SP400X Series

SP400X Scanner Printer

Communication Protocols

handheld

Copyright information

Trademark credits

Wi-Fi® is a registered trademark of the Wi-Fi Alliance.

Edition 1: November 2012

Contents

1. Overview

Scope

This document contains information about the communication protocols used with the Handheld SP400X. There are two protocols supported; ASCII and binary. Either communication protocol may be used in development; however the server and device must communicate in like. It is strongly encouraged to use the recommended communication protocol for like applications based upon the intended use.

Reference Documents

Handheld SP400X Series Template Design Software User Guide

Handheld SP400X Series Configuration Software User Guide

Handheld SP400X Series SP400X Scanner Printer - Quick Start Guide

Definitions

Device	The Handheld SP400X
Host	Host system communicating directly with the device
Transport	Means of communication between the host and the device—could be USB or wireless

2. ASCII Communication Protocol

Introduction

This first half of this document describes the layout and usage of the messages used to communicate between the SP400X device and a server application. A message is composed of a general transport header and a payload. The transport header has the same layout for all messages but the payload is specific to each message.

Messages are sent between a device and server over a wireless protocol. This wireless protocol is typically an 802.11 wireless connection using UDP, but in the future this protocol may be extended to other transports as well.

This document also describes the general sequence of the exchange of messages between the server and the device.

Messaging Protocol

The device communicates by sending messages containing a command code and a command-specific payload. This section describes the general message usage; the following section documents the exact message layout and details for each command code.

Message Sequence

Most messages are used in a transaction. The sender sends a request (for example, GetFile) and waits for the expected response (for example, GetFileReply). If a response for the request is received, the transaction is done; otherwise the sender resends the same request a configurable number of times before terminating the transaction.

The receiver sends a reply to the sender at the address defined in the message header. The message header also contains a sequence number which must be maintained in the reply to ensure that out-ofsequence messages do not occur.

FIGURE 2-1 shows the typical sequence of the request and response handling of messages.

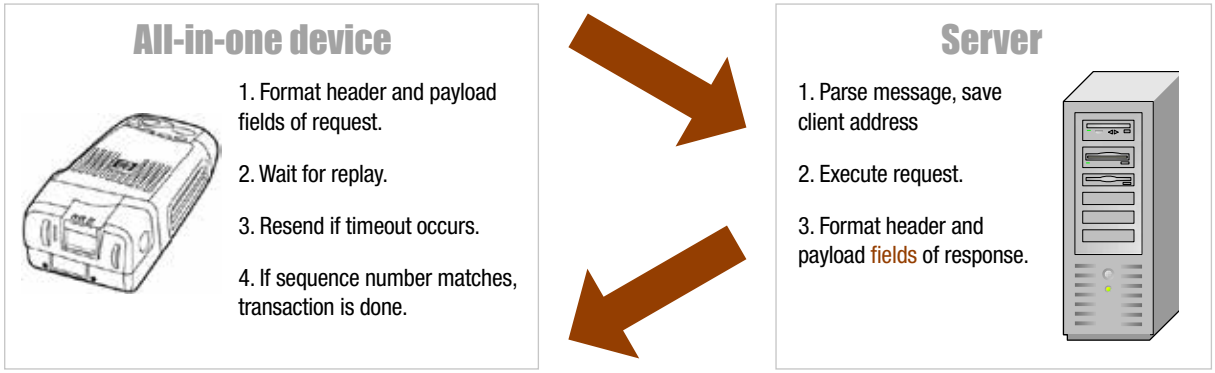


FIGURE 2-1 Sequence of the request and response handling of messages

The amount of time the All-in-One waits for a reply and the number of times the All-in-One resends the transaction are configurable parameters. When a timeout occurs the device sends the request with the same transaction number as the original message. The server application should always respond to incoming requests, even if a reply for that request has already been sent.

Command Table

The following table shows the available command codes that are used in the message transactions.

Most messages are initiated by the device with the exception of GetConfig and SetConfig which are sent by a server or other host to update configuration parameters on the device. The shading in the following table shows how messages are grouped into requests and replies. In general, each request is matched with a specific response, but in the case of ScanData the device sends an additional PrintResultData message to the server after printing the data contained in the ScanDataReply.

TABLE 2-1 Command code descriptions

Command code	Mnemonic	Description
03	GetFile	Get a file from the server – used to transfer firmware, configuration and print template files. This message is used in a loop until all data blocks have been received.
04	GetFileReply	Reply for 03 – block of file data
05	GetServer	Get server address for scan data messages. At startup, the device is configured to send requests to a specific file server. The GetServer command is sent after startup and allows the initial application server to redirect devices to a different application server.
06	GetServerReply	Reply for 05 - application server addresses. May be the same address as the server that is sending this message
07	Heartbeat	Heartbeat / status data. The device sends this message at a configurable interval to allow a server to track how many devices are currently in use and obtain information about the ink and battery level of those devices.
08	HeartbeatReply	Reply for 07 – server status data. The server sends this message to update the device with required version information. It also sends a reboot signal when the server requires the device to reboot.
09	ScanData	Scan data – The device sends this message after scanning a barcode.

TABLE 2-1 Command code descriptions

Command code	Mnemonic	Description
10	ScanDataReply	Reply for 09 – print data or error message sent from server to device after the server has determined print data for the scan data sent in the preceding ScanData message.
11	PrintResultData	Print result and timing data sent by the device to the server after receiving and printing the data sent in the preceding ScanDataReply message. This message does not have a reply so it is considered "best effort" and may be lost. Sending of this message can be switched off with a configuration parameter.
12	SetConfig	Update configuration parameter. Sent by the server to change configuration parameters on the device.
13	SetConfigReply	Reply for 12 – result of configuration update to indicate success or failure of the update to the server.
14	GetConfig	Get configuration parameter. Sent by the server to query current configuration parameters on the device.
15	GetConfigReply	Reply for 14 – value of configuration parameters.

Servers

Logically there are three types of servers that are required to communicate with the SP400X, each performing a different function. These servers can all be housed in one computer or they can each be located in separate computers. The following servers are required to operate the SP400X:

- File server: A storage place for files that can be accessed by the SP400X devices, such as update files. When the SP400X is turned on, it sends a GetFile request to the file server to check for upgrades.
- Registration server: After checking for upgrades, the SP400X sends a request to the registration server to obtain the address of the application server.
- Application server: Runs the application which receives scanned barcodes and replies with print data.

Message Sequence at Device Startup

The following flowchart shows the startup sequence qualitatively. The device first sends a GetFile request to the file server which may initiate a cycle of upgrades. If the file server is not available, the device skips the update cycle and proceeds by sending a GetServer request to the registration server.

To keep firmware up to date, each time a device is turned on, it downloads a file from the file server called "download.txt" that contains file version information. If the device detects that the file server has newer versions of the configuration, firmware, print template definitions, or background images, the device downloads these files from the file server and stores them internally.

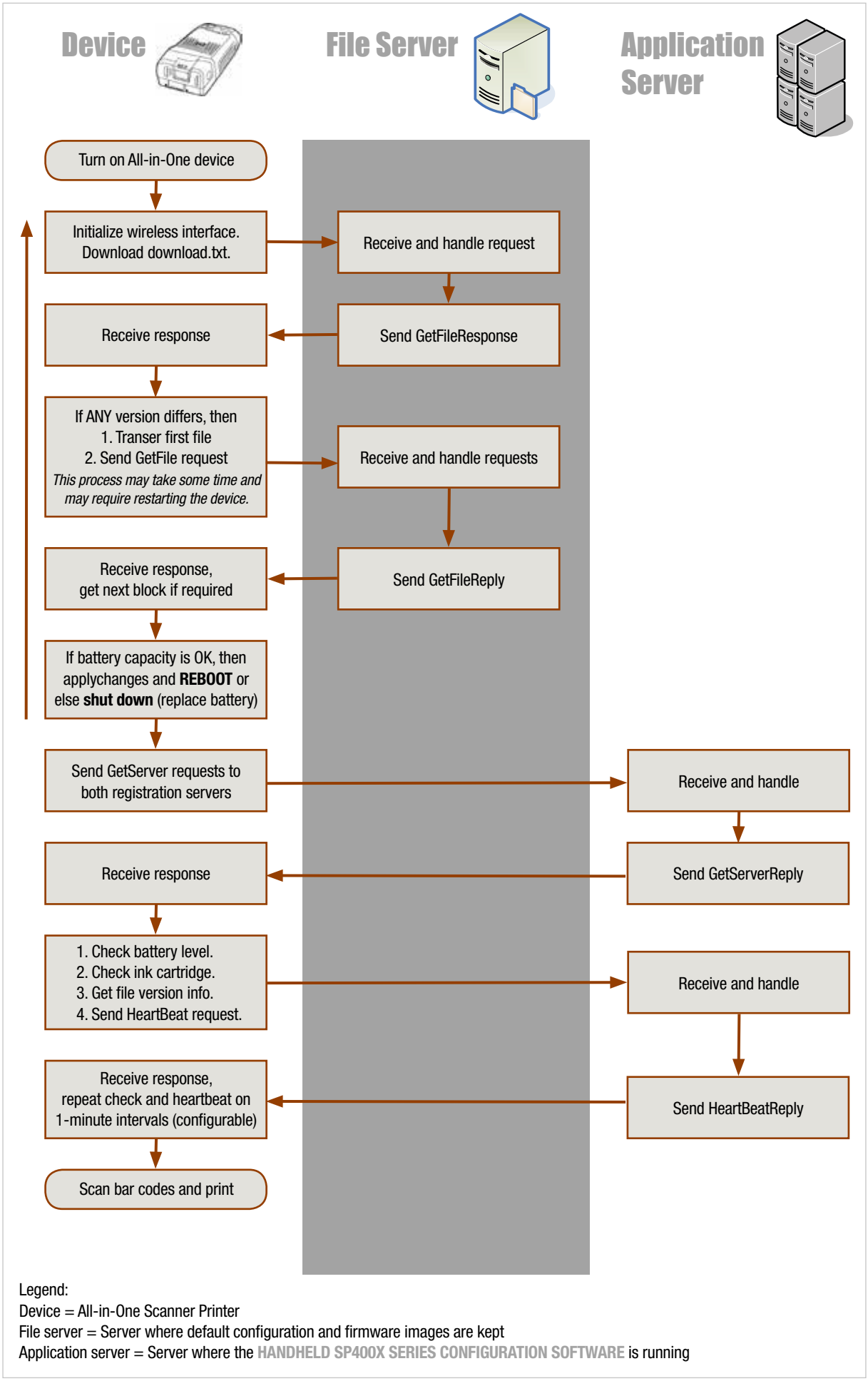
The download.txt file consists of multiple lines of the following format:

0, c:\serverpath\filename, filename, checksum

The first column is reserved and should be set to 0; the second column is the path to the upgrade file on the server; the third column is the file type as it is known to the device and is taken from TABLE 2-16; the final column is the CRC16 checksum of the entire file. Generally, download.txt and the actual files will be generated by a utility supplied by Handheld.

Once the device is fully updated, it sends the registration server a GetServer request. The registration server replies with the address of the application server in the GetServerReply message. The address of the application server can be the same as the registration server, or for load-balancing purposes the addresses of registration and application servers can be different.

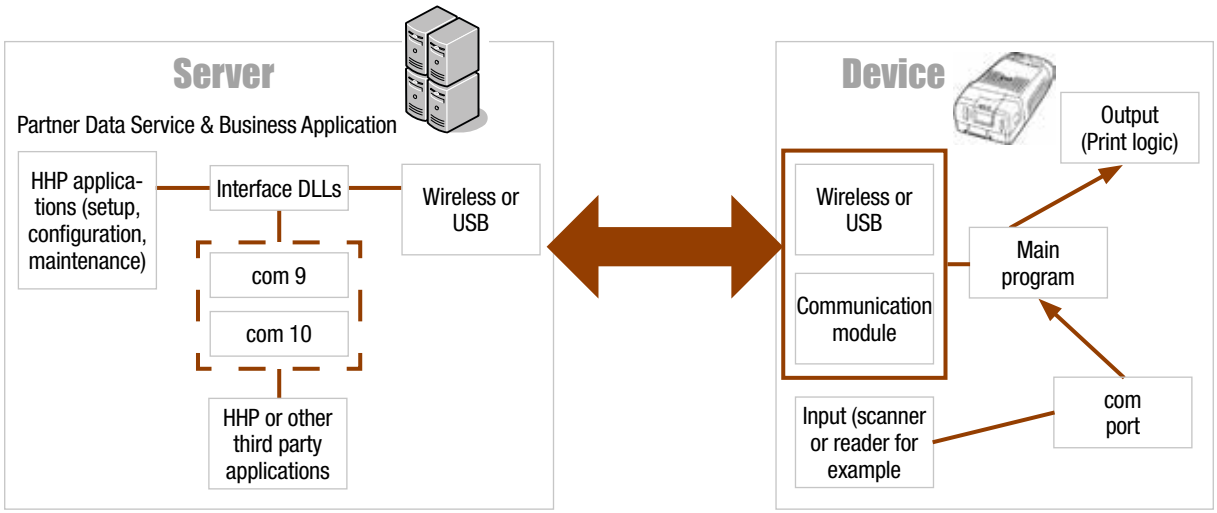
The file server and registration server addresses, which the device sends the initial requests to, are stored in flash. The addresses can be DNS aliases.



Legend:
Device = All-in-One Scanner Printer
File server = Server where default configuration and firmware images are kept
Application server = Server where the HANDHELD SP400X SERIES CONFIGURATION SOFTWARE is running

FIGURE 2-2 Message sequence at device startup

HHP communication interfaces



Message reference

This section of this document details the message fields used in the transactions between the device and server. The individual commands are grouped into logical transactions. Within each transaction the SequenceNumber field must be the same. The device does not handle concurrent transactions; only one transaction may be active at any one time. The sequence number for each transaction has to be different from the previous one; generally they will increase monotonically, but this is not a strict requirement.

Messages are sent in packed ASCII format. All fields, with the exception of binary data fields in the file transfer and scan data messages, consist entirely of printable ASCII characters.

General message header

Each message starts with the message header shown in Table 2-2. The TransportType is a two-byte ID that must be set to **HA**. This allows the receiver to filter out packets that were accidentally sent to the same socket.

TABLE 2-2 Message headers

Field Name	Len	Type	Format	Notes	Sample Value
TransportType	2	String	Justification not required	String identifying type of this packet. Must be set to HA.	HA
CommandCode	2	Integer	Right justified; numeric; zero padded;	Numeric command code from list of valid command codes in TABLE 3-20	02
Device ID	16	String	Left justified;	The device serial number	12345678

TABLE 2-2 Message headers

Field Name	Len	Type	Format	Notes	Sample Value
SourceIPAddress	16	String	Left justified; numeric+dot or other address separators; no imbedded spaces;	IP or other transport address to send the reply to. Can be used for routing inheterogeneous networks.	123.222.111.200
SourcePort	5	Integer	Right justified; numeric; zero padded;	IP or other transport port to send the reply to. Can be used for routing in heterogeneous networks.	50010
PayloadLength	4	Integer	Right justified; numeric; zero padded;	Length of the payload following this header	0325
SequenceNumber	5	Integer	Right justified; numeric; zero padded;	Number to match a request and reply belonging to one transaction	00010
ErrorCode	4	Integer	Right justified; numeric; zero padded;	Error code for reply messages	0000

GetFile and GetFileReply

The GetFile transaction is used to transfer data files from the server to the device. In each transaction the device requests to transfer a specific portion of a file with a given starting block number and length. The server responds with the block number, block length, and the actual data. The position of the file on the server is calculated as BlockSize*BlockNumber. When the end of the file is reached, the server returns a partial block and sets the LastBlockIndicator.

TABLE 2-3 GetFile payload

Field Name	Len	Type	Format	Notes	Sample Value
ApplicationName	20	String	Left justified	Name of application configured in parameter "ApplicationName" (see TABLE 4-22)	Packaging1
BlockNumber	10	Integer	Right justified; numeric; zero padded	Number of block to send, zero-based	0000000000
BlockSize	5			Size of the requested block, currently only 1024 supported	01024
AliasIndicator	1			Indicates whether to use a file alias (1) or an actual file name (0)	1
FileName	255	String	Left justified; alpha numeric;	A file aliases from TABLE 2-16 or the path to a file on the server.	HHPConfig

TABLE 2-4 GetFile payload

Field Name	Len	Type	Format	Notes	Sample Value
ApplicationName	20	String	Left justified	Copy of value from GetFile request	Packaging1
BlockNumber	10	Integer	Right justified; numeric; zero padded	Number of block to send, zero-based	0000000000
BlockSize	5			Size of the requested block, currently only 1024 supported	01024
LastBlockIndicator	1			Set to 1 if this was the last block in the file, otherwise 0	1
BlockLength	5			Actual length of current block, from 1 to BlockSize	00023
FileData	1024	binary	N/A	The actual binary data from file	this is file data

GetServer and GetServerReply

The GetServer transaction is used to obtain information about which application server will handle ScanData requests. When the device is turned on, it sends a GetServer request to the primary and alternate registration servers stored in its internal configuration memory (see PrimaryRegServerDNSName and AlternateRegServerDNSName TABLE 4-22). The application server responds with a GetServerReply containing a server IP address to be used in subsequent ScanData requests. The registration server may chose to send its own address or the address of a different server.

The device stores the addresses returned from the registration servers as application server 1 and application server 2. The device will send requests to application server 1 until a timeout occurs, at which point it switches over to application server 2 for subsequent requests. If another timeout occurs, the device switches back.

TABLE 2-5 GetServer payload

Field Name	Len	Type	Format	Notes	Sample Value
ClientIPAddress	16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address like 127.0.0.1 to send ScanData requests to	10.0.1.2
ClientListenPort-Number	5	Integer	Right justified; numeric; zero padded	Port number to send ScanData requests to	50010
ClientMACAddress	12	String	zero padded; HEX values; UPPERCASE	MAC address of wireless interface used by this message protocol	00ABCDEF1111
ApplicationName	20	String	Left justified	Name of application configured in parameter “ApplicationName” (see TABLE 4-22)	Packaging1

TABLE 2-6 GetServerReply payload

Field Name	Len	Type	Format	Notes	Sample Value
AppServerAddress	300	String	Left justified; alpha numeric; colon and semicolon delimited	The CurrentServerIP and CurrentServerPort values the device should use for ScanData requests.	CurrentServerIP: 100.11.12.130; CurrentServerPort: 12345

Heartbeat and HeartbeatReply

The device periodically sends out a Heartbeat request to the current application server. The server is expected to respond with a matching HeartbeatReply message. If no reply is received, the device switches to the other application server and resends the Heartbeat request. The heartbeat interval in milliseconds is configurable by parameter HeartbeatIntervalMS from Table 4-22 .

The heartbeat request sends battery, ink level, and version information to the server to allow the application server to track the state of the devices.

TABLE 2-7 Heartbeat payload

Field Name	Len	Type	Format	Notes	Sample Value
ClientIPAddress	16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address like 127.0.0.1 to send ScanData requests to	10.0.1.2
ClientListenPort Number	5	Integer	Right justified; numeric; zero padded;	Port number to send ScanData requests to	50010
ClientMACAddress	12	String	zero padded; HEX values; UPPERCASE	MAC address of wireless interfa-ce used by this message protocol	00ABCDEF1111
BatteryLevel Percent	3	Integer	Right justified; numeric; zero padded;	Current battery charge level in percent between 000 and 100	090
InkLabelsPrinted	5			Number of ink labels that have been printed	04000
PrimaryRegServer PortNumber	5	Integer	Right justified; numeric; zero padded;	Port number where this device will send GetServer requests	09101
AlternateReg ServerIPAddress	16	String	Left justified; numeric+dot; no imbedded spaces;	Server address to which this de-vice will send GetServer requests if the primary server becomes unavailable	10.0.1.3
AlternateReg ServerPortNumber	5	Integer	Right justified; numeric; zero padded;	Server port number to which this device will send GetServer requests if the primary server becomes unavailable.	09105
PrimaryServerIP Address	16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address where this device will send ScanData requests	10.0.1.2
PrimaryServerPort Number	5	Integer	Right justified; numeric; zero padded;	Port number where this device will send ScanData requests	09101
AlternateServerIP Address	16	String	Left justified; numeric+dot; no imbedded spaces;	Server address to which this de-vice will send ScanData requests if the primary server becomes unavailable	10.0.1.3
AlternateServer PortNumber	5	Integer	Right justified; numeric; zero padded;	Server port number to which this device will send ScanData requests if the primary server becomes unavailable.	09105

TABLE 2-7 Heartbeat payload

Field Name	Len	Type	Format	Notes	Sample Value
FileVersions	300	String	Left justified; alpha numeric;	A list of pairs of file aliases and their versions delimited by semicolons	HHPConfig:0;HH PSecurity:0;HHP Template: 62344; HHPTemplateIm age:7056;HHPCode-Firmware: 28 769;HHPFPGAFirmware: 13481;H HPImagerFirmware:8999; HHPCommunicationFirmware:17538;

TABLE 2-8 Heartbeat payload

Field Name	Len	Type	Format	Notes	Sample Value
RebootTime-Stamp	17	Date time	YYYYMMDDHHM MSSfff	The server timestamp when all imprinters are to reboot. Imprinter does not keep time, therefore when the value changes from the one currently stored on the device, it will save the new value and reboot the device.	2001010120 3301000

ScanData, ScanDataReply, and PrintResultData

The device sends the ScanData request to the server when a successful scan occurs. The ScanDatarequest contains the actual data received from the scan engine in addition to the type of symbology (such as barcode or data matrix) that was decoded. The device also sends a DuplicateScanIndicator to tell the server that this bar code is identical to the one scanned in the previous transaction. Note that retries within one transaction do not have the DuplicateScanIndicator set.

The server determines what should be printed in response to the scan data and sends the label print instructions to the device. The server also carries out any other action that might take place for the scan data (such as validation or logon).

The ScanDataReply contains two items: the FeedbackCode, which tells the device which of several audiovisual feedback signals to use and whether to print or not, and the PrintData, which contains the label information to be printed. The PrintData field consists of key:value pairs containing the template name to select and the contents of each field of the selected template.

The device also sends a StateInformation field which contains an arbitrary text field sent from the server in a previous ScanDataReply message. When a device is turned on, the StateInformation field will be blank; after the first ScanDataReply, the StateInformation field will contain a copy of the StateInformation of the preceding ScanDataReply.

TABLE 2-9 ScanData payload

Field Name	Len	Type	Format	Notes	Sample Value
DuplicateScan Indicator	1	Bool	Left justified; numeric+dot; no imbedded spaces;	Indicator that states if this barcode is the same as the previously sent transaction (1) or different (0)	0
ScanObject SymbologyType Code	2	Integer	Right justified; numeric; zero padded;	Unique identifier for the type of symbology of the barcode from TABLE 2-17	12
ScanObjectText	160	String	Left justified; alpha numeric;	Scan data exactly as received from imager module	55555555555
StateInformation	50	String	Left justified	Arbitrary state data from the server, sent in a previous ScanDataReply, otherwise blank	COOKIE

TABLE 2-10 ScanDataReply payload

Field Name	Len	Type	Format	Notes	Sample Value
FeedbackCode	2	Integer	Right justified; numeric; zero padded;	The code representing the feedback signal given to a person using the LEDs and Audio speaker (device specific) and the print action to be carried out: 00 = SUCCESS and PRINT 01 = SUCCESS and NO PRINT 02 = FAILURE and NO PRINT 03 = WARNING and PRINT 04 = WARNING and NO PRINT Anything else = reserved for future use, no label will be printed	00
PrintDataText	300	String	Left justified; key value pair separated by colon. Each pair is separated by a semicolon	The data required to print a label. Content is label specific. Order of fields is not important. Colon and semicolon delimited	PrintTemplate Name:MyLabel1; FIELD1:HHG DESKJET;FIEL D2:QUANTITY 9999;
StateInformation	50	String	Left justified	State information from the server. The data sent in this field will be sent back from the device to the server in subsequent ScanData requests	COOKIE

TABLE 2-11 PrintResultData payload

Field Name	Len	Type	Format	Notes	Sample Value
ResultAndTiming	300	String	Left justified; key value pair separated by colon. Each pair is separated by a semicolon	A listing of timing values for each operation on the imprinter for performance measuring (in milliseconds). Timing values are followed by the name of the print template, the return code from the ScanDataReply and the error code of the print operation. See TABLE 2-18.	Scanning:000120; Communicating:000200; Rendering:000088; Appoaching:001230; Printing:000623; Total:002261; PrintTemplateName: MyTemplate; ReturnCode:00000; PrintingReturnCode:0;

SetConfig and SetConfigReply

The application server or any other host on the network can send the SetConfig to update configuration parameters on the device. The device sends the SetConfigReply message back to the address specified in the header of the message. The payload of the SetConfigReply contains a string with error codes for each parameter from the SetConfig request. The ErrorCode field of the header contains 0 if all parameters were set successfully and -1 if any parameter was not set. The payload of the SetConfigReply contains a list of error codes for each parameter that was updated.

TABLE 2-12 SetConfig payload

Field Name	Len	Type	Format	Notes	Sample Value
ConifigKey Values	500	String	Left justified; alpha numeric;	A list of colondelimited configuration parameters from TABLE 4-22 and values. The pairs are delimited with semicolons.	RangerDetectLimitMinMM:180; RangerDetectLimitMaxMM:200;

TABLE 2-13 SetConfigReply payload

Field Name	Len	Type	Format	Notes	Sample Value
ConifigKey Values	500	String	Left justified; alpha numeric;	A copy of the list of configuration parameters from the corresponding SetConfig request and the error code for each parameter. The pairs are delimited with semicolons.	RangerDetectLimitMinMM:0; RangerDetectLimitMaxMM:0;

GetConfig and GetConfigReply

The application server or any other host on the network can send the GetConfig to query configuration parameters on the device. The device sends the GetConfigReply message back to the address specified in the header of the message. The payload of the GetConfigReply contains a string with current values for each parameter from the SetConfig request.

TABLE 2-14 GetConfig payload

Field Name	Len	Type	Format	Notes	Sample Value
ConifigKeys	500	String	Left justified; alpha numeric;	A list of semicolon-delimited configuration parameters from TABLE 4-22	"ESSID;SPPRetryCount;"

TABLE 2-15 GetConfigReply payload

Field Name	Len	Type	Format	Notes	Sample Value
ConifigKey Values	500	String	Left justified; alpha numeric;	A copy of the list of configuration parameters from the corresponding GetConfigReply request and the current values for each parameter. The pairs are delimited with semicolons.	ESSID:GoHHG;SPPRetryCount:2;

File Aliases

The following table contains a list of the file aliases used to reference files stored on the file serverwhich the device can request to download. Each file has a version associated with it which can be queried using the GetVersions command. Each file can then be downloaded by the device using the GetFile command. The server is free to store the actual file in any format and under any physical file name as long as the original file content is returned to the device in the GetFile transaction. Used definable HHP files are created using the sp400 Configuration Software application. Refer to the sp400 Configuration Software User Guide for more information.

The server maintains a version number for each file. The version number can be any printable text string between 1 and 20 characters.

TABLE 2-16 HHP file aliases

Field Alias	Description
HHPCatalog	Catalog file used for performing wireless updates. See "MESSAGE SEQUENCE AT DEVICE STARTUP"
HHPConfig	Default user configuration parameters. See Table 4-22.
HHPSecurity	Security keys (WEP / WPA keys)
HHPTemplate	Label template as supplied by Handheld or generated by the template design utility
HHPTemplateImage	Background image for label template as supplied by Handheld or generated by the template design utility
HHPCodeFirmware	Application firmware code as supplied by Handheld
HHPFPGAFirmware	Firmware for the Field Programmable Gate Array as supplied by Handheld
HHPImagerFirmware	Firmware for the imager engine as supplied by Handheld

TABLE 2-16 HHP file aliases

Field Alias	Description
HHPCommunicationFirmware	Firmware for the communication module as supplied by Handheld
HHPParameterUpdate	System parameter updates as supplied by Handheld

Supported Symbology Types

The following list shows the symbology types supported by the imager module. The device sends the numeric symbology code as part of the ScanData message.

TABLE 2-17 Symbology types

Code	Symbology
01	Australian Post
02	Aztec Code
03	British Post
04	Canadian Post
05	China Post
06	Codabar
07	Codablock F
08	Code 11
09	Code 16K
10	Code 2 of 5
11	Code 49
12	Code128
13	Code39 (3of9)
14	Code93
15	Data Matrix
16	EAN-UCC Composite Codes
17	EAN/JAN-13
18	EAN/JAN-8
19	IATA Code 2 of 5
20	Interleaved 2 of 5
21	Japanese Post
22	Kix (Netherlands)Post
23	Korea Post
24	Matrix 2 of 5
25	MaxiCode
26	MicroPDF417

TABLE 2-17 Symbology types

Code	Symbology
27	MSI
28	PDF417
29	Planet Code
30	Plessey Code
31	PosiCode A and B
32	Postnet
33	QR Code
34	RSS Expanded
35	RSS Limited
36	RSS-14

Timing and Result Data Keys

The following table contains the keys for timing values that are sent in the PrintDataResults message.

TABLE 2-18 Key names for timing values

Field Name	Len	Format	Notes	Sample Value
Scanning	6	Right justified; numeric; zero padded;	Time imager took to process barcode (milliseconds)	000120
Communicating			Time message took to send and get reply (milliseconds	000200
Rendering			Time to render print data string and merge with back-ground image (milliseconds)	000088
Approaching			Time between ready-to-print signal and detection of encoder motion (milliseconds)	001230
Printing			Time to print the label (milliseconds)	000623
Total			Time for the entire cycle from scan trigger to end of print (milliseconds)	002261
PrintTemplateName	1..25	String, left justified Name of Template used in the printing of label	MyTemplate	
ReturnCode	5	Right justified; numeric; zero padded;	Return code from server sent in the ScanDataReply message.	00000
FeedbackCode	2		Feedback code from server sent in the ScanDataReply message.	00
PrintingReturnCode	6		Error code of printing of label Codes are defined in TABLE 5-23.	000000

Wi-Fi Key Management

The device supports the standard WEP and WPA-PSK encryption mechanisms for Wi-Fi® security. The current encryption key can be set with the WifiKey parameter. The device does not return WifiKey when queried to avoid exposing the encryption key to unauthorized hosts on the network.

Key setup

If setting the WifiKey in clear text is not possible, a simple key encryption mechanism can be used to protect keys from public view. To use the encryption mechanism, follow these steps:

1. Set the WifiName parameter to the desired encryption string.

2. Encrypt the WEP or WPA keys to be used on the device with the encryption algorithm shown in "ENCRYPTION ALGORITHM" using the same encryption string.

3. Set the WifiKey1 through WifiKey4 keys to the encrypted values of the actual keys.

4. Set the WifiKeyIndex to select which of the 4 keys to use. Note that the device will automatically cycle through the keys if the selected key does not work.

This mechanism ensures that users without access to this document can set keys without deciphering the actual keys. Unauthorized users with packet sniffers will not be able to easily decipher the actual network keys even if they have access to this document since generally the WifiName parameter will be unknown and the device will not return even if queried.

Note that strong WEP/WPA keys are still essential for Wi-Fi security. For example, there should be no readable text in the keys.

Key file transfer

The file referenced by HHPSecurity (see Table 2-16) contains the Wi-Fi key information as well as the ESSID the device will connect to. This file can be created using the sp400 Configuration Software. Refer to the sp400 Configuration Software User Guide to more information. When the device downloads a new key file as part of the startup process, the contents of the file are saved in non-volatile storage and decrypted using the currently configured WifiName parameter when they are applied.

Encryption algorithm

A C# version of the encryption algorithm is shown below.

```
static string SymmetricalEncrypt(string buffer, string password)
{

  string encryptedBuffer = "";
  int    bufLen = buffer.Length;
  int    passLen = password.Length;
  string p1 = ""; //1 character of password
  string b1 = ""; //1 character of buffer
  string e1 = ""; //1 character encrypted
  int    e  = 0;  //numeric encrypted value

  //check params
  if ( (bufLen == 0) || (passLen == 0) )
    return "";  //return empty string if bad paramaters

  for (int i = 0; i < bufLen; i++)
  {
    //get 1 character at a time of password
    p1 = password.Substring((i+1)% passLen, 1);//off by 1

    //get 1 character at a time of buffer
    b1 = buffer.Substring(i, 1);

    //encrypt values (xor)
    e = Char.ConvertToUtf32(p1, 0) ^ Char.ConvertToUtf32(b1, 0);

    //restrict values
    if (  (e < 32)  //non printable characters
        || (e == 58)//no colon allowed
        || (e == 59)//no semi-colon allowed
       )
    {
      e1 = b1; //no conversion
    }
    else
    {
      e1 = Char.ConvertFromUtf32(e); //convert integer back to a character
    }
    encryptedBuffer += e1; //concat character to string
  }

  return encryptedBuffer;
}
```

3. Binary Communication Protocol

Introduction

This section of the document describes the communication protocol used by Handheld SP400X devices to send configuration data between the host system and a device. The protocol is also used for maintenance and diagnostic purposes. This document does not describe general usage of the devices, it is meant as a reference for application developers who want to write applications to communicate with the device. It can be assumed the general use of the device is to send data (typically via a scan/image capture) to a PC, which then sends back data to be printed by the device.

Protocol

The communication protocol uses fixed-size frames to send commands to the device and responses back to the host. Command and response currently have the same size and format. The optional data for a command or response always immediately follows the command or response frame.

Framing

The frame uses little-endian byte ordering. The command and response frames use the layout specified in TABLE 3-19.

TABLE 3-19 Frame layout

Field	Type	Description
Type	byte	
Mode	byte	
Command	uint16	
Flags	uint32	
Param1	uint32	
Param2	uint32	
Checksum	uint32	

The type field identifies which protocol is being used for the transmission. A printable value (0x20 – 0x7E) in the type field identifies other protocols used with the device. Any other value identifies the protocol described here.

The mode field is currently not used by the Presto protocol and should be set to zero.

The command code is a 16-bit code that identifies the command. TABLE 3-20 describes the different command codes.

The frames include a checksum field which is used to verify the integrity of the frame itself. The checksum is calculated using Fletcher's Checksum algorithm, which is described in RFC1146 and in this Wikipedia page. All frame fields are used to calculate the checksum.

Fletcher's Checksum in C#

```
private UInt32 CalcChecksum(byte[] buf)
{
     UInt32 sum1 = 0xFFFF, sum2 = 0xFFFF;
     int len = buf.Length / 2;// if odd length then last byte is
     ignored
     int i = 0;
     while (len > 0)
     {
          int tLen = (len > 360) ? 360 : len;
          len -= tLen;
          do
          {
          UInt16 d = (UInt16)((buf[i + 1] << 8) + buf[i]);
          sum1 += (UInt32)d;
          sum2 += sum1;
          i += 2;
          } while (--tLen > 0);

          sum1 = (sum1 & 0xffff) + (sum1 >> 16);
          sum2 = (sum2 & 0xffff) + (sum2 >> 16);
     }
     // Second reduction step to reduce sums to 16 bits
     sum1 = (sum1 & 0xffff) + (sum1 >> 16);
     sum2 = (sum2 & 0xffff) + (sum2 >> 16);

     return sum2 << 16 | sum1;
}
```

Data Flow

All transactions such as sending configuration data, querying device status, and so on are initiated by the host. A transaction consists of a command sent from the host to the device and a response sent back from the device to the host. The device echoes the command code in the response—this allows the host to match up requests and replies.

The protocol is strictly sequential; there are no overlapping commands. FIGURE 3-1 shows a typical transaction with both command and response data. Parameter 1 usually contains an error code and parameter 2 usually contains the length of the response data, but there are some exceptions.

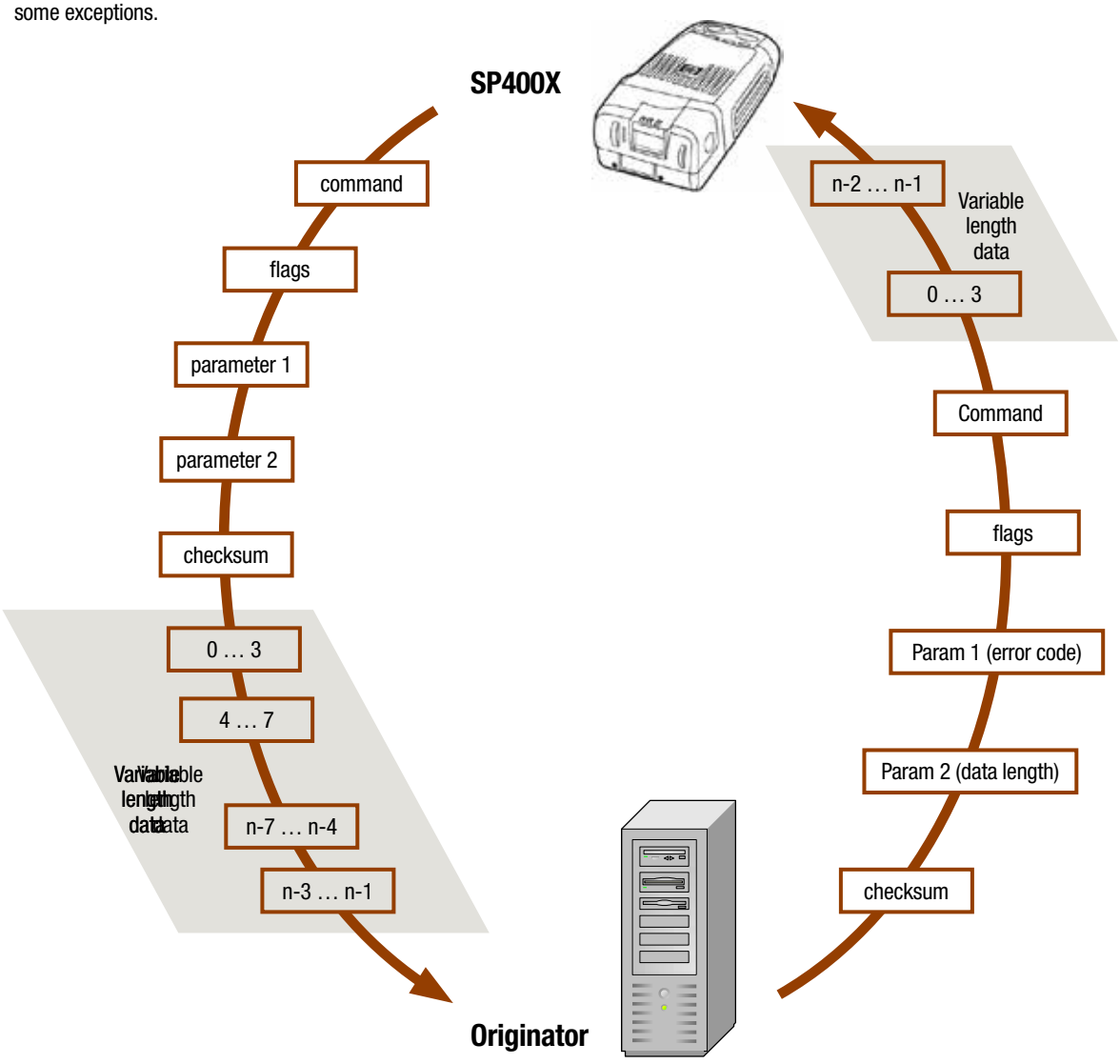


FIGURE 3-1 Typical transaction with both command and response data

The payload data can be any number of bytes. The transport layer may do any necessary padding as long as the protocol handler and the originator each see a stream as shown above.

Command Reference

This section lists the command codes and explains their meaning.

Command Overview Table

The following table lists the command ID in the left-most column. This ID is the number in the command code field of the frame. Each command has specific values for the parameter 1, parameter 2, and data payload fields. For the responses, parameter 1 and parameter 2 fields normally contain error codes and data lengths, respectively. TABLE 3-20 does not cover the Flags field, which is explained in TABLE 3-21 Global Command Flags.

TABLE 3-20 Command overview table

ID	Mnemonic	Dir.	Param 1	Param 2	Data payload
0	Echo	send	any value	any value	-
		recieve	copy of sent value	copy of sent value	-
1	Get Version String	send	-	-	-
		recieve	error code	data length	version string
2	Get Memory	send	addess	data length	-
		recieve	error code	data length	contents of device memory
3	Put Memory	send	adress	data length	contents of device memory
		recieve	error code	-	-
		recieve	error code	-	-
22	Get device configuration	send	start key	num keys	-
		recieve	error code	data length	configuration parameter(s)
23	Set device configuration	send	-	-	configuration strings (key:value;key:value;…)
		recieve	error code	-	-
26	Get utility buffer info	send	-	-	-
		recieve	error code	buffer size	-
30	Get print template names	send	-	-	-
		recieve	error code	data length	Names of print templates
31	Query versions	send	-	-	-
		recieve	error code	data length	version info from all subsystems
64	Apply HHP file	send	-	-	-
		recieve	error code	-	-

Command Flags

The flags field is used for both global flags (upper bits) and command-specific flags (lower bits).

TABLE 3-21 Global Command Flags

Flag	Mask
New protocol as described here (with checksum)	0x80000000
Response data follows frame	0x40000000
Use checksum	0x10000000

Command Details

This section presents a list of all commands with details of their operation and usage.

Echo

The echo command is used mainly to test whether the communication between host and device works. The host sends arbitrary values into the param1 and param2 fields and the device echoes these values back. The device replies with an echo of the Command, Param1, and Param2.

The echo command can be used as a type of heartbeat to periodically check whether the device is in a known state and communicating with the host.

Get Version String

The Get Version String command retrieves a version string from the device. The string contains the date and time of the last compilation as well as the date and time when the FPGA configuration was built. These strings can be used to determine whether a device needs to be upgraded. The device sends a response frame with an error code in parameter 1 and the length of the version string in parameter 2. The version string itself follows the response frame.

Get Memory

The Get Memory command can be used to retrieve the content of arbitrary memory locations on the device. The host sends the starting address of the memory region to retrieve in parameter 1 and the length of the memory region in parameter 2.

The device will respond with a response frame containing an error code in parameter 1 and the length of the memory region in parameter 2. The content of the memory region follows the response frame.

Put Memory

The Put Memory command can be used to write arbitrary values to a memory region on the device.

The host sends the starting address of the memory region to write in parameter 1 and the length of the memory region in parameter 2 followed by the values to be written to device memory.

The device responds with a frame containing an error code in parameter 1.

Get Device Configuration Values

The Get Device Configuration Values command is used to retrieve one or more device configuration parameters from the device. Parameters are returned as one or more consecutive strings containing key=value pairs.

The host sends the index of the first configuration value that will be retrieved in parameter 1 and the number of configuration values that will be retrieved in parameter 2. To enumerate all configuration values the host may continually request a very large number of values and check the number of actually returned values until nothing more is returned.

The device responds with an error code in parameter 1 and the number of returned bytes in parameter 2. The device then sends a stream of key=value pairs delimited by carriage return (0x0D) characters. The end of the stream is indicated by an additional carriage return character. The device might, for example, return “key1=v1\rkey2=v2\r\r”, where \r is the carriage return character. In this case, the parameter 2 in the response frame would be 17.

For a list of configuration parameters refer to TABLE 4-22.

Set Device Configuration Values

The Set Device Configuration Value command is used to set a configuration value(s). The configuration value(s) are sent as key:value; pairs which indicates to the device that the configuration value called “key” should be set to “value”.

NOTE: This command makes use of the “write to flash” flag on the device. This flag can be either 1 or 0 and indicates whether a configuration value should be immediately committed to flash or should be kept in RAM only. The flag should normally always be set to 1 in order to avoid losing any values when the device is turned off. If the host wants to do a batch update of many parameters, it is faster and reduces the wear of the flash memory to set the flag to 0, update a number of values, and set the flag to 1 before updating the final value. When the final value gets updated, all previous changes are also committed to flash since all configuration parameters are committed at once.

The host sends the “write to flash” flag in parameter 1 and the length of the configuration data string in parameter 2. The host then sends the key:value; string(s).

The device responds with an error code in parameter 1.

For a list of configuration parameters refer to TABLE 4-22.

Get Utility Buffer Info

The Get Utility Buffer Info command can be used to retrieve information about the utility buffer, which is a temporary scratch buffer used for certain commands.

The host sends no additional data with this command.

The device responds with the base address of the utility buffer in parameter 1 and the length of the utility buffer in parameter 2.

Get Print Template Names

The Get Print Template Names command is used to retrieve a list of print template names from the device.

The host does not send any additional data with this command.

The device responds with an error code in parameter 1 and the number of returned bytes in parameter 2. The device then sends the print template names as a series of zero-terminated strings.

Query Versions

The Query Versions command can be used to retrieve a series of strings with version information of the different subsystems. The exact layout depends on the subsystems, but generally consists of humanreadable strings.

The host does not send any additional data with this command.

The device responds with an error code in parameter 1 and the number of returned bytes in parameter 2. The device then sends the version information strings as a series of zero-terminated strings.

Strings are not zero terminated. Each of the four version strings (FW, FPGA, Imager FW, and Comm FW) consists of a label, linefeed, carriage return, and the version string followed by a line feed.

Apply HHP File

The Apply HHP File command is used to update a device using a particular HHP file. To use this command the host first transfers the HHP file to the utility buffer using the Get Utility Buffer Info and Put Memory commands. The host then sends the Apply HHP File command to cause the device to process the HHP file.

The device responds with an error code in parameter 1.

4. Configuration Parameters

Introduction

This section identifies and describes the configuration parameters of the sp400 SP400X. These parameters are stored in internal non-volatile storage. They will retain their value when the device is turned off.

Table of Configuration Parameters

TABLE 4-22 Configuration parameters

Key Name	Len	Type	Format	Notes	Sample Value
ESSID	1..50	String	Left justified; Alpha numeric	Extended Service Set ID. The ESSID is the identifying name of a wireless network (or sometimes Access Point).	Blue
WifiKey1 WifiKey2 WifiKey3 WifiKey4	10..64	String; hex	Left justified; hexadecimal digits	Encrypted WEP or WPA-PSK keys. These keys must be encrypted with the WifiName parameter. These are write-only parameter for this protocol.	0123AABBCC
WifiKeyIndex	1	Integer	digit	Which of the four WifiKeyN values to use, range 1-4. Note that the device will cycle through the other keys if the one selected by WifiKeyIndex does not work.	1
WifiName	10..64	String	Left justified, no imbedded spaces	This parameter acts as a decryption key for WifiKey1-4. The encryption algorithm is listed in "Encryption algorithm"	9A77QZ14P5
FileServerIP	0..16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address like 127.0.0.1 of the File server for upgrade files. Empty means don't connect to file server; instead connect to app server right away. Default = 192.168.2.110	10.0.1.100
FileServerPort	5	Integer	Right justified; numeric; zero padded;	Port number of file Server Default= 50010 Min = 4000 Max = 65535	50010
PrimaryRegServerDNSName	0..16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address like 127.0.0.1 of the primary server for scan and print requests Default = 192.168.2.120	10.0.1.100

TABLE 4-22 Configuration parameters

Key Name	Len	Type	Format	Notes	Sample Value
PrimaryRegServerPort	0..5	Integer	Right justified; numeric; zero padded;	Port number of Primary Server Default= 50010 Min = 4000 Max = 65535	50010
AlternateRegServerDNSName	0..16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address like 127.0.0.1 of the alternate server to which the device switches when a timeout occurs	10.0.1.101
AlternateRegServerPort	0..5	Integer	Right justified; numeric; zero padded;	Port number of alternate Server Default= 50010 Min = 4000 Max = 65535	50010
EnableDHCP	1	Bool	numeric;	Will device use DHCP or static IP address 1= True (default) 0= false (use static IP addressing, see StaticIP and SubnetMask parameters)	1
DHCPAcquireTimeoutSeconds	3	Integer	Right justified; numeric; zero padded	Maximum time to wait for a DHCP address. Default=020 Min =001 Max =255	020
StaticIP	0..16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address like 127.0.0.1 of the device when EnableDHCP is set to 0.	10.110.1.1
SubnetMask	7..16	String	Left justified; numeric+dot; no imbedded spaces;	A 32-bit subnet mask used when EnableDHCP is set to 0	255.255.255.0
DefaultGateway	7..16	String	Left justified; numeric+dot; no imbedded spaces;	Default IP gateway when EnableDHCP is set to 0	10.110.1.1
PrimaryDNS	7..16	String	Left justified; numeric+dot; no imbedded spaces;	Primary DNS server address when EnableDHCP is set to 0.	10.110.1.1
SecondaryDNS	7..16	String	Left justified; numeric+dot; no imbedded spaces;	Secondary DNS server address when EnableDHCP is set to 0.	10.110.1.1
DeviceSerialNumber	10	String	Left justified; Alpha numeric	UNIQUE serial number given to device	HGAI012345
DeviceName	1..50			Free form text to identify device, model or other user data. Not expected to be unique per device.	Bobs Labeler

TABLE 4-22 Configuration parameters

Key Name	Len	Type	Format	Notes	Sample Value
HeartBeatIntervalMS	10	Integer	Right justified; numeric; zero padded;	Number of milliseconds between each heart beat message that will be send Default=0000060000 Min =0000000100 Max =0010000000	0000060000
BarcodeReadTimeoutMS				Number of milliseconds before Engine stops searching for a barcode. (note that it will start again if the proximity detector is triggered) Default=0000000500 Min =0000000100 Max =0000100000	0000000500
IdleAfterPrintDataReceiveTimeoutMS				Number of milliseconds user has to start printing after print data was received Default=0000000500 Min =0000000001 Max =0000100000	0000000500
WifiConnectionTimeoutMS				Number of milliseconds to wait for the Wi-Fi network connection to initialize. Default=0000010000 Min =0000000100 Max =0010000000	0000001000
ReTriggerDelayAfterPrintMS				Number of milliseconds to wait between print-done and enabling scanner. Default=0000000000 Min =0000000000 Max =0000005000	0000002000
PrintIncompleteTimeoutMS				Number of milliseconds to wait to for user to finish printing before it times out. Default=0000005000 Min =0000000500 Max =0000100000	0000002000
WifiSecurityMode	1	Integer	Integer numeric;	Number identifying which of several Wi-Fi security modes is used. 0= no security 1= WEP64 2= WEP128 3= (Default) WPA PSK 4= LEAP (user/pwd) 5= WPA PSK128 (Symbol compatibility)	3
UserName	1..16	String	Left justified;	ID used for Wi-Fi security protocols that require logon.	Tony

TABLE 4-22 Configuration parameters

Key Name	Len	Type	Format	Notes	Sample Value
UserPassword				Password used for Wi-Fi security protocols that require logon	Pass
UDPListenPort	5	Integer	Right justified; numeric; zero padded;	UDP port number the device is listening on for incoming messages. Default=50010 Min =04000 Max =65535	50010
SPPAckTimeoutMS	5	Integer	Right justified; numeric; zero padded;	Timeout for replies to messagesnof this protocol. If the time out expires the device retries SPPRetryCount number of times. Default=100 Min =1 Max =65535	00100
WifiInitMode	1	Integer	numeric;	Initialization flag for the Wi-Fi interface. Automatically changes back to 0 after initialization (bit field, combine bits below) 0= do nothing (default) 1= initialize general parameters (IP addresses, for example) using current settings 2= initialize Wi-Fi security parameters using current settings	1
RangerDetectTimeoutMS	10	Integer	Right justified; numeric; zero padded;	Number of milliseconds to wait for an object to be detected, also used for general housekeeping time slice: Default=0000002000 Min =0000000100 Max =0000100000	0000002000
RangerDetectLimitMinMM	10	Integer	Right justified; numeric; zero padded;	Minimum number of millimeters away from object to activate the imager. Default=0000000100 Min =0000000050 Max =0000000400	0000000100
RangerDetectLimitMaxMM				Maximum number of millimeters away from object to activate the imager. Default=0000000225 Min =0000000050 Max =0000000400	0000000225
ImagerIniCmds	1..500	String	Left justified;	Commands to initialize the imager separated by forward slash character “/”.	128DLY0/SUFCA2/SUFBK2995C800D0A

TABLE 4-22 Configuration parameters

Key Name	Len	Type	Format	Notes	Sample Value
SPPConfig	5	Integer	Right justified; numeric; zero padded; this is a bit field in decimal	Flags describing additional protocol options (Bit field, combine bits below) Default value is 13 (0x0c) 1 = ASCII (default) 2 = Binary 4 = enable heartbeat (default) 8 = send print-done message (default)	00013
SPPRetryCount	2	Integer	Right justified; numeric; zero padded;	Number of times to retry sending a message to the server default value is 2	02
LoggingOptions	5	Integer	Right justified; numeric; zero padded; this is a bit field in decimal	Bit field with flags to select different logging options 1 = log startup phase 2 = log scan and print 4 = log additional warnings 8 = log informational data	01
ApplicationName	20	String	Left justified	Name of application in use on this device	Packaging1
Non persistent (RAM only) parameters that will get reset the next time it is turned off and on					
CurrentServerIP	0..16	String	Left justified; numeric+dot; no imbedded spaces;	Dot notation of IP address like 127.0.0.1 of the current server it is communicating with. Defaults to Primary-ServerIP when turned on.	10.0.1.100
CurrentServerPort	0..5	Integer	Right justified; numeric; zero padded;	Port number of current server it is communicating with. Defaults to PrimaryServerIP when turned on.	50010

5. Error Codes

Introduction

This section identifies and describes the error codes available from the message and command responses as defined in the ASCII and binary protocols of the sp400 SP400X.

Table of Error Codes

TABLE 5-23 Error codes

Error name	Value
Generic Errors	
PE_NO_ERROR	0
PE_NOT_FOUND	- 49999
PE_BUSY	- 49998
PE_TIMEOUT	- 49997
PE_NOT_INITIALISED	- 49996
PE_TYPE_MISMATCH	- 49995
PE_BAD_PARAM_0	- 49994
PE_BAD_PARAM_1	- 49993
PE_BAD_PARAM_2	- 49992
PE_BAD_PARAM_3	- 49991
PE_BAD_PARAM_4	- 49990
PE_BAD_PARAM_5	- 49989
PE_BAD_PARAM_6	- 49988
PE_BAD_PARAM_7	- 49987
PE_BAD_PARAM_8	- 49986
PE_NO_TIMER	- 49985
PE_OS_ERROR	- 49984
PE_NO_MUTEX	- 49983
PE_NOT_OWNER	- 49982
PE_WRITEONLY	- 49981
PE_DOOR_OPEN	- 49980
PE_EXISTS	- 49979
PE_MICCI_COM_ERROR	- 49978
PE_NOT_CONNECTED	- 49977

TABLE 5-23 Error codes

Error name	Value
PE_CHECKSUM	- 49976
PE_DOOR_CLOSE	- 49975
PE_ASYNC	- 49974
PE_OUT_OF_RANGE	- 49973
PE_SHUTTING_DOWN	- 49972
General Communication Errors	
PE_ACK_TIMEOUT	- 39999
PE_BAD_UART_NUM	- 39998
PE_BAD_BAUD	- 39997
Wifi Module Errors	
PE_DPAC_CMD_ERROR	- 38999
PE_DPAC_UNKNOWN_RESPONSE	- 38998
Imager errors	
PE_IMAGER_ERROR	- 37999
PE_IMAGER_NAK	- 37998
PE_IMAGER_ENQ	- 37997
PE_IMAGER_FW_DOWNLOAD_FAILED	- 37996
PE_IMAGER_INIT_ERROR	- 37995
PE_IMAGER_COMM_PRES_ERROR	- 37994
PE_IMAGER_COMM_TRIGG_ERROR	- 37993
Bluetooth Module Errors	
PE_BT_MODULE_ERROR	- 36999
PE_BT_UNKNOWN_ERROR	- 36998
PE_BT_NO_BUFFER	- 36997
PE_BT_NO_CONNECTION	- 36996
PE_BT_UNABLE_TO_SEND	- 36995
Printing Errors	
PE_PRINT_INCOMPLETE	- 35999
PE_PRINT_TIMEOUT	- 35998
PE_PRINT_CANCEL	- 35997
PE_PRINT_NO_IMAGE_TEMPLATE	- 35996
PE_PRINT_DIR_CHANGE	- 35995
PE_PRINT_BIDIRECTION_NOT_ENABLE	- 35994
PE_PRINT_SERVER_NO_PRINT	- 35993
PE_PRINT_BARRACUDA_ERROR	- 35992

TABLE 5-23 Error codes

Error name	Value
Ranger Errors	
PE_RANGER_TIMEOUT	- 34999
Scan Print Protocol Errors	
PE_TRANSACTION_ID_MISMATCH	- 33999
Print Information Errors	
PE_PI_IMAGE_NOT_FOUND	- 32999
PE_PI_TEMPLATE_NOT_FOUND	- 32998
PE_PI_KEY_NOT_FOUND	- 32997
PE_PI_SERVER_ERROR	- 32996
PE_PI_UI_SERVER_FEEDBACK_ERROR	- 32995
PE_PI_NUM_MERGES_EXCEEDED	- 32994
Flash Errors	
PE_BLOCK_OUT_OF_RANGE	- 31999
PE_BAD_BLOCK_LINK	- 31998
PE_BAD_BLOCK_NUMBER	- 31997
PE_BAD_IMAGE_NUMBER	- 31996
PE_OUT_OF_SPACE	- 31995
PE_IMAGE_TABLE_OVERFLOW	- 31994
PE_IMAGE_TOO_LARGE	- 31993
PE_MAGIC_NUMBER_CORRUPT	- 31992
PE_DUPLICATE_HEADER	- 31991
PE_BAD_FLASH_OFFSET	- 31990
PE_FLASH_WRITE_ERROR	- 31989
PE_EPCS_CORRUPT	- 31988
PE_EPCS_LENGTH_MISMATCH	- 31987
PE_EPCS_CONFIG_NOT_FOUND	- 31986
PE_FLASH_INFO_ERROR	- 31985
PE_FLASH_CORRUPT	- 31984
Service Station Errors	
PE_SERVICE_STATION_LIMIT_SWITCH_ERROR	- 30999
PE_SERVICE_STATION_OPEN_TIMEOUT	- 30998
PE_SERVICE_STATION_CLOSE_TIMEOUT	- 30997
PE_SERVICE_STATION_OPEN_PRINT_DATA_WAIT_TIMEOUT	- 30996
PE_SERVICE_STATION_IMAGE_CREATION_TIMEOUT	- 30995
PE_SERVICE_STATION_IDLE_UPDATE_TIMEOUT	- 30994

TABLE 5-23 Error codes

Error name	Value
PE_SERVICE_STATION_ENTER_PRINT_READY_TIMEOUT	- 30993
Pen Errors	
PE_PEN_INTERCONNECT_ERROR	- 29999
PE_PEN_INVALID	- 29998
PE_PEN_FUSE_ERROR	- 29997
PE_PEN_UNKNOWN_LEVEL	- 29996
PE_PEN_ERROR	- 29995
Font Errors	
PE_FONT_FILE_CORRUPT	- 28999
PE_FONT_NOT_FOUND	- 28998
PE_FONT_CHAR_NOT_FOUND	- 28997
SMBus and I2C Communication Errors	
PE_SMBUS_COMM_ERROR	- 27999
PE_I2CBUS_COMM_ERROR	- 27998