque	char	*A Bytes	Data to persist
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.29 CMD_SET_TEMPLATE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_TEMPLATE	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_TEMPLATE	None	-	-

REMARKS:

Sets the gallery template buffer. Command Reserved for future use.

The format of the input template must correspond to the template mode set during `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C	Opaque 1	unsigned int	4	Size of Template
	C	Opaque 2	unsigned char	Opaque 1	Template Bytes
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.30 CMD_TRUNCATE_378

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_TRUNCATE_378	None	*Z	*A,*B,*C
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_TRUNCATE_378	None	*Z	*D,*E

REMARKS:

Truncates a template obtained from a call to CMD_GET_TEMPLATE. Note that the requested template size (*A) will usually be slightly greater than the Actual Template Size (*D) due to minutia template format conventions.

The format of the input template must correspond to the template mode set during `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C		unsigned int	4	Req. Template Size (Out)
B	C		unsigned int	4	Template Size (In)
C	C		unsigned char	*B	Template Data
D	R		unsigned int	4	Actual Template Size
E	R		unsigned char	*D	Template Data
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.31 CMD_UPDATE_FIRMWARE

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_UPDATE_FIRMWARE	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_UPDATE_FIRMWARE	None	*Z	-

REMARKS:
Updates firmware on the unit. You must poll with CMD_GET_OP_STATUS to wait for completion.

Footnote	C/R	Category	Data Type	Size bytes	Description
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

3.32 CMD_VERIFY_378

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_VERIFY_378	None	*Z	*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_VERIFY_378	GEN_OK	-	-

REMARKS:
Captures image, extracts minutia, verifies against input template.
You must poll with CMD_GET_OP_STATUS to wait for completion. After it returns completion, you must use CMD_ID_GET_RESULT in order to get the result of the identification.

Footnote	C/R	Category	Data Type	Size bytes	Description
B	C	Opaque 1	unsigned int	4	Template Size
	C	Opaque 2	unsigned char	Opaque 1	Template
Z	C	SIZE	unsigned int	4	Size of Opaque Data

3.33 CMD_VID_STREAM

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_VID_STREAM	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_VID_STREAM	None	*Z	-

REMARKS:

Set the Video Stream Mode, on or off. When video stream is on, one can retrieve the current raw image using CMD_GET_IMAGE. It is highly recommended that the only VCOM calls made to the system between modes CMD_VID_STREAM(on) and CMD_VID_STREAM(off) is CMD_GET_IMAGE. Recommended for high-bandwidth transport modes only.

Footnote	C/R	Category	Data Type	Size bytes	Description
A	C	Opaque 1	unsigned int	4	size of _V100_VID_STREAM
		Opaque 2	_V100_VID_STREAM	sizeof	VIdeo Stream mode.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

4 Diagnostic Command Descriptions

4.1 CMD_RESET

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_RESET	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_RESET	-	*Z	-
REMARKS: Performs hardware reboot of device. Reboot occurs after Reply Packet sent.				

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

5 1:1 Verification (V30x only)

The V30x sensors are capable of 1:1 verification (excludes V30x-30, V30x-40 and V30x sensors with FW 24087 and later), storing templates on the device, enrolling users, and synchronization of user databases across multiple devices. The number of records the device is capable of storing varies across product lines.

The policies associated with verification and enrollment are defined using a set of verification rules that are sent in by the client using the CMD_SET_VERIFICATION_RULES command. The verification rules structure is as below:

```
typedef struct
{
// Number of Impressions to take per enrollment
    int nNumberOfImpressions;
// Maxmimum Template Size, per impression
    int nMaxTemplateSizePerImpression;
// Inter-Enrollment Threshold
    int nInterEnrollmentThreshhold;
// Verification Threshold
    int nVerificationThreshhold;
// Spoof Threshold
    int nSpoofThreshold;
// Inter-Impression Time-Out
    int nInterImpressionTimeOutSeconds;
// Overwrite Existing Records?
    int nFlags;
}
```

5.1 Verification Rules

5.1.1 nNumberOfImpressions

The number of impressions that the device will attempt to enroll during a call to CMD_ENROLL_USER. If *nNumberOfImpressions* is set to 1 and the *nFlags* parameter has FLAG_ENROLLMENT_QUALIFICATION set, the device will attempt to verify that this is a good enrollment by automatically scanning for a second print, and comparing it against the first. In this case, only the first record is saved.

If *nNumberOfImpressions* is greater than 1, and FLAG_ENROLLMENT_QUALIFICATION is set, then each successive scanned impression will be matched against the previous scanned impressions. If the matching score in nInterEnrollmentThreshold is not met, an error will be flagged. If FLAG_ENROLLMENT_QUALIFICATION is not set, no inter-enrollment matching will be performed.

5.1.2 nMaxTemplateSizePerImpression

Maximum template size per impression. Any template which is generated by enrollment, which is greater than this size, will be reduced to this size. Matching performance can be affected by reducing template size.

5.1.3 nInterEnrollmentThreshhold

See *nNumberOfImpressions.* Default is set at 1%FRR at 1/10000 FAR.

5.1.4 nVerificationThreshhold

The matcher uses this value in order to determine whether a record matches a verification attempt. Default is set at 1%FRR at 1/10000 FAR.

5.1.5 nSpoofThreshold

Spoof threshold. Default is set at 1% FRR at a1/10000 FAR. This is only used on spoof-enabled units.

5.1.6 nInterImpressionTimeOutSeconds

During enrollment, this value determines, in seconds, when the unit times out. Default is set at 15 seconds.

5.1.7 nFlags

OR'able flags which govern unit behavior, as defined below:

5.1.7.1 FLAG_OVERWRITE_EXISTING_RECORDS

Calls to CMD_ADD_USER and CMD_ENROLL_USER will overwrite existing records corresponding to nUserID if this flag is set.

5.1.7.2 FLAG_FAIL_ENROLL_ON_SPOOF

If this flag is set, calls to CMD_ENROLL_USER will fail if a spoof is detected during enrollment.

5.1.7.3 FLAG_FAIL_VERIFY_ON_SPOOF

If this flag is set, calls to CMD_VERIFY_USER will fail if a spoof is detected during verification.

5.1.7.4 FLAG_ENROLLMENT_QUALIFICATION

If this flag is set, each successive scan during enrollment will verify against all other scans for that nUserID. If they do no match, an error occurs. Setting this flag would not allow a user to be enrolled with mutliple fingers for one user record.

5.1.7.5 FLAG_VERIFICATION_MUST_MATCH_ONE

Verification must match at least one of the enrolled scans in a user record to be considered a "Match".

5.1.7.6 FLAG_VERIFICATION_MUST_MATCH_ALL

Verification must match all scans in a user record to be considered a "Match".

5.1.7.7 FLAG_ENROLLMENT_APPEND_RECORD

If this flag is set, and FLAG_OVERWRITE_EXISTING RECORDS is not set, enrollments that are attempted using an existing nUserID will append the record to the existing UserRecord.

5.2 User Records

The commands CMD_ENROLL_USER, CMD_VERIFY_USER, CMD_ADD_USER, CMD_GET_USER, and CMD_DELETE_USER all use the _V100_USER_RECORD structure to marshal user data to and from the device. The _V100_USER_RECORD is defined as follows:

```
typedef struct
{
    unsigned int UID;                                             unsigned
int nRecords;                                               unsigned int
nSizeRecord;                                   unsigned int nSizeMetaData;
                                   unsigned char
MetaData[MAX_SIZE_META_DATA];            // Meta
} _V100_USER_RECORD;
```

5.2.1 UID

Unique 32-bit ID to assign to a user.

5.2.2 nRecords

Number of scans for this record.

5.2.3 nSizeRecord

Total Size of this record

5.2.4 nSizeMetaData

Total size of metadata.

5.2.5 MetaData

Any Meta-data you wish to associate with this user.

5.3 Flow Control Notes

Clients should set their Verification Rules upon initialization, if the default values are not desired.

Calls to CMD_ENROLL_USER, CMD_VERIFY_USER, CMD_FORMAT_DB return immediately, as do all VCOM commands. After calls to the above are issued, the client must poll using CMD_GET_OP_STATUS in order to find the status of their operation. CMD_GET_OP_STATUS returns a _V100_OP_STATUS structure which describes the current state of the command, as described below. CMD_ENROLL_USER, and CMD_VERIFY_USER may be cancelled at any time using CMD_ARM_TRIGGER. See the command descriptions for more details.

The _V100_OP_STATUS structure is described below:

```
typedef struct
{
    _V100_COMMAND_SET        eMacroCommand;
    _V100_OP_MODE            nMode;
    unsigned int       nParameter;
    _V100_ACQ_STATUS_TYPE    eAcqStatus;
} _V100_OP_STATUS;
```

eMacroCommand	Mode	Parameter
CMD_ENROLL_USER	IN_PROGRESS	Describes which finger is currently being enrolled. (1-indexed)
	ERROR	Error code related to enrollment
	COMPLETE	None

Command	Mode	Parameter
CMD_VERIFY_USER	IN_PROGRESS	None
	ERROR	Error code related to verification
	COMPLETE	Match/No Match

Command	Mode	Parameter
CMD_FORMAT_DB	IN_PROGRESS	Number of Records deleted so far
	ERROR	Error code related to formatting DB
	COMPLETE	Number of Records deleted.

6 1:1 Command Descriptions

6.1 CMD_ADD_USER

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ADD_USER	-	*Z	*A,*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ADD_USER	None	*Z	

REMARKS:
Adds a user record retrieved by CMD_GET_USER to the Database. Record is committed immediately. If the record exists already, and the `FLAG_OVERWRITE_EXISTING_RECORDS` is set during CMD_SET_VERIFICATION_RULES call, the record is overwritten, otherwise an error is returned.
This call is useful when synchronizing user records across multiple devices.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	_V100_USER_RECORD	Sizeof(...)	User Record
*B	C	Opaque	char*	*A Bytes	User Record Opaque Data
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.2 CMD_DELETE_USER

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_DELETE_USER	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_DELETE_USER	None	*Z	-

REMARKS:
Deletes User. Returns GEN_USER_NOT_FOUND if UserID is not found.
The _V100_USER_RECORD structure must have the UserID field populated.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes)**	**Description**
*A	C	Opaque	_V100_USER_RECORD	sizeof(_V100_USER_RECORD)	_V100_USER_RECORD structure
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.3 CMD_ENROLL_USER

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ENROLL_USER	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ENROLL_USER	None	*Z	-

REMARKS:

Enrolls a user based upon rules set using the CMD_SET_VERIFICATION_RULES command. This command returns immediately. Client must poll using CMD_GET_OP_STATUS until enrollment has completed. See CMD_GET_OP_STATUS for more details.

CMD_ENROLL_USER may be cancelled by the client with a call to V100_ARM_TRIGGER, with _V100_TRIGGER_MODE set to CANCEL_VERIFICATION.

The _V100_USER_RECORD structure only needs to have its nUserID, nSizeMetaData, and MetaData members populated (optionally).

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	Opaque	_V100_USER_RECORD	sizeof(_V100_USER_RECORD)	_V100_USER_RECORD structure
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.4 CMD_FORMAT_DB

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_FORMAT_DB	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_DB_METRICS	None	*Z	-

REMARKS:

Deletes all user records from database. This command returns immediately — you must poll for completion using CMD_GET_OP_STATUS in order to find when the system has finished formatting the database.

Footnote	C/R	Category	Data Type	Size bytes	Description
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.5 CMD_GET_DB_METRICS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_DB_METRICS	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_DB_METRICS	None	*Z	*A

REMARKS:
Gets database metrics. See _V100_DB_METRICS structure for more information.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	R	Opaque	_V100_DB_METRICS	Sizeof (...)	Database Metrics structure.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.6 CMD_GET_USER

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_GET_USER	*A	*Z	*B
Reply Packet				
SOF	CMD	ARG	SIZE	OPAQUE DATA
560D	CMD_GET_USER	None	*Z	*C , *D

REMARKS:
Gets user record in one of two ways:
Getting a user record by ID:
If *A == 0, the unit will expect *B1 to be a _V100_USER_RECORD
Getting a user by index:
If *A == 1, the unit will expect *B1 to be an unsigned int corresponding to the index of the record to get. In order to get the number of users enrolled on the system, use CMD_GET_DB_METRICS.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	ARG	Unsigned short	2	Either 0 or 1
*B(1)	C	Opaque	_V100_USER_RECORD	Sizeof(...)	User Record w/ ID to get
*B(2)	C	Opaque	Unsigned int	4	Index to Retrieve
*C(1)	R	Opaque	_V100_USER_RECORD	Sizeof(...)	User Record returned
*D	R	Opaque	char*	*C Bytes	User Record Opaque Data
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.7 CMD_GET_VERIFICATION_RULES

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_VERIFICATION_RULES	None	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_VERIFICATION_RULES	None	*Z	*A
REMARKS: Gets current rules for enrollment and verification. RETURNS: GEN_OK: Verification Rules retrieved successfully.				

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	R	Opaque	_V100_VERIFICATION_ RULES	sizeof(…)	Enrollment Rules Struct.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.8 CMD_SET_VERIFICATION_RULES

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_VERIFICATION_RULES	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_SET_VERIFICATION_RULES	None	*Z	-
REMARKS: Sets rules for enrollment and verification. RETURNS: GEN_OK: Verification Rules set successfully. GEN_ERR_VER_PARAMETER: Not set successfully.				

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	Opaque	_V100_VERIFICATION_ RULES	sizeof(…)	Enrollment Rules Struct.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

6.9 CMD_VERIFY_USER

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_VERIFY_USER	None	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_VERIFY_USER	None	*Z	-

REMARKS:

Performs verification of a user. _V100_USER_RECORD structure must be populated with a valid UserID.

If UserID is not found on system, GEN_ERR_UID_NOT_FOUND will be returned.

if UserID is found on system, CMD_VERIFY_USER will return immediately. The user is then verified, contingent on a successful capture. User must poll using CMD_GET_OP_STATUS for completion status.

CMD_VERIFY_USER may be cancelled by the client with a call to V100_ARM_TRIGGER, with _V100_TRIGGER_MODE set to `CANCEL_VERIFICATION`.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	_V100_USER_RECORD	sizeof(_V100_USER_RECORD)	_V100_USER_RECORD structure
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

7 1:N Identification/Verification (M-Series and V31x)

This section describes the identification features of the vCOM command set. The M30x, M31x, and V31x series sensors are capable of 1:N identification, storing templates on the device (M30x only), enrolling users, and synchronization of user databases across multiple devices. The number of records the device is capable of storing varies across product lines.

A proprietary 1:N engine is available for the V31x sensor using the Lumidigm Device Service 6.00 or higher to improve biometric performance. 1:N Identification database groups created and enrolled with v5.30.53 and earlier are supported in 6.00 in a legacy mode, but are not compatible with the new Minex III certified Extractor and Matcher which is considered a proprietary 1:N database group. To migrate to the Minex III certified proprietary 1:N database, a new proprietary database group will need to be created and users will need to be re-enrolled.

The policies associated with identification, verification and enrollment are defined using a set of identification parameters that are sent in by the client using the CMD_ID_SET_PARAMETERS command. The identification parameters structure is as below:

```
typedef struct
{
// Security level to select
    uint nSecurityLevel;
// Flags to govern behavior
    uint nFlags;
// Reserved
    uint nParam1;
    uint nParam2;
    uint nParam3;
}_MX00_ID_PARAMETERS
```

7.1 Identification Parameters

7.1.1 nSecurityLevel

The security level for identification, verification and enrollment. Valid values are

1 - Low security level

2 - Medium security level

3 - High security level

4 – Very high security level

The default level is medium security. For security applications, we recommend using the high or very high security levels.

7.1.2 nFlags

OR'able flags which govern unit behavior, as defined below:

7.1.2.1 FLAG_FAIL_ENROLL_ON_SPOOF

If this flag is set, calls to CMD_ID_ENROLL_USER_RECORD will fail if a spoof is detected during enrollment.

7.1.2.2 FLAG_FAIL_VERIFY_ON_SPOOF

If this flag is set, calls to CMD_ID_VERIFY_USER_RECORD will fail if a spoof is detected during verification.

7.1.2.3 FLAG_FAIL_IDENTIFY_ON_SPOOF

If this flag is set, calls to CMD_ID_IDENTIFY will fail if a spoof is detected during identification.

7.1.2.4 FLAG_ENROLLMENT_QUALIFICATION

If this flag is set, each successive scan during enrollment will verify against all other scans for that nUserID. If they do no match, an error occurs. Setting this flag would not allow a user to be enrolled with mutliple fingers for one user record. For Identification capable databases even if the flag is not set, successive scans are checked for enrollment qualification. By default this flag is set.

7.1.2.5 FLAG_VERIFICATION_MUST_MATCH_ONE

Verification must match at least one of the enrolled scans in a user record to be considered a “Match”. By default this flag is set.

7.1.2.6 FLAG_VERIFICATION_MUST_MATCH_ALL

Verification must match all scans in a user record to be considered a “Match”.

7.1.2.7 FLAG_FAIL_ENROLL_ON_DUPLICATE

If this flag is set, for Identfication capable databases duplicate check is performed during enrollment.

7.1.3 nParam1, nParam2, nParam3

Reserved fields.

7.2 User Records

Each user record corresponds to a user and finger which enrolled in a system. Each _MX00_ID_USER_RECORD is followed by a _MX00_ID_USER_RECORD.nInstances number of _MX00_TEMPLATE_INSTANCE records which contain the template instance information. The user record is defined as follows:

```
typedef struct
{
    uint nGroupID;
    char szUserID[32];
    uint nFinger;
    uint nInstances;
} _MX00_ID_USER_RECORD;

typedef struct
{
    uchar p378Template[MAX_378_TEMPLATE_SIZE];
} _MX00_TEMPLATE_INSTANCE;
```

7.2.1 nGroupID

Group ID of the database the user belongs to

7.2.2 szUserID

Unique ID of the user.

7.2.3 nFinger

Finger number of the user.

7.2.4 nInstances

Number of instances of the finger. This defines how many _MX00_TEMPLATE_INSTANCE records follow this structure.

7.3 Flow Control Notes

The general order of operations for identification are described in this section. See the command descriptions for more details. Clients should set their Identification Parameters, if the default values are not desired.

7.3.1 Step 1: Create the Database

Create a database using CMD_ID_CREATE_DB call. The database is stored on the device for M30x series and on the system for M31x and V31x series. The default database directory for M31x and V31x sensors is <Lumidigm Device Service installation directory>/bin/IDDB directory. User can change the database directory for M31x and V31x sensors using SEDeviceConfig GUI installed with Lumidigm Device Service. Please refer to the *Lumidigm Device Service 6.00 Migration* document for more details on database directory and SEDeviceConfig.exe. The number of databases you may create depends on sensor series and database type. The database types you can create are:

7.3.1.1 Identification and Verification Database

Identification and Verification databases has a maximum capacity of 1000 user-fingers for M30x series and 10000 user-fingers for M31x and V31x series, depending on configuration. You may create maximum 30 Identification and Verification databases with maximum capacity on M30x series. Sample configurations based on this limitation for M30x series are shown in the table below:

Configuration	Number of Users	Number of Fingers	Instances Per Finger	Max User-Fingers
A0	1000	1	3	1000
A1	500	2	3	1000
A2	200	5	3	1000

Definitions:

Number of Users:	Number of unique people in the database.
Number of Fingers:	How many fingers will be enrolled per User in database.
Instances Per Finger:	Number of impressions per finger you wish to enroll.
Max Num Templates:	Maximum number of templates stored in this configuration.

Max User-Fingers: Maximum number of user-fingers stored in this configuration.

Identificaton and verification databases require enrolling 3 instances per finger for maximum identification performance. As this does not take more storage on the device than 2 instances, or 1 instance, the only up-front cost in your client application is asking the user to place their finger three times per enrollment. Again, this is required, as it can increase performance of the system by an order of magnitude.

Please notice that configuration A has less templates which may be stored on the DB. This is because we use a template generalization model in order to attain the highest performance in the Identification system.

7.3.1.2 Verification-Only Database

Verification only databases has a maximum capacity of 200,000 templates, depending on configuration. Sample configurations based on this limitation are shown in the table below:

Configuration	Number of Users	Number of Fingers	Instances Per Finger	Max Num Templates
A0	6000	1	3	18000
A1	3000	2	3	18000
A2	1200	5	3	18000
B0	4500	1	2	9000
B1	2250	2	2	9000
B2	450	10	2	9000
C0	1000	1	1	1000
C1	500	2	1	1000
C2	100	10	1	1000

Note: Verification only databases cannot be used for identification.

7.3.1.3 DB InitializationStructure

The _MX00_DB_INIT_STRUCT must be properly populated before it is sent in using CMD_ID_CREATE_DB call:

```
typedef struct
{
    uint nGroupID;
    uint nUsers;
    uint nFingersPerUser;
    uint nInstancesPerFinger;
    uint nFlags;
    uint nReserved_0;
    uint nReserved_1;
    uint nReserved_2;
} _MX00_DB_INIT_STRUCT;
```

7.3.1.4 nGroupID

The Group ID of the database you wish to create. The Group ID is used from that point on to reference this particular database.

7.3.1.5 nUsers

Maximum number of users you will enroll in this database.

7.3.1.6 nFingersPerUser

Number of fingers per user you will enroll in this database.

7.3.1.7 nInstancesPerFinger

Instances per finger you will enroll in this database. Note that this is enforced during enrollment of a user. If FLAG_ENROLLMENT_QUALIFICATION in ID parameters is set, each finger-instance that you enroll is checked against previous instances of that finger to make certain of a good quality enrollment. For identification and verification databases instances per finger should be 3.

7.3.1.8 nFlags

Flags which govern database initialization, as defined below:

DB_INIT_FLAG_DEFAULT

This is the default mode of the identification system on the device. A database loaded using this configuration takes up to 5 seconds to load on M30x series sensors. This flag is applicable for identification and verification databases.

DB_INIT_FLAG_VERIFY_ONLY - Verify only DB

Database can only be used as verifcation DB.

7.3.1.9 nReserved_0, nReserved_1, nReserved_2

Reserved for future use.

7.3.2 Step 2: Set the working DB

Once database(s) have been created, you must activate the database you wish to use in active memory. If the database is empty, succesful completion of CMD_ID_SET_WORKING_DB allows you to enroll members into the database using CMD_ID_ENROLL_USER_RECORD, or by the CMD_ID_SET_USER_RECORD command.

Most importantly, successful completion of CMD_ID_SET_WORKING_DB allows you to identify a user, using the CMD_ID_IDENTIFY or CMD_ID_IDENTIFY_378 commands.

Note: Though some commands will function without setting a working DB, the command latency is much lower to execute those commands if the working DB is set.

Polling for completion:

CMD_ID_SET_WORKING_DB will generally return CMD_OK except when the system is busy. To check progress, you must poll the system using CMD_GET_OP_STATUS. Please see the CMD_GET_OP_STATUS reference for various conditions to expect during the execution of this command.

Completion Times:

Depending on the nFlags parameter that was passed into the CMD_ID_CREATE_DB, setting the working DB may take up to 5 seconds for the M30x series sensors with 1000 user-fingers. Smaller databases, or databases not fully populated may load much faster.

7.3.3 Step 3: Getting DB Parameters

CMD_ID_GET_DB_METRICS command may be used to get the parameters and metrics of a database. Information retrieved includes:

```
typedef struct
{
    uint nGroupID;
    uint nMaxUsers;
    uint nFingersPerUser;
    uint nInstancesPerFinger;
    int nFlags;
    int nCurEnrolledUserFingers;
int nCurEnrolledUsers;
} _MX00_DB_METRICS;
```

7.3.3.1 nGroupID

The Group ID of the database for which the parameters are retrieved.

7.3.3.2 nUsers

Maximum number of users you can enroll in this database.

7.3.3.3 nFingersPerUser

Number of fingers per user you can enroll in this database.

7.3.3.4 nInstancesPerFinger

Instances per finger you will enroll in this database

7.3.3.5 nFlags

Database initialization flags for this database.

7.3.3.6 nCurEnrolledUserFingers

Number of user-finger records currently enrolled in the database.

7.3.3.7 nCurEnrolledUsers

Number of unique users currently enrolled in the database. This represents each enrolled user has atleast one finger in the database but not necessary all the nFingersPerUser.

7.3.4 Step 4: Add user records, or enroll user-fingers

Once you have set the working group of the database, and it is loaded into memory, you may enroll users onto the database using CMD_ID_ENROLL_USER_RECORD and/or add existing user records enrolled from another sensor using CMD_ID_SET_USER_RECORD.

Note: Though these commands work without setting working DB we recommend you to set working DB before adding/enrolling user records.

If FLAG_ENROLLMENT_QUALIFICATION flag is set, each successive scan/template during enroll/set user record will verify against all other scans/templates for that nUserID. If they do not match, an error occurs. For Identification and verification databases even if the flag is not set, successive scans/templates are checked for enrollment qualification.

For M31x and V31x sensors, during enrollment a check for finger clear is done after each capture to make sure there is no valid finger placement on device platen before capturing next print. User is required to lift the finger after each capture during enrollment.

7.3.5 Step 5: Identify a User

If you have loaded a database, you may identify a user from live capture using CMD_ID_IDENTIFY and/or from a template using CMD_ID_IDENTIFY_378.

Polling for completion

CMD_ID_IDENTIFY and CMD_ID_IDENTIFY_378 will generally return CMD_OK except when the system is busy. To check progress, you must poll the system using CMD_GET_OP_STATUS. Please see the CMD_GET_OP_STATUS reference for various conditions to expect during the execution of these commands. If opertion completes successfully CMD_GET_OP_STATUS returns one of the following _V100_OP_ERROR code in nParameter member of _V100_OP_STATUS structure:

STATUS_ID_USER_FOUND	User Found
STATUS_ID_USER_NOT_FOUND	User not Found

Getting Result

After CMD_GET_OP_STATUS returns completion, you must use CMD_ID_GET_RESULT in order to get the result of the identification.

7.3.6 Step 6: Verify a User

You may verify a user from live capture using CMD_ID_VERIFY_USER_RECORD and/or from a template using CMD_ID_VERIFY_378.

Polling for completion

CMD_ID_VERIFY_USER_RECORD and CMD_ID_VERIFY_378 will generally return CMD_OK except when the system is busy. To check progress, you must poll the system using CMD_GET_OP_STATUS. Please see the CMD_GET_OP_STATUS reference for various conditions to expect during the execution of these commands. If operation completes successfully CMD_GET_OP_STATUS returns one of the following _V100_OP_ERROR code in nParameter member of _V100_OP_STATUS structure:

STATUS_ID_MATCH	Match
STATUS_ID_NO_MATCH	No match

Getting Result

After CMD_GET_OP_STATUS returns completion, you must use CMD_ID_GET_RESULT in order to get the result of the verification.

Note: We recommend you to set working DB to verify users for lower command latency.

7.3.7 Step 7: Delete a user

You can delete a user or user-finger using CMD_ID_DELETE_USER_RECORD. If the database specified in _MX00_ID_USER_RECORD is currently loaded then it is unloaded during this call. If desired, client has to call CMD_ID_SET_WORKING_DB to load DB into memory after this call.

7.3.8 Step 8: Delete Database

CMD_ID_DELETE_DB deletes specified database. If the database is currently set using CMD_ID_SET_WORKING_DB, the database is unloaded from memory.

CMD_ID_DELETE_DB is a macro command, thus you must poll for completion using CMD_GET_OP_STATUS.

8 1:N Identification/Verification (V30x-30 or V30x sensor with Firmware 24087)

The V30x sensor with a firmware rev 24087 is capable of 1:N identification and verification. The commands and interface are the same as V31x and M3xx. However its database is internal and therefore more limited:

- The V30x total user-finger limit for the device is 400 for identification.
- The V30x total template limit for the device is 2500 for verification only.

Note: These limits are independent of the number of groups used.

Note: The V30x-40 sensor (firmware 29428) does not support 1:N Identification and does not support on board template storage.

9 1:N Command Descriptions

9.1 CMD_ID_CREATE_DB

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_CREATE_DB	-	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_CREATE_DB	None	*Z	-

REMARKS:

Creates a new database. Error conditions that can occur include the DB already existing, running out of storage, or passing parameters in using the _MX00_DB_INIT_STRUCT which are out of the constraints described above.

Return values:

Will generally return CMD_OK unless the system is busy. You must use CMD_GET_OP_STATUS to poll for operation completion and/or error conditions that occurred during the database creation process. Please see CMD_GET_OP_STATUS for error codes related to this command.

Related commands:

CMD_GET_OP_STATUS, CMD_ID_DELETE_DB

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	_MX00_DB_INIT_STRUCT	Sizeof(…)	DB Initialization structure.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.2 CMD_ID_DELETE_DB

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_DELETE_DB	-	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_DELETE_DB	None	*Z	

REMARKS:

Deletes specified database. If the database is currently set using CMD_ID_SET_WORKING_DB, the database is unloaded from memory.

CMD_ID_DELETE_DB is a macro command, thus you must poll for completion using CMD_GET_OP_STATUS.

Return Codes:

See CMD_GET_OP_STATUS for error codes related to this command. You must poll for status with CMD_GET_OP_STATUS to check for completion of this command.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	unsigned int	4	Dabase number to delete
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.3 CMD_ID_DELETE_USER_RECORD

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_DELETE_USER_RECORD	*A	*Z	*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_DELETE_USER_RECORD	None	*Z	

REMARKS:

Deletes a user OR user-record from a database. Because one can enroll multiple fingers per user, CMD_ID_DELETE_USER_RECORD allows the client to choose whether to remove a user completely, or whether to remove a user-finger record.

*A: if **0**: Deletes User-Finger specified in _MX00_ID_USER_RECORD

*A: if **1:** Deletes all user records corresponding to the User field in _MX00_ID_USER_RECORD

*B: User record structure. If ARG is 0, the szUserID and nFinger elements of the _MX00_ID_USER_RECORD structure are considered. If ARG is 1, only the szUserID is considered.

If the database specified in _MX00_ID_USER_RECORD is currently loaded then it is unloaded from memory during this call. If desired client has to call CMD_ID_SET_WORKING_DB to load DB into memory after this call.

RETURNS:

Unless the system is busy, the CMD_ID_DELETE_USER_RECORD command will usually return GEN_OK as long as the command is well-formed.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Arguement	unsigned short	4	Arguement
*B	C	Opaque	_MX00_ID_USER_RECORD	Sizeof(…)	Header structure
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.4 CMD_ID_ENROLL_USER_RECORD

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_ENROLL_USER_RECORD	-	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_ENROLL_USER_RECORD	None	*Z	-

REMARKS:

Begins the enrollment process. CMD_ID_ENROLL_USER_RECORD is a macro command, thus you must use CMD_GET_OP_STATUS to poll for completion/errors.

Users are enrolled in the system using the "rules" set using the CMD_ID_SET_PARAMETERS call.

You must call CMD_ID_ENROLL_USER_RECORD for each User-Finger you wish to enroll. However, this command does automatically enroll multiple instances of said User-Finger.

If the database specified in the nGroupID member of the _MX00_ID_USER_RECORD has multiple instances per user finger specified, CMD_ID_ENROLL_USER will attempt to enroll that many instances of the user's finger. If the nFlags of CMD_ID_SET_PARAMETER contains the `FLAG_ENROLLMENT_QUALIFICATION` flag, each instance of the captured print is checked against each other captured print in order to ensure enrollment quality.

Note: The FLAG_ENROLLMENT_QUALIFICATION is a requirement when enrolling into any database which is capable of identification. This is a requirement to ensure good identification performance.

If the `FLAG_FAIL_ENROLL_ON_DUPLICATE` flag is set, for Identification capable databases check for duplicate is performed. If the captured prints match with a user-finger in the database, CMD_GET_OP_STATUS returns "ERROR_ID_DUPLICATE" _V100_OP_ERROR code in nParameter member of _V100_OP_STATUS structure. You may call CMD_ID_GET_RESULT to get the status/user information.

If the `FLAG_FAIL_ENROLL_ON_SPOOF` flag is set, a check for spoof is performed for each instance of the captured print.

For M31x and V31x sensors, a check for finger clear is done after each capture to make sure there is no valid finger placement on device platen before capturing next print. User is required to lift the finger after each capture during enrollment. For M30x sensors and V30x sensors with firmware greater than 9538 sensors, user can set the force finger lift for enrollment using V100_Set_Option call with OPTION_SET_FORCE_FINGER_LIFT_MODE as the _V100_OPTION_TYPE `and` FORCE_FINGER_LIFT_MODE_ON `as` the option data.

RETURNS:

This command generally returns GEN_OK, unless the system is busy. See CMD_GET_OP_STATUS for extended definitions of error codes.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	`_MX00_ID_USER_RECORD`	Sizeof(…)	_MX00_ID_USER_RECOR D
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.5 CMD_ID_GET_DB_METRICS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_DB_METRICS	*A	*Z	*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_DB_METRICS	None	*Z	*C

REMARKS:

Gets the metrics and parameters of a database.

If Argument *A = 0, then the system will use the nGroupID member of the _MX00_DB_METRICS structure that is sent in, in order to determine which database you would like the metrics for.

If Argument *A = 1, then the system will retrieve the database metrics of the currently loaded group. If no group is currently loaded, an error will be returned.

Return Codes:

See Error code definitions for error codes related to this command.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	Argument	Short	2	Arguement
*B	C	Opaque	_MX00_DB_METRICS	Sizeof(…)	Database metrics structure.
*C	R	Opaque	_MX00_DB_METRICS	Sizeof(…)	Database metrics structure.

9.6 CMD_ID_GET_PARAMETERS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_PARAMETERS	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_PARAMETERS	None	*Z	*A

REMARKS:

Execute this command to retrieve the current ID parameters.

`See CMD_ID_SET_PARAMETERS for structure definition.`

Return Codes:

See Error code definitions for error codes related to this command.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	Opaque	_MX00_ID_PARAMETERS	Sizeof(…)	The ID Parameters structure
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.7 CMD_ID_GET_RESULT

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_RESULT	-	*Z	
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_RESULT	None	*Z	*A

REMARKS:

Retrieve the result of the last successful identify executed. This includes successful completion of the commands CMD_ID_IDENTIFY_378, CMD_ID_IDENTIFY, CMD_ID_VERIFY_378 and CMD_ID_VERIFY_USER_RECORD as polled by CMD_GET_OP_STATUS.

This command returns the `_MX00_ID_RESULT` structure.

```
eLastStatus - Status of the last operation
szUserID - User found
nFinger - Finger found
nIDScore - ID Score
nM1Score - Internal use only
nM2Score - Internal use only
nSpoofScore - If supported, spoof score otherwise -1
nIDTime - Internal use only
nC1Time - Internal use only
nC2TIme - Internal use only
```

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	R	Opaque	_MX00_ID_RESULT	Sizeof	The results structure.
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.8 CMD_ID_GET_SYSTEM_METRICS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_DB_METRICS	-	*Z	
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_DB_METRICS	*A	*Z	*B

REMARKS:

Provides information on the database groups currently loaded on the system.

The response packet is structured as follows:

The ARG field contains the number of sets, thus the number of _MX00_DB_METRICS which follow in the opaque data field.

In the _MX00_DB_METRICS structure **`nCurEnrolledUserFingers`** and **`nCurEnrolledUsers`** members returned by this call should be ignored since these are not valid values. To get this information use CMD_ID_GET_DB_METRICS.

Return Codes:

See Error code definitions for error codes related to this command.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	R	ARG	Short	2	Num _MX00_DB_METRICS Records to follow
*B	R	Opaque	_MX00_DB_METRICS	Sizeof(…)	_MX00_DB_METRICS records, quantity *A

9.9 CMD_ID_GET_USER_RECORD

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_USER_RECORD	*A	*Z	*B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_USER_RECORD	None	*Z	*C, *D

REMARKS:

This command retrieves a user record from a database.

*A = -1: The structure the *B field contains is used to indicate which user-finger you would like to retrieve.

*A = **n**: This will retrieve the user-finger stored at 0-based *index* **n,** where **n** is between 0 and the **`nCurEnrolledUserFingers`** field retrieved during a call to CMD_ID_GET_DB_METRICS**.** This is useful for retrieving user-finger entries in the database sequentially, when user-finger information is unknown. The *B field is ignored except **nGroupID** of the structure when the ARG field is positive.

*B: A _MX00_ID_USER_RECORD structure. The **nInstances** of the structure is ignored.

Return value:

*C: `_MX00_ID_USER_RECORD:`

User record header structure. The nInstances member of this structure defines how many _MX00_TEMPLATE_INSTANCE structures follow this structure.

`*D: _MX00_TEMPLATE_INSTANCE(s):`

nInstances of _MX00_TEMPLATE_INSTANCE structures which hold the templates. The format of the templates returned corresponds to the template mode set during the CMD_SET_OPTION call using OPTION_SET_TEMPLATE_MODE.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	ARG	Short	2	Argument
*B	C	Opaque	_MX00_ID_USER_RECORD	sizeof(…)	User Record
*C	R	Opaque	_MX00_ID_USER_RECORD	Sizeof(…)	User Record
*D	R	Opaque	_MX00_TEMPLATE_INSTANCE	nInstances*Sizeof(…)	Template Instance
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.10 CMD_ID_GET_USER_RECORD_HEADER

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_USER_RECORD	*A	*Z	
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_GET_USER_RECORD	None	*Z	*C

REMARKS:

This command retrieves a user record header from working database.

*A = **n**: This will retrieve the user-finger stored at 0-based *index* **n,** where **n** is between 0 and the **`nCurEnrolledUserFingers`** field retrieved during a call to CMD_ID_GET_DB_METRICS**.** This is useful for retrieving user-finger entries in the database sequentially, when user-finger information is unknown.

Return value:

*C: `_MX00_ID_USER_RECORD:`

`User record header structure.`

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	ARG	Short	2	Argument
*C	R	Opaque	_MX00_ID_USER_RECORD	Sizeof(…)	User Record
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.11 CMD_ID_IDENTIFY

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_IDENTIFY	-	*Z	-
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_IDENTIFY	None	*Z	-

REMARKS:

Begins the Identification process. CMD_ID_IDENTIFY is a macro command, thus you must poll for completion using CMD_GET_OP_STATUS. Please see CMD_GET_OP_STATUS for expected values.

After CMD_GET_OP_STATUS returns completion, you must use CMD_ID_GET_RESULT in order to get the result of the identification.

A database must be loaded using CMD_ID_SET_WORKING_DB in order for this command to succeed.

CMD_ID_IDENTIFY works in two stages. Stage 1 is capturing the fingerprint. Polling CMD_GET_OP_STATUS during this stage will allow you to get status on the capture, and will also return error conditions in case of time-outs, latent prints detected, etc. If the FALG_FAIL_IDENTIFY_ON_SPOOF flag is set, a check for spoof is performed.

After a successful capture, CMD_ID_IDENTIFY will begin the identification phase. After Identification is complete, you may call CMD_ID_GET_RESULT to get the status information related to the last identification performed.

RETURNS:

This command generally returns GEN_OK, unless the system is busy. See CMD_GET_OP_STATUS for extended definitions of error codes.

Footnote	C/R	Category	Data Type	Size bytes	Description
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.12 CMD_ID_IDENTIFY_378

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_IDENTIFY_378	-	*Z	*A, *B
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_IDENTIFY_378	None	*Z	-

REMARKS:

Identifies a user from a template*. CMD_ID_IDENTIFY_378 is a macro command, thus you must poll for completion using CMD_GET_OP_STATUS. Please see CMD_GET_OP_STATUS for expected values.

After CMD_GET_OP_STATUS returns completion, you must use CMD_ID_GET_RESULT in order to get the result of the identification.

A database must be loaded using CMD_ID_SET_WORKING_DB in order for this command to succeed.

* The format of the input template must correspond to the template mode set during the `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`.

RETURNS:

This command generally returns GEN_OK, unless the system is busy. See CMD_GET_OP_STATUS for extended definitions of error codes.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	Unsigned int	4	Size of template to be sent.
*B	C	Opaque	uchar[*A]	*A	Template
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.13 CMD_ID_SET_PARAMETERS

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_SET_PARAMETERS	-	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_SET_PARAMETERS	None	*Z	-

REMARKS:

Sets the parameters which the identification engine uses to operate. The _MX00_ID_PARAMETERS structure is defined below:

```
typedef struct
{
// Security level to select
                uint nSecurityLevel;
// Flags to govern behavior
                uint nFlags;
// Reserved
                uint nParam1;
                uint nParam2;
                uint nParam3;
}
```

nSecurityLevel

The security level for identification, verification and enrollment. Valid values are

1 - Low security level

2 - Medium security level

3 - High security level

4 – Very high security level

The default level is medium security. For security applications, we recommend using the high or very high security levels.

nFlags

OR’able flags which govern unit behavior, as defined below:

FLAG_FAIL_ENROLL_ON_SPOOF

If this flag is set, calls to CMD_ID_ENROLL_USER_RECORD will fail if a spoof is detected during enrollment.

FLAG_FAIL_VERIFY_ON_SPOOF

If this flag is set, calls to CMD_ID_VERIFY_USER_RECORD will fail if a spoof is detected during verification.

FLAG_FAIL_IDENTIFY_ON_SPOOF

If this flag is set, calls to CMD_ID_IDENTIFY will fail if a spoof is detected during identification.

FLAG_ENROLLMENT_QUALIFICATION

If this flag is set, each successive scan during enrollment will verify against all other scans for that nUserID. If they do no match, an error occurs. Setting this flag would not allow a user to be enrolled with mutliple fingers for one user record.

FLAG_VERIFICATION_MUST_MATCH_ONE

Verification must match at least one of the enrolled scans in a user record to be considered a “Match”.

FLAG_VERIFICATION_MUST_MATCH_ALL

Verification must match all scans in a user record to be considered a “Match”.

FLAG_FAIL_ENROLL_ON_DUPLICATE
If this flag is set, for Identfication capable databases duplicate check is performed during enrollment.
nParam1, nParam2, nParam3
Reserved fields.
Return Codes:
See Error code definitions for error codes related to this command.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	Opaque	_MX00_ID_PARAMETERS	Sizeof(...)	The ID Parameters structure
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.14 CMD_ID_SET_USER_RECORD

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_SET_USER_RECORD	-	*Z	*A, *B_
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_SET_USER_RECORD	None	*Z	

REMARKS:
This command adds a user record to a database. The user record consists of two parts:
`_MX00_ID_USER_RECORD: User record header structure. The nInstances member of this structure defines how many _MX00_TEMPLATE_INSTANCE structures follow this structure.`
`_MX00_TEMPLATE_INSTANCE:` **`nInstances`** `of _MX00_TEMPLATE_INSTANCE structures which hold the templates*.`
If the database specified in the nGroupID member of the _MX00_ID_USER_RECORD has multiple instances per user finger and the nFlags of CMD_ID_SET_PARAMETER contains the `FLAG_ENROLLMENT_QUALIFICATION` flag, each template instance is checked against each other template in order to ensure enrollment quality.
Note: The FLAG_ENROLLMENT_QUALIFICATION is a requirement when adding user record into any database which is capable of identification. This is a requirement to ensure good identification performance.
After adding all the user records using this command you must call CMD_ID_SET_WORKING_DB in order to commit all the records to the device.
* The format of the input templates must correspond to the template mode set during the `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`.
Return value:
See error code descriptions for return value description.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	Opaque	_MX00_ID_USER_RECORD	Sizeof(...)	Header structure
*B(multiple)	C	Opaque	_MX00_TEMPLATE_INSTANCE	nInstances*sizeof(...)	Template Instance
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.15 CMD_ID_SET_WORKING_DB

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_SET_WORKING_DB	-	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_SET_WORKING_DB	None	*Z	

REMARKS:

Sets the working database. Database must have been created using CMD_ID_CREATE_DATABASE for this command to succeed. If the database is empty, this command will allow you to enroll users into the database using CMD_ID_ENROLL_USER_RECORD, or will allow you to add existing records using CMD_ID_SET_USER_RECORD. If this database is populated, it will load into active memory. Once database is in active memory, CMD_ID_IDENTIFY, or CMD_ID_IDENTIFY_378 may be used to identify.

Return Values:

You must poll for status with CMD_GET_OP_STATUS to check for completion of this command. Please see CMD_GET_OP_STATUS for error codes related to this command.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	Unsigned int	4	Working DB To Set.
Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.16 CMD_ID_VERIFY_378

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_VERIFY_378	*D	*Z	*A,*B,*C
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_VERIFY_378	None	*Z	-

REMARKS:

Verifies template *C against user record corresponding to *A. CMD_ID_VERIFY_378 is a macro command, thus you must poll for completion using CMD_GET_OP_STATUS. Please see CMD_GET_OP_STATUS for expected values. The format of the input template must correspond to the template mode set during the `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`.

*D =0: The nFinger member of _MX00_USER_RECORD structure is ignored and specified user is verified considering all the fingers enrolled for that user.
*D =1: User-finger specified in _MX00_USER_RECORD structure is verified.

This command should be used to verify an existing 378 template against a user record present in the database. You must populate all members of the _MX00_USER_RECORD structure you send in, except for the nInstances member, which is ignored.

When this command completes, use CMD_ID_GET_RESULT to determine whether or not there was a match.

A database must be loaded using CMD_ID_SET_WORKING_DB in order for this command to succeed.

RETURNS:

This command generally returns GEN_OK, unless the system is busy. See CMD_GET_OP_STATUS for extended definitions of error codes.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	_MX00_USER_RECORD	Sizeof(…)	The user record to verify against
*B	C	Opaque	Unsigned int	4	Size of template to be passed in
*C	C	Opaque	Char	[*B]	Template
*D	C	ARG	Short	2	Indicates whether to consider the specified finger or not
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.17 CMD_ID_VERIFY_MANY

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_VERIFY_MANY	-	*Z	*A*B*C
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_VERIFY_MANY	None	*Z	-

REMARKS:

This command captures a finger print and verifies it against a set of input templates. The format of the input templates must correspond to the template mode set during the `CMD_SET_OPTION` call using `OPTION_SET_TEMPLATE_MODE`. Max 30 input templates are allowed. CMD_ID_VERIFY_MANY is a macro command, thus you must poll for completion using CMD_GET_OP_STATUS. Please see CMD_GET_OP_STATUS for expected values.

After CMD_GET_OP_STATUS returns completion, you must use CMD_ID_GET_RESULT in order to get the result of the verification.

CMD_ID_VERIFY_MANY works in two stages. Stage 1 is capturing the fingerprint. Polling CMD_GET_OP_STATUS during this stage will allow you to get status on the capture, and will also return error conditions in case of time-outs, latent prints detected, etc. If the FLAG_FAIL_VERIFY_ON_SPOOF flag is set, a check for spoof is performed.

After a successful capture, CMD_ID_VERIFY_MANY will begin the verification phase. If the operation completes successfully CMD_GET_OP_STATUS returns following _V100_OP_ERROR codes in nParameter member of _V100_OP_STATUS structure

```
STATUS_ID_MATCH   -  Match
STATUS_ID_NO_MATCH  -  No Match
```

You may then call CMD_ID_GET_RESULT to get the status information related to the last verification performed. CMD_ID_GET_RESULT returns the `_MX00_ID_RESULT` structure with the following information for CMD_ID_VERIFY_MANY call.

```
eLastStatus - Status of the last operation. For successful operation
STATUS_ID_MATCH  or STATUS_ID__NO_MATCH
szUserID - index(0-based) of the input template which has highest     match score
nFinger - NA
nIDScore - match score
nM1Score - Internal use only
nM2Score - Internal use only
nSpoofScore - If supported, spoof score otherwise -1
nIDTime - Internal use only
nC1Time - Internal use only
nC2TIme - Internal use only
```

RETURNS:

This command generally returns GEN_OK, unless the system is busy. See CMD_GET_OP_STATUS for extended definitions of error codes.

Footnote	**C/R**	**Category**	**Data Type**	**Size bytes**	**Description**
*A	C	Opaque	Unsigned int	4	Number of input templates
*B(multiple)	C	Opaque	Unsinged int	[*A]*4	Input templates sizes
*C(multiple)	C	Opaque	char	[*B]	Input templates
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

9.18 CMD_ID_VERIFY_USER_RECORD

Command Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_VERIFY_USER_RECORD	*B	*Z	*A
Reply Packet				
SOF	**CMD**	**ARG**	**SIZE**	**OPAQUE DATA**
560D	CMD_ID_VERIFY_USER_RECORD	None	*Z	-

REMARKS:

Begins the verification process. CMD_ID_VERIFY_USER_RECORD is a macro command, thus you must poll for completion using CMD_GET_OP_STATUS. Please see CMD_GET_OP_STATUS for expected values.

*D =0: The nFinger member of _MX00_USER_RECORD structure is ignored and specified user is verified considering all the fingers enrolled for that user.
*D =1: User-finger specified in _MX00_USER_RECORD structure is verified.

This command should be used to verify against a user record present in the database. You must populate all members of the _MX00_USER_RECORD structure you send in, except for the nInstances member.

After CMD_GET_OP_STATUS returns completion, you must use CMD_ID_GET_RESULT in order to get the result of the verification.

A database must be loaded using CMD_ID_SET_WORKING_DB in order for this command to succeed.

CMD_ID_VERIFY_USER_RECORD works in two stages. Stage 1 is capturing the fingerprint. Polling CMD_GET_OP_STATUS during this stage will allow you to get status on the capture, and will also return error conditions in case of time-outs, latent prints detected, etc. If the FLAG_FAIL_VERIFY_ON_SPOOF flag is set, a check for spoof is performed.

After a successful capture, CMD_ID_VERIFY_USER_RECORD will begin the verification phase. After verification is complete, you may call CMD_ID_GET_RESULT to get the status information related to the last verification performed.

RETURNS:

This command generally returns GEN_OK, unless the system is busy. See CMD_GET_OP_STATUS for extended definitions of error codes.

Footnote	C/R	Category	Data Type	Size bytes	Description
*A	C	Opaque	_MX00_ID_USER_RECORD	Sizeof(…)	The user record to verify against
*B	C	ARG	Short	2	Indicates whether to consider the specified finger or not
*Z	C/R	SIZE	unsigned int	4	Size of Opaque Data

Appendix: A Data Type Definitions

A.1 _V100_Interface_Configuration_Type

This is a READ only configuration structure providing hardware and software revision information, physical interface and software service availability. Reserved fields are for future expansion and/or customer specific configuration builds.

```
typedef struct {

unsigned short
Vendor_Id,      // Vendor Identification
Product_Id,     // Product Identification
Device_Serial_Number, // Unique Device Serial Number
Hardware_Rev,   // HW Revision Number       (xxx.xxx.xxx)
Firmware_Rev,   // FW Revision Number       (xxx.xxx.xxx)
Spoof_Rev,      // Spoof Revision Number    (xxx.xxx.xxx)
PreProc_Rev,    // PreProcessor Revision Number    (xxx.xxx.xxx)
Feature_Extractor_Rev,      // Feature Extractor Revision Number     (xxx.xxx.xxx)
Matcher_Rev,    // Matcher Revision Number  (xxx.xxx.xxx)
ID_Rev, // Identifier Revision Number      (xxx.xxx.xxx)
Imager_Chip_Version,  // Imager Chip Silicon Version
Number_Image_Planes,  // Number of Image Planes in Native Image Stack
Native_Image_Size_X,  // Number of Pixels in Column (Native)
Native_Image_Size_Y,  // Number Pixels in Rows (Native)
Native_DPI,     // Native Resolution Pixel Dots/Inch
Composite_Image_Size_X,      // Pixels in Col. Composite Processed Image (500 dpi)
Composite_Image_Size_Y,      // Pixels in Row Composite Processed Image (500 dpi)
Composite_DPI, // Composite Image Resolution
Image_Format,   // Format of Native Images (_V100_IMAGE_FORMAT_TYPE)
Boresight_X,    // Pixel Units wrt Native Coords
Boresight_Y,    // Pixel Units wrt Native Coords
LED_Wavelength1,      // Nanometers (State Ordered)
LED_Wavelength2,      // Nanometers (State Ordered)
LED_Wavelength3,      // Nanometers (State Ordered)
LED_Wavelength4,      // Nanometers (State Ordered)
LED_Type1,      // (_V100_LED_TYPE)
LED_Type2,      // (_V100_LED_TYPE)
LED_Type3,      // (_V100_LED_TYPE)
LED_Type4,      // (_V100_LED_TYPE)
State1_Exposure,      // Current Exp Value from Last Image Acquisition
State2_Exposure,
State3_Exposure,
State4_Exposure,
State1_Gain,    // Current Gain Value from Last Image Acquisition
State2_Gain,
State3_Gain,
State4_Gain,
Phy_Interface_Available,     // Physical Interfaces Available
(_V100_PHY_INTERFACE_TYPE)
PreProc_Available,    // PreProcessors Available (_V100_PREPROC_TYPE)
Feature_Extract_Available,   // Feature Extractors Available (_V100_FE_TYPE)
Template_Match_Available,    // Template Matchers Available (_V100_FM_TYPE)
Template_Format_Available,   // Template Formats Available _V100_TEMPLATE_FORMAT_TYPE)
Template_ID_Available,       // Template Identification Available ( _V100_ID_TYPE)
Presence_Detect_Available,   // Presence Detection (_V100_PRESENCE_DETECTION_TYPE)
FW_Flash_Available,   // In-circuit Flash Capability Available (_V100_FLASH_TYPE)
Spoof_Available,      // Spoof Algorithms Available (_V100_SPOOF_TYPE)
Struct_Size;   // Size in Bytes of This Structure
unsigned int
pImageBuffer,  // For internal use
pPDBuffer,     // For internal use
pStaticMask,   // For internal use
pDarkBuffer,   // For internal use
pCompositeBuffer;     // For internal use

unsigned short
Device_Serial_Number_Ex,     // Extended Serial Number data (Upper Word of SN on device
-   M31x and later devices only)
MfgStateFlag,  // Mfg State Flag (Will be SPT Rev in early M31x units)
RESERVED_2,    // RESERVED for Future Use
RESERVED_3,    // RESERVED for Future Use
RESERVED_4,    // RESERVED for Future Use
RESERVED_5;    // RESERVED for Future Use
} _V100_INTERFACE_CONFIGURATION_TYPE;
```

A.2 _V100_Interface_Command_Type

This is a READ/WRITE control structure for issuing commands to the device. User selection of available services and device behavior are found here.

```
typedef struct
{
/*** USER COMMANDS ***/
unsigned short
Trigger_Delay, // (msec) Delay btw Pres. Detect and Acq
TimeOut,// (sec) Timeout on Blocking Ops
Override_Trigger,     // if TRUE, override presence detection
Override_HeartBeat_Display,  // if TRUE, Turns off HeartBeat
Override_Default_LED, // if TRUE, Turns off default LED behavior
Match_Threshold,      // Thresold Score for Matching
Set_Exposure_Mode,    // Manual or Auto Exposure Mode
Select_PreProc,       // Bitfield to Select PreProcessor
Select_Identifier,    // Bitfield to Select Identifier
Select_Matcher,       // Bitfield to Select Matcher
Select_Extractor,     // Bitfield to Select Extractor
Select_Spoof_Model,   // Bitfield to Select Spoof Model
Select_Template_Format,      // Template Format
Exposure1,     // Reserved for internal use
Exposure2,     // Reserved for internal use
Exposure3,     // Reserved for internal use
Exposure4,     // Reserved for internal use
Gain1,  // Reserved for internal use
Gain2,  // Reserved for internal use
Gain3,  // Reserved for internal use
Gain4,  // Reserved for internal use
Struct_Size;   // Size in Bytes of This Structure

unsigned short
RESERVED_0,    // RESERVED for Future Use
RESERVED_1,    // RESERVED for Future Use
RESERVED_2,    // RESERVED for Future Use
RESERVED_3;    // RESERVED for Future Use

unsigned short
PAD_0,
PAD_1,
PAD_2,
PAD_3,
PAD_4,
PAD_5;
} _V100_INTERFACE_COMMAND_TYPE;
```

A.3 _V100_Interface_Status_Type

This is a READ only structure returning the complete status and condition of the device. Start up error, runtime errors, and built-in-test results are latched into these registers. Reading the registers will clear the latched error condition.

```
typedef struct
{
u16
General_Error,   // cast as _V100_General_Error_Codes
Service_Error,   // cast as _V100_Service_Error_Codes
BIT_Status,      // cast as _V100_BIT_Error_Codes
COM_Error,       // cast as _V100_I2C_Error_Codes
Struct_Size
RESERVED_0,      // Expansion
RESERVED_1,      // Expansion
RESERVED_2,      // Expansion
RESERVED_3,      // Expansion
} _V100_Interface_Status_Type;
```

Appendix: B Error Handling

A.1 Error Response Packets

If an error condition occurs after the client sends the Command Packet, the host will return a CMD_ERROR packet.

_V100_GENERAL_ERROR_CODES	Description
GEN_ERROR_APP_BUSY	Device Busy (Macro Command Operating).
GEN_ERROR_COMM_TIMEOUT	RS-232 error occurred
GEN_ERROR_DB_DOESNOT_EXIST	Database doesnot exist
GEN_ERROR_DB_FULL	Device Database Full
GEN_ERROR_DB_NOT_LOADED	Database not loaded in to active memory
GEN_ERROR_DB_USER_FINGERS_FULL	User already enrolled all fingers in the database
GEN_ERROR_DB_USERS_FULL	Database is full with users and cannot accept new users
GEN_ERROR_DEVICE_NOT_FOUND	Invalid Device ID
GEN_ERROR_DEVICE_NOT_READY	M31x or V31x Device not ready to accept commands
GEN_ERROR_DEVICE_UNCONFIGURED	Device unconfigured
GEN_ERROR_ENROLLMENTS_DO_NOT_MATCH	Enrollment templates do not match
GEN_ERROR_INTERNAL	Unhandled error occurred
GEN_ERROR_INVALID_CMD	Command undefined, and not supported
GEN_ERROR_INVALID_SIZE	Invalid size of parameter passed in
GEN_ERROR_LICENSE	Inadequate license to execute command
GEN_ERROR_MATCH	Error occurred during matching
GEN_ERROR_MEMORY	Out of memory
GEN_ERROR_PARAMETER	Bad parameter passed into command
GEN_ERROR_PIPE_READ	Communication error with LumiDevice Service
GEN_ERROR_PIPE_WRITE	Communication error with LumiDevice Service
GEN_ERROR_PROCESSING	Error occurred during processing
GEN_ERROR_RECORD_NOT_FOUND	User record not found
GEN_ERROR_SENGINE_SHUTTING_DOWN	M31x or V31x devices can no longer accept commands
GEN_ERROR_TAG_NOT_FOUND	Cannot find User Tag
GEN_ERROR_TIMEOUT	A Timeout occurred
GEN_ERROR_TIMEOUT_LATENT	Device detected latent and timed out
GEN_ERROR_USER_EXISTS	User already exists
GEN_ERROR_USER_NOT_FOUND	User not found in the Database
GEN_ERROR_VERIFY	Error occurred during verification
GEN_FS_ERR_CD	Used Internally
GEN_FS_ERR_DELETE	Used Internally
GEN_FS_ERR_FIND	Used Internally
GEN_FS_ERR_FORMAT	Used Internally

_V100_GENERAL_ERROR_CODES	Description
GEN_FS_ERR_READ	Used Internally
GEN_FS_ERR_WRITE	Used Internally
GEN_INVALID_ARGUEMENT	Invalid argument passed
GEN_NOT_SUPPORTED	Command defined, but not supported
GEN_OK	Command executed successfully
GEN_VER_INVALID_RECORD_FORMAT	Invalid/Corrupted User Record format

A.2 _V100_SERVICE_ERROR_CODES

Enumerated type for device service error codes.

```
/*
** Status Structure Error Codes
*/
typedef enum
{
SRV_OK     = 0x0000,
SRV_IM_ERROR     = 0x0001,    // Interrupt Manager
SRV_EBIU_ERROR   = 0x0002,    // EBIU
SRV_PM_ERROR     = 0x0004,    // Power Manager
SRV_DCB_ERROR    = 0x0008,    // Deffered Callback Manager
SRV_DMA_ERROR    = 0x0010,    // DMA Manager
SRV_INT_ERROR    = 0x0020,    // Interrupt Hook
SRV_DEV_ERROR    = 0x0040,    // Device Driver
SRV_PF_ERROR     = 0x0080,    // Program Flag Service
SRV_CAM_ERROR    = 0x0100,    // Camera Comm
SRV_UART_ERROR   = 0x0200,    // UART Error
SRV_USB_ERROR    = 0x0400,    // USB Error
} _V100_Service_Error_Codes;
```

A.3 _V100_OP_ERROR

Enumerated type for operational status codes.

```
typedef enum
{
STATUS_OK = 0x00,
ERROR_UID_EXISTS= 0x01,
ERROR_ENROLLMENT_QUALIFICATION     = 0x02,
ERROR_UID_DOES_NOT_EXIST     = 0x03,
ERROR_DB_FULL    = 0x04,
ERROR_QUALIFICATION    = 0x05,
ERROR_DEV_TIMEOUT      = 0x06,
ERROR_USER_CANCELLED   = 0x07,
ERROR_SPOOF_DETECTED   = 0x08,
ERROR_DB_EXISTS = 0x09,
ERROR_DB_DOES_NOT_EXIST      = 0x0A,
/* Identification Codes */
ERROR_ID_DB_TOO_LARGE = 0x10,
ERROR_ID_DB_EXISTS     = 0x11,
ERROR_ID_USER_EXISTS   = 0x12,
ERROR_ID_USER_NOT_FOUND      = 0x13,
STATUS_ID_MATCH = 0x14,
STATUS_ID_NO_MATCH     = 0x15,
ERROR_ID_PARAMETER     = 0x16,
ERROR_ID_GENERAL= 0x17,
ERROR_ID_FILE    = 0x18,
ERROR_ID_NOT_INITIALIZED     = 0x19,
ERROR_ID_DB_FULL= 0x1A,
ERROR_ID_DB_DOESNT_EXIST     = 0x1B,
ERROR_ID_DB_NOT_LOADED= 0x1C,
ERROR_ID_RECORD_NOT_FOUND    = 0x1D,
ERROR_ID_FS      = 0x1E,
ERROR_ID_CREATE_FAIL   = 0x1F,
ERROR_ID_INTERNAL      = 0x20,
ERROR_ID_MEM     = 0x21,
STATUS_ID_USER_FOUND   = 0x22,
STATUS_ID_USER_NOT_FOUND     = 0x23,
ERROR_ID_USER_FINGERS_FULL   = 0x24,
ERROR_ID_USER_MULTI_FINGERS_NOT_FOUND    = 0x25,
ERROR_ID_USERS_FULL    = 0x26,
ERROR_ID_OPERATION_NOT_SUPPORTED   = 0x27,
ERROR_ID_NOT_ENOUGH_SPACE    = 0x28,
ERROR_ID_DUPLICATE     = 0x29,
/* Capture */
ERROR_CAPTURE_TIMEOUT = 0x30,
ERROR_CAPTURE_LATENT   = 0x31,
ERROR_CAPTURE_CANCELLED      = 0x32,
ERROR_CAPTURE_INTERNAL= 0x33,
/* NOOP */
STATUS_NO_OP     = 0xFF,
} _V100_OP_ERROR;
```

hidglobal.com